Proceedings of The 13th International Conference on Parsing Technologies

IWPT-2013

November 27–29, 2013

Nara, Japan

Contents

Front matter . . . . **iii**

Discontinuous Parsing with an Efficient and Accurate DOP Model **7**
Andreas van Cranenburgh and Rens Bod

An Efficient Typed Feature Structure Index: Theory and Implementation **17**
Bernd Kiefer and Hans-Ulrich Krieger

Unsupervised Learning of Bilingual Categories in Inversion Transduction Grammar Induction **26**
Markus Saers and Dekai Wu

Comparative Evaluation of Argument Extraction Algorithms in Discourse Relation Parsing **36**
Evgeny Stepanov and Giuseppe Riccardi

Improved Chinese Parsing Using Named Entity Cue **45**
Dongchen Li, Xiantao Zhang and Xihong Wu

Improving a symbolic parser through partially supervised learning **54**
Eric De La Clergerie

On Different Approaches to Syntactic Analysis Into Bi-Lexical Dependencies. An Empirical Comparison of Direct, PCFG-Based, and HPSG-Based Parsers **63**
Angelina Ivanova, Stephan Oepen, Rebecca Dridan, Dan Flickinger and Lilja vrelid

Generalization of Words for Chinese Dependency Parsing **73**
Xianchao Wu, Jie Zhou, Yu Sun, Zhanyi Liu, Dianhai Yu, Hua Wu and Haifeng WANG

Dynamic-oracle Transition-based Parsing with Calibrated Probabilistic Output . . **82**
Yoav Goldberg

Dependency Structure for Incremental Parsing of Japanese and Its Application . . **91**
Tomohiro Ohno and Shigeki Matsubara

Active Learning for Dependency Parsing by A Committee of Parsers **98**
Saeed Majidi and Gregory Crane

Development of Amharic Grammar Checker Using Morphological Features of Words and N-Gram Based Probabilistic Methods **106**

Aynadis Temesgen and Yaregal Assabie

LCFRS binarization and debinarization for directional parsing **113**
Wolfgang Maier

Towards Fully Lexicalized Dependency Parsing for Korean **120**
Jungyeul Park, Daisuke Kawahara, Sadao Kurohashi and Key-Sun Choi

Document Parsing: Towards Realistic Syntactic Analysis **127**
Rebecca Dridan and Stephan Oepen

Neo-Davidsonian Semantics in Lexicalized Grammars **134**
Petr Homola

Effective Parsing for Human Aided NLP Systems **141**
Naman Jain and Sambhav Jain

Preface

Welcome to the 13th International Conference on Parsing Technologies in the beautiful ancient city of Nara, Japan. This conference continues the tradition of biennial conferences organized by SIGPARSE, ACL's Special Interest Group on Parsing. The first International Workshop on Parsing Technologies (IWPT) took place in 1989 in Philadelphia, and was followed by successful biennial workshops in Cancun ('91); Tilburg ('93); Prague ('95); Boston ('97); Trento (2000); Beijing (2001); Nancy (2003); and Vancouver (2005), after which the name was changed from 'workshop' to 'conference' , while retaining the abbreviation 'IWPT'. Subsequent IWPT conferences were held in Prague (2007), Paris (2009) and Dublin (2011). Over time, these conferences have developed more and more as the primary specialized forum for research on natural language parsing.

Based on contributions to IWPT workshops and conferences, five books on parsing have been published of which the latest one, based on IWPT 2007 and '09, was published by Springer in 2010 as *Trends in Parsing Technology*, edited by Harry Bunt, Paola Merlo and Joakim Nivre. Selected revised papers from IWPT 2011 have been accepted for publication in a special issue of the *Journal for Logic and Computation* which is scheduled to appear in April 2014; see `http://logcom.oxfordjournals.org/content/early/recent` for online prepublication of these papers.

This year we received a total of 28 valid submissions, Of these, 9 were accepted as long papers and 8 as short papers. All accepted papers are published in these proceedings and presented at the conference either as a long talk or as a short talk and a poster

In addition to the contributed papers, IWPT 2013 as usual features invited talks on topics relevant to natural language parsing. This year we are delighted to welcome two distinguished Japanese invited speakers: Jun'ichi Tsujii (Microsoft Research) and Taro Watanabe (National Institute for Communication Technologies NiCT).

Organizing IWPT 2013 would not have been possible without the dedicated work of a number of people. We would like to thank the local organizing committee, chaired by Yuji Matsumoto and including Kevin Duh and Sorami Hisamoto, for an outstanding job in taking care of the local and practical organization of this conference. We also thank Kai Zhao for his excellent work in setting up and maintaining the IWPT 2013 website. We are extremely grateful to the members of the program committee, who spent precious time on reviewing submitted papers in the middle of the summer holiday season. Many thanks are also due to the sponsors whose support helped to make IWPT 2013 possible: NAIST - the Nara Institute of Science and Technology, Springer Publishers, and the Nara Vistors Bureau. Finally, we like to thank the directors of the NAIST, NiCT, and NTT Communication Sciences laboratories for the post-conference visit to these labs that the IWPT 2013 participants are all invited to.

Enjoy the conference and its setting!

Harry Bunt	Khalil Sima'an and Liang Huang
General Chair	Program Chairs

Organizers:

General Chair	Harry Bunt (Tilburg University, The Netherlands)

Program Chairs

Chair	Khalil Simaan (University of Amsterdam, Netherlands)
Co-chair	Liang Huang (City University of New York, USA)

Local Organization Committee

Chair	Yuji Matsumoto (Nara Institute of Science and Technology, Japan)
Co-chair	Kevin Duh (Nara Institute of Science and Technology, Japan)
Co-chair	Sorami Hisamoto (Nara Institute of Science and Technology, Japan)

Program Committee:

Srinivas Bangalore (AT&T Labs-Research, USA)
Philippe Blache (CNRS/Provence University, France)
Harry Bunt (Tilburg University, Netherlands)
Aoife Cahill (Educational Testing Services, USA)
John Carroll (University of Sussex, UK)
Eugene Charniak (Brown University, USA)
David Chiang (USC Information Sciences Institute, USA)
Eric Villemonte de la Clergerie (INRIA, Rocquencourt, France)
Jason Eisner (Johns Hopkins University, USA)
Jennifer Foster (Dublin City University, Ireland)
Josef van Genabith (Dublin City University, Ireland)
Yoav Goldberg (Bar Ilan University, Israel)
Carlos Gomez-Rodriguez (University of La Corua, Spain)
Nizar Habash (Columbia University)
James Henderson (University of Geneva, Switzerland)
Julia Hockenmaier (University of Illinois at Urbana-Champaign, USA)
Zhongqiang Huang (BBN Technologies, USA)
Mark Johnson (Macquarie University, Australia)
Ronald Kaplan (Nuance Communications, USA)
Daisuke Kawahara (Kyoto University, Japan)
Martin Kay (Stanford University, USA)
Terry Koo (Google Inc, USA)
Sandra Kuebler (Indiana University, USA)
Marco Kuhlmann (Uppsala University, Sweden)
Jonas Kuhn (University of Stuttgart, Germany)
Sadao Kurohashi (Kyoto University, Japan)
Alon Lavie (Carnegie-Mellon University, USA)
Yuji Matsumoto (Nara Institute of Science and Technology, Japan)
David McClosky (IBM T. J. Watson Research Center, USA)
Ryan McDonald (Google Inc, USA)
Yusuke Miyao (University of Tokyo, Japan)
Mark-Jan Nederhof (University of St. Andrews, UK)
Joakim Nivre (University of Uppsala, Sweden)
Stephan Oepen (University of Oslo, Norway)

Gerald Penn (University of Toronto, Canada)
Slav Petrov (Google Inc, USA)
Brian Roark (Google Inc, USA)
Kenji Sagae (Institute for Creative Technologies, USA)
Djam Seddah (University Paris-Sorbonne)
Vijay Shanker (University of Delaware, USA)
Noah Smith (Carnegie Mellon University, USA)
Mark Steedman (University of Edinburgh, UK)
Ivan Titov (Saarland University, Germany)
Reut Tsarfaty (Uppsala University, Sweden)
Dekai Wu (Hong Kong University of Science and Technology, China)
Hao Zhang (Google Inc, USA)
Yi Zhang (Saarland University, Germany)
Yue Zhang (Singapore University of Technology and Design, Singapore)

Invited Speaker:

Junichi Tsujii, Microsoft Research (MSR)
Taro Watanabe, National Institute for Communication Technnologies (NICT), Japan

Invited Talks

Semantic Processing – The Depth and Width

Jun'ichi Tsujii
Microsoft Research

Abstract
Accumulation of large amounts of structured or semi-structured facts and knowledge, such as FreeBase, Yago, Wikipedia, etc., will make semantics-based Natural Language Processing plausible in practical application settings. In this talk, I would like to address crucial problems involved in wide and deep semantic processing. While the ability of making inferences would be crucial, logically rigorous frameworks, which are required to solve problems intelligently, would not be what we need in Natural Language Processing for semantics-based information access. What we need to do first is to make linkages between textual expressions and structured facts/knowledge. We can extend word-centered concepts such as synonymy to more general concepts such as paraphrases of larger units of linguistic expressions such as phrases, clauses, etc. We discuss several on-going projects in this direction of semantic processing.

Grammar Induction for Machine Translation

Taro Watanabe
National Institute for Communication Technnologies
Kyoto, Japan

Abstract
Unsupervised methods for inducing bilingual correspondence are important components in machine translation. In this talk I would like to focus on automatically inducing grammatical information from bilingual data. Based on non-parametric Bayesian methods, first, an inversion transduction grammar is learnt though binary branching in two languages. The recursive splitting strategy is combined with the hierarchical Pitman-Yor process to memorize all the granularities of phrasal rules. Next, I would discuss part-of-speech tag induction given dependency trees in one language to improve the performance of machine translation. In particular the monolingual infinite tree model is extended to a bilingual scenario by emitting a source word with its aligned target words, either jointly or independently, from each hidden state of a source-side dependency tree. Finally, I would like to present some work in progress for machine translation.

Discontinuous Parsing with an Efficient and Accurate DOP Model

Andreas van Cranenburgh[*†]
[*]Huygens ING
Royal Dutch Academy of Science
P.O. box 90754, 2509 LT, The Hague, The Netherlands
`andreas.van.cranenburgh@huygens.knaw.nl`

Rens Bod[†]
[†]Institute for Logic, Language and Computation
University of Amsterdam
Science Park 904, 1098 XH, The Netherlands
`rens.bod@uva.nl`

Abstract

We present a discontinuous variant of tree-substitution grammar (TSG) based on Linear Context-Free Rewriting Systems. We use this formalism to instantiate a Data-Oriented Parsing model applied to discontinuous treebank parsing, and obtain a significant improvement over earlier results for this task. The model induces a TSG from the treebank by extracting fragments that occur at least twice. We give a direct comparison of a tree-substitution grammar implementation that implicitly represents all fragments from the treebank, versus one that explicitly operates with a significant subset. On the task of discontinuous parsing of German, the latter approach yields a 16 % relative error reduction, requiring only a third of the parsing time and grammar size. Finally, we evaluate the model on several treebanks across three Germanic languages.

1 Introduction

A Probabilistic Context-Free Grammar (PCFG) extracted from a treebank (Charniak, 1996) provides a simple and efficient model of natural language syntax. However, its independence assumptions are too strong to form an accurate model of language syntax. A tree-substitution grammar (TSG) provides a generalization of context-free grammar (CFG) that operates with larger chunks than just single grammar productions. A probabilistic TSG can be seen as a PCFG in which several productions may be applied at once, capturing structural relations between those productions.

Tree-substitution grammars have numerous applications. They can be used for statistical parsing, such as with Data-Oriented Parsing (DOP; Scha, 1990; Bod et al., 2003; Sangati and Zuidema, 2011) and Bayesian

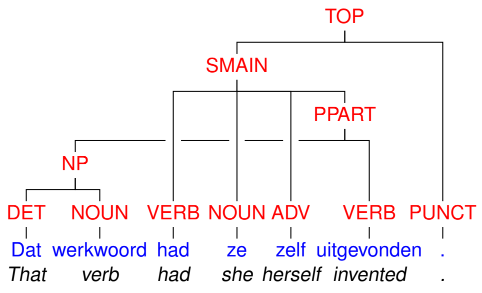


Figure 1: A tree from the Dutch Alpino treebank (van der Beek et al., 2002). PPART is a discontinuous constituent (indicated with crossing branches) due to its extraposed NP object. Translation: *She had invented that verb herself.*

TSGs (Post and Gildea, 2009; Cohn et al., 2010; Shindo et al., 2012). Other applications include grammaticality judgements (Post, 2011), multi-word expression identification (Green et al., 2011), stylometry and authorship attribution (Bergsma et al., 2012; van Cranenburgh, 2012c), and native language detection (Swanson and Charniak, 2012).

An orthogonal way to extend the domain of locality of PCFG is to employ a formalism that produces richer derived structures. An example of this is to allow for producing trees with discontinuous constituents (cf. figure 1 for an example). This can be achieved with (string rewriting) Linear Context-Free Rewriting Systems (LCFRS; Vijay-Shanker et al., 1987). Kallmeyer and Maier (2010, 2013) use this formalism for statistical parsing with discontinuous constituents.

The notion of discontinuous constituents in annotation is useful to bridge the gap between the information represented in constituency and dependency structures. Constituency structures capture the hierarchical structure of phrases—which is useful for identifying re-

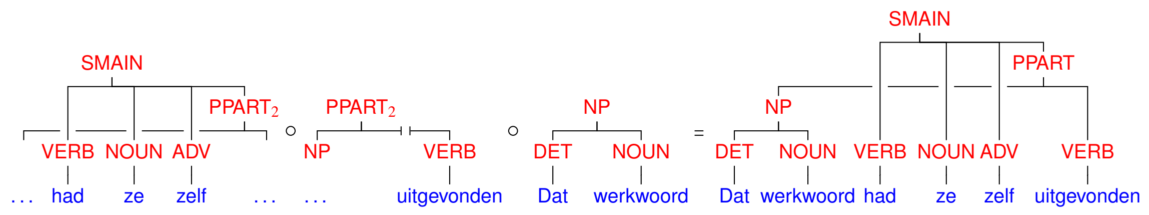


Figure 2: A discontinuous tree-substitution derivation of the tree in figure 1. Note that in the first fragment, which has a discontinuous frontier non-terminal, the destination for the discontinuous spans is marked in advance, shown as ellipses.

usable elements; discontinuous constituents extend this to allow for arbitrary long-distance relations that may arise due to such phenomena as extraposition and word-order freedom. The essential difference between traditional phrase-structure trees and discontinuous ones is that the former is a two-dimensional (planar) structure of a one-dimensional surface form, while the latter allows for a higher dimensional structure. This can be contrasted with the alternative of introducing artificial empty categories, which encode the same information but in the labels instead of the tree structure.

The two approaches of tree-substitution and discontinuity have been synthesized into a discontinuous all-fragments grammar (van Cranenburgh et al., 2011; van Cranenburgh, 2012a) defined implicitly through a reduction (Goodman, 2003). The present paper extends this work. We present a grammar transformation to parse with an arbitrary discontinuous TSG and present results with this new implementation, using TSGs induced by extracting fragments from treebanks. Our method outperforms previous results for discontinuous constituency parsing.

2 Grammar formalisms

In this section we formulate how a discontinuous Tree-Substitution Grammar can be implemented using a Linear Context-Free Rewriting System as the base grammar.

2.1 Linear Context-Free Rewriting Systems

LCFRS can be seen as the discontinuous equivalent of CFG, and its probabilistic variant can be used as a discontinuous treebank grammar. LCFRS generalizes over CFG by rewriting a fixed number of strings at a time for each non-terminal. This number, the measure of discontinuity in a constituent, is called the fan-out. A CFG is an LCFRS with a maximum fan-out of 1. In this paper we use the simple RCG notation (Boullier, 1998) for LCFRS. We will define a restricted variant that operates on unary and binary productions.

A binarized, string-rewriting LCFRS is a 4-tuple $G = \langle N, T, P, S \rangle$. N and T are disjoint finite sets of non-terminals and terminals, respectively. A function $\varphi : N \to \{1, 2, \dots, \}$ specifies the unique fan-out for every non-terminal symbol. S is the distinguished start symbol with $S \in N, \varphi(S) = 1$. We assume a set V of variables of the form b_i and c_i with $i \in \mathbb{N}$. P is a finite set of productions, which come in three forms:

$$A(\alpha_1, \dots, \alpha_{\varphi(A)}) \to B(b_1, \dots, b_{\varphi(B)})\ C(c_1, \dots, c_{\varphi(C)})$$
$$A(\alpha_1, \dots, \alpha_{\varphi(A)}) \to B(b_1, \dots, b_{\varphi(B)})$$
$$D(t) \to \varepsilon$$

where $A, B, C, D \in N$, $\alpha_i \in V^*$ for $1 \leqslant i \leqslant \varphi(A_i)$, $t \in T$, and $\varphi(D) = 1$.

Productions must be *linear*: if a variable occurs in a production, it occurs exactly once on the left hand side (LHS), and exactly once on the right hand side (RHS). A production is *ordered* if for any two variables x_1 and x_2 occurring in a non-terminal on the RHS, x_1 precedes x_2 on the LHS iff x_1 precedes x_2 on the RHS. A production can be *instantiated* when its variables can be bound to spans such that for each component α_i of the LHS, the concatenation of its terminals and bound variables forms a contiguous span in the input. In the remainder we will notate discontinuous non-terminals with a subscript indicating their fan-out.

LCFRS productions may be induced from a discontinuous tree, using a procedure described in Maier and Søgaard (2008). We extend this procedure to handle frontier non-terminals, i.e., non-terminals that do not dominate terminals or non-terminals, because they are part of a tree fragment extracted from a treebank.

Given a discontinuous tree, we extract a grammar production for each non-leaf non-terminal node. The node itself forms the LHS non-terminal, and the non-terminals that are immediately dominated by it forms the RHS. The yield of each node is converted to a sequence of indices reflecting sentence order; this includes the spans of any frontier non-terminals. For each span in the yield, identified as a maximally continuous range in the sequence of indices, a variable is introduced. The variables become the arguments of the LHS and RHS non-terminals, ordered as in the original yield. For the RHS non-terminals, each argument is a single variable. The arguments to the LHS non-terminal consist of a tuple of one or more variables corresponding to consecutive ranges of the sequence of indices. Pre-terminals yield a production with their terminal as a direct argument to the pre-terminal, and an empty RHS. Frontier non-terminals only appear on the RHS of a production. See figure 3 for examples of LCFRS productions extracted from discontinuous trees.

2.2 Tree-Substitution Grammar

In this section we present a reduction of an arbitrary discontinuous TSG to a string-rewriting LCFRS. We first look at general strategies for reducing a TSG to a simpler formalism, and then show that these also apply for the discontinuous case.

A TSG is a tuple $\langle N, T, R, S \rangle$. N and T are disjoint finite sets of non-terminals and terminals, respectively. R is a finite set of elementary trees of depth greater than or equal to 1. Elementary trees from R are combined by substituting a derived tree rooted at a non-terminal X at some leaf node in an elementary tree with a frontier node labeled with X. Derived trees rooted at the start symbol S with leaf nodes labeled by terminal symbols are taken to be the trees generated by the grammar. See figure 2 for an example of a TSG derivation; this derivation contains discontinuous constituents, how these are combined with a TSG shall be explained below.

Goodman (2003) gives a reduction to a PCFG for the special case of a TSG based on all fragments from a given treebank. This reduction is stochastically equivalent after the summation of probabilities from equivalent derivations—however, it does not admit parsing of TSGs with arbitrary sets of elementary trees or arbitrary probability models.

We use a transformation based on the one given in Sangati and Zuidema (2011). Internal nodes are removed from elementary trees, yielding a flattened tree of depth 1. We binarize this flattened tree with a left-factored binarization that adds unique identifiers to every intermediate node introduced by the binarization. In order to separate phrasal and lexical productions, a new POS tag is introduced for each terminal, which selects for that specific terminal. A sequence of productions is then read off from the transformed tree. The unique identifier in the first production is used to look up the original elementary tree in a backtransform table,[1] which is used to restore the internal nodes after parsing. The weight associated with an elementary tree carries over to the first production it produces; the rest of the productions are assigned a weight of 1.

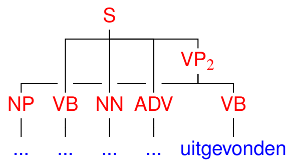
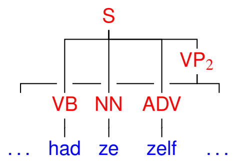
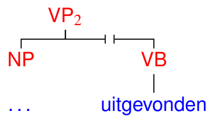
Elementary tree	Productions
	$S(ab) \rightarrow S^1(a)\ VB(b)$ $S^1(ab) \rightarrow S^2(a)\ ADV(b)$ $S^2(ab) \rightarrow S^3(a)\ NN(b)$ $S^3(ab) \rightarrow NP(a)\ VB^4(b)$ $VB^4(\text{uitgevonden}) \rightarrow \varepsilon$
	$S(abc) \rightarrow S^5_2(a,c)\ ADV^6(b)$ $S^5_2(ab,c) \rightarrow S^7_2(a,c)\ NN(b)$ $S^7_2(ab,c) \rightarrow VP_2(a,c)\ VB^8(b)$ $VB^8(\text{had}) \rightarrow \varepsilon$ $NN^7(\text{ze}) \rightarrow \varepsilon$ $ADV^6(\text{zelf}) \rightarrow \varepsilon$
	$VP_2(a,b) \rightarrow NP(a)\ VB^9(b)$ $VB^9(\text{uitgevonden}) \rightarrow \varepsilon$

Figure 3: Transforming a tree-substitution grammar into an LCFRS. The elementary trees are extracted from the tree in figure 1 with abbreviated labels. The right column shows the productions after transforming each elementary tree. Note that the productions for the first elementary tree contain no discontinuity, because the discontinuous internal node is eliminated. Conversely, the transformation may also introduce more discontinuity, due to the binarization.

The transformation defined above assumes that a sequence of productions can be read off from a syntactic tree, such as a standard phrase-structure tree that can be converted into a sequence of context-free grammar productions. Using the method for inducing LCFRS productions from syntactic trees given in the previous section, we can apply the TSG transformation for discontinuous trees as well. Figure 3 illustrates the transformation of a discontinuous TSG.

[1]Note that only this first production requires a globally unique identifier; to reduce the grammar constant, the other identifiers can be merged for equivalent productions.

3 Inducing a TSG from a treebank

In Data-Oriented Parsing the grammar is the treebank itself, and in principle all possible fragments from its trees can be used to derive new sentences. Grammar induction is therefore conceptually simple (although the grammar is very large), as there is no training or learning involved. A fragment of a tree T is defined as a connected subgraph of T with two or more nodes, where each node in the fragment either has no children or the same children as the corresponding node in T.

The use of all possible fragments allows for multiple derivations of the same tree; this spurious ambiguity is seen as a virtue in DOP, because it combines the advantages of specific larger fragments and the smoothing of smaller fragments. This is in contrast to more parsimonious approaches which decompose each tree in the training corpus into a sequence of fragments representing a single derivation, such as in Bayesian TSG (Post and Gildea, 2009; Cohn et al., 2010)

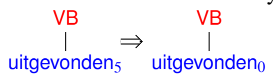
Representing all possible fragments of a treebank is infeasible, since the number of fragments is exponential in the number of nodes. A practical solution is to define a subset. A method called Double-DOP (2DOP; Sangati and Zuidema, 2011) realizes this without compromising on the principle of data-orientation by restricting the set to recurring fragments, i.e., fragments that occur at least twice. These are found by considering every pair of trees and extracting the largest tree fragments they have in common. It is feasible to do this exhaustively for the whole treebank. This is in contrast to the sampling of fragments in earlier DOP models (Bod, 2001) and Bayesian TSGs. Since the space of fragments is enormous (viz. exponential), it stands to reason that a sampling approach will not extract all relevant fragments in a reasonable time frame.

Sangati et al. (2010) presents a tree-kernel method for extracting maximal recurring fragments that operates in time quadratic in the number of nodes in the treebank. However, using a different tree kernel, tree fragments can be obtained from a treebank in linear average time (van Cranenburgh, 2012b).

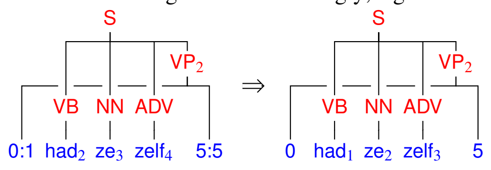
3.1 Discontinuous fragments

The aforementioned fragment extraction algorithms can be adapted to support trees with discontinuous constituents, using a representation where leaf nodes are decorated with indices indicating their ordering. This

1. Translate indices so that they start at 0; e.g.:



2. Reduce spans of frontier non-terminals to length 1; move surrounding indices accordingly; e.g.:



3. Compress gaps to length 1; e.g.:

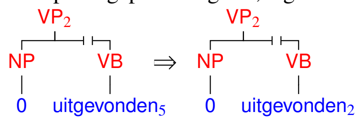


Figure 4: Canonicalization of fragments extracted from parse trees. The example fragments have been extracted from the tree in figure 1. The fragments are visualized here as discontinuous tree structures, but since the discontinuities are encoded in the indices of the yield, they can be represented in a standard bracketing format as used by the fragment extractor.

makes it possible to use the same data structures as for continuous trees, as long as the child nodes are kept in a canonical order (induced from the order of the lowest index of each child). Indices are used not only to keep track of the order of lexical nodes in a fragment, but also for that of the contribution of frontier non-terminals. This is necessary in order to preserve the configuration of the yield in the original sentence. The indices are based on those in the original sentence, but need to be decoupled from this original context. This process is analogous to how LCFRS productions are read off from a tree with discontinuous constituents. The canonicalization of fragments is achieved in three steps, described and illustrated in figure 4. In the examples, frontier non-terminals have spans denoted with inclusive *start:end* intervals, as extracted from the original parse tree, which are reduced to variables denoting a contiguous spans whose relation to the other spans is reflected by their indices.

When binarized with $h = \infty$, $v = 1$ markovization (Klein and Manning, 2003), about 8.5 % of fragments extracted from the Negra treebank (cf. sec. 5.1) contain a discontinuous root or internal node, compared to 30 % of sentences in the treebank that contain one or more discontinuous constituents.

4 Parsing

After extracting fragments, we apply the grammar transformation to turn them into grammar productions. In order to achieve full coverage, we augment the set of fragments with cover fragments of depth 1 corresponding to all single productions in the treebank. Productions corresponding to fragments are assigned a probability based on the frequency of the respective fragment, productions introduced by the transformation are given a probability of 1.

We parse with the transformed grammar using the `disco-dop` parser (van Cranenburgh et al., 2011; van Cranenburgh, 2012a). This is an agenda-based parser for PLCFRS based on the algorithm in Kallmeyer and Maier (2010, 2013), extended to produce *n*-best derivations (Huang and Chiang, 2005) and exploit coarse-to-fine pruning (Charniak et al., 2006).

4.1 Probabilities and disambiguation

To instantiate the probabilistic model we use the relative frequency estimate (RFE), since it has shown good results with the Double-DOP model (Sangati and Zuidema, 2011). The frequency of fragments is obtained by the fragment extractor, divided by the total frequency mass of fragments with the same root node.

In DOP many derivations may produce the same parse tree, and it has been shown that approximating the most probable parse, which considers all derivations for a tree, yields better results than the most probable derivation (Bod, 1995). To select a parse tree from a derivation forest, we marginalize the 10,000-best DOP derivations and select the tree with the most probability mass.

4.2 Coarse-to-fine pruning

In order to tame the complexity of LCFRS and DOP, we apply the same coarse-to-fine pruning as in van Cranenburgh (2012a). Namely, we parse in three stages:

1. Split-PCFG: A PCFG approximation of the discontinuous treebank grammar; rewrites spans of discontinuous constituents independently
2. PLCFRS: The discontinuous treebank grammar; rewrites discontinuous constituents in a single operation
3. The discontinuous DOP grammar: tree fragments instead of individual productions from treebank

The first stage is necessary because without pruning, the PLCFRS generates too many discontinuous spans, the majority of which are implausible or not even part of a complete derivation. The second stage is not necessary for efficiency but gives slightly better accuracy on discontinuous constituents.

The PCFG approximation splits discontinuous constituents into several non-terminals related through their labels; e.g.:

$$\begin{aligned} \text{PLCFRS production:} \quad & \mathrm{VP}_2(a,b) \rightarrow \mathrm{VB}(a)\ \mathrm{PRT}(b) \\ \text{PCFG approximation:} \quad & \{\ \mathrm{VP}_2{*}1 \rightarrow \mathrm{VB}, \\ & \quad \mathrm{VP}_2{*}2 \rightarrow \mathrm{PRT}\ \} \end{aligned}$$

In a post-processing step PCFG derivations are converted to discontinuous trees by merging siblings marked with '*'. This approximation overgenerates compared to the respective LCFRS; e.g., two components $\mathrm{VP}_2{*}1$ and $\mathrm{VP}_2{*}2$ may be generated which where extracted from different discontinuous constituents, such that their combination could not be generated by the LCFRS.

Pruning is achieved by limiting the second and third stages to the labeled spans occurring in the *k*-best derivations of the previous stage. The initial values for *k* are 10,000 and 50, for the PLCFRS and DOP grammar respectively. These values are chosen to be able to directly compare the new approach with the results in van Cranenburgh (2012a). However, experimenting with a higher value for *k* for the DOP stage has shown to yield improved performance.

4.3 Reconstructing derivations

After a derivation forest is obtained and a list of *k*-best derivations has been produced, the backtransform is applied to these derivations to recover their internal structure. This proceeds by doing a depth-first traversal of the derivations, and expanding each non-intermediate[2] node into a template of the original fragment. These templates are stored in a backtransform table indexed by the first binarized production of the fragment in question. The template fragment has its substitution sites marked, which are filled with values obtained by recursively expanding the children of the current constituent. To reconstruct 10,000 derivations takes 2 seconds on average.

[2]An intermediate node is a node introduced by the binarization.

Language	treebank	train	dev	test
GERMAN	Negra	sent. 1–18,602	sent. 19,603–20,602	sent. 18,603–19,602
GERMAN	Tiger	sec. 2–9 / 1–9	sec. 1	sec. 0
ENGLISH	PTB: WSJ	sec. 2-21	sec. 24	sec. 23
DUTCH	Alpino	16,319 sents.	extra: 446 sents.	CoNLL2006: 386 sents.
DUTCH	Lassy small	52,157 sents.	6,520 sents.	6,523 sents.

Table 1: The discontinuous treebanks used in the experiments.

5 Experimental setup

In this section we describe the experimental setup for benchmarking our discontinuous Double-DOP implementation on several discontinuous treebanks.

5.1 Data

We evaluate on three languages: the German Negra & Tiger treebanks (Skut et al., 1997; Brants et al., 2002), a discontinuous version of the Penn treebank (Evang and Kallmeyer, 2011), and the Dutch Alpino & Lassy treebanks (van der Beek et al., 2002; Van Noord, 2009); cf. table 1. Negra contains discontinuous annotations by design, as a strategy to cope with the relatively free word-order of German. The discontinuous Penn treebank consists of the WSJ section in which traces have been converted to discontinuous constituents; we use the version used in Evang and Kallmeyer (2011, sec. 5.1-5.2) without restrictions on the transformations. The Alpino treebank is referred to as a dependency treebank but when discontinuity is allowed it can be directly interpreted as a constituency treebank. Finally, Tiger and Lassy use similar annotation schemes as Negra and Alpino, respectively. The train-dev-test splits we employ are as commonly used for the Penn treebank; for Negra we use the one defined in Dubey and Keller (2003); for Tiger we follow Hall and Nivre (2008) who define sections 0–9 where sentence i belongs to section i mod 10; for Alpino and Lassy the split is our own.[3]

For purposes of training we remove grammatical functions from the treebanks, and binarize the trees in the training sets head-outward with $h = 1, v = 1$ markovization ($v = 2$ for PTB) and heuristics for head assignment (Klein and Manning, 2003); i.e., n-ary nodes are factored into nodes specifying an immediate sibling and parent. We add fan-out markers to guarantee unique fan-outs for non-terminal labels, e.g., $\{\mathrm{VP}, \mathrm{VP}_2, \mathrm{VP}_3, \ldots\}$, which are removed again for evaluation. We apply a few simple manual state splits.[4] In order to compare the results on Negra with previous work, we do not apply the state splits when working with gold POS tags.

The complexity of parsing with an LCFRS depends on the maximal sum of the non-terminal fan-outs of its productions (Gildea, 2010). Using this measure, parsing with the DOP grammars extracted from Negra, WSJ, and Alpino has a worst-case time complexity of $O(n^9)$. The complexities for Tiger and Lassy are $O(n^{10})$ and $O(n^{12})$ respectively, due to a handful of anomalous sentences; by discarding these sentences, a grammar with a complexity of $O(n^9)$ can be obtained with no or negligible effect on accuracy.

5.2 Unknown words

In initial experiments we present the parser with the gold part-of-speech tags, as in previous experiments on discontinuous parsing. Later we show results when tags are assigned automatically with a simple unknown word model, based on the Stanford parser (Klein and Manning, 2003). Tags that rewrite more than σ words are considered open class tags, and words they rewrite are open class words. Open class words in the training

[3]The Alpino training set consists of all manually corrected sentences distributed with the Alpino parser, except for the Lassy corpus samples, `gen_g_suite`, and our development and test set, `extra` and `CoNNL2006` respectively. The Lassy split derives from 80-10-10 partitions of the canonically ordered sentence IDs in each subcorpus (viz. `dpc`, `WR`, `WS`, and `wiki`).

[4]For English we apply the state splits described in Evang and Kallmeyer (2011, sec. 4.2). S nodes with a WH-element are marked as such. VPs with as head a bare infinitive, to-infinitive, or particle verb are marked as such. The marking for VPs headed by a bare or to-infinitive is percolated to the parent S-node.

For Dutch and German we split the POS tags for sentence-ending punctuation '.!?'. For German we additionally split S nodes that are relative clauses, based on the respective grammatical function label.

set that do not occur more than 4 times are replaced with features; words in the test set which are not part of the known words from the training set are replaced with the same features. The features are defined in the Stanford parser as model 4, which is relatively language independent. ϵ probability mass is handed out for combinations of known open class words with unseen tags. For ϵ we use 0.01; σ is tuned on each training set to ensure that no closed class words are identified as open class words; for English and German we use 150, and 100 for Dutch.

5.3 Discontinuity without LCFRS

The idea up to now has been to generate discontinuous constituents using formal rewrite operations of LCFRS. However, the PCFG approximation used in the pruning stage encodes discontinuities as part of the labels. Instead of using this technique only as a crutch for pruning, it can also be combined with the use of fragments to obtain a purely cubic time pipeline. While the PCFG approximation increases the independence assumptions for discontinuous constituents, the use of large fragments can mitigate this increase. We shall refer to this alternative approach as 'Split-2DOP.'

5.4 Metrics

We employ the exact match and the Parseval measures (Black et al., 1992) as evaluation metrics. The latter can be straightforwardly generalized to discontinuous spans by representing spans of bracketings as sets of indices (Maier, 2010). Unfortunately it is not always made explicit in previous work on Negra parsing what kind of evaluation parameters are being used. We use the evaluation parameters typically used with `EVALB` on the Penn treebank. Namely, the root node, as well as punctuation, are not counted towards the score (similar to `COLLINS.prm`, except that we discount all punctuation, including brackets). Counting the root node as a constituent should not be done because it is not part of the corpus annotation and the parser is able to generate it without doing any work; when the root node is counted it inflates the F-score by several percentage points. Punctuation should be ignored because in the original annotation of the Dutch and German treebanks, punctuation is attached directly under the root node instead of as part of constituents. Punctuation can be re-attached using heuristics for the purposes of parsing, but evaluation should not be affected by this.

Model	k=50	k=5000
DOP reduction: disco-DOP	74.3	73.5
Double-DOP: disco-2DOP	76.3	**77.7**

Table 2: Comparing the DOP reduction (implicit fragments) with Double-DOP (explicit fragments) on the Negra development set with different amounts of pruning (higher *k* means less pruning; gold POS tags).

6 Evaluation

Table 2 compares previous results of Disco-DOP to the new Disco-2DOP implementation. The second column shows the accuracy for different values of *k*, i.e., the number of coarse derivations that determine the allowed labeled spans for the fine stage. While increasing this value did not yield improvements using the DOP reduction, with Disco-2DOP there is a substantial improvement in performance, with $k = 5000$ yielding the best score among the handful of values tested.

Table 3 lists the results for discontinuous parsing of three Germanic languages, with unknown word models. The cited work by Kallmeyer and Maier (2013) and Evang and Kallmeyer (2011) also uses LCFRS for discontinuity but employs a treebank grammar with relative frequencies of productions. Hall and Nivre (2008) use a conversion to dependencies from which discontinuous constituents can be recovered. For English and German the results improve upon the best known discontinuous constituency parsing results. The new system achieves a 16 % relative error reduction over the previous best result for discontinuous parsing on sentences $\leqslant 40$ in the Negra test set. In terms of efficiency the Disco-2DOP model is more than three times as fast as the DOP reduction, taking about 3 hours instead of 10 on a single core. The grammar is also more compact: the size of the Disco-2DOP grammar is only a third of the DOP reduction, at 6 MB versus 18 MB compressed size.

The substantial improvements on the larger German and Dutch treebanks Tiger and Lassy suggest that providing more training data will keep improving accuracy. The results for Dutch are not comparable to earlier work because such work has only been evaluated on dependency relations of grammatical functions, which our model does not produce. Earlier work on recovering empty categories and their antecedents in the Penn treebank (Johnson, 2002; Gabbard et al., 2006;

Parser, treebank	$\lvert w \rvert$	DEV			TEST		
		POS	F1	EX	POS	F1	EX
GERMAN							
*vanCra2012, Negra	$\leqslant 40$	100	74.3	34.3	100	72.3	33.2
†*KaMa2013, Negra	$\leqslant 30$				100	75.8	
*this paper, Negra	$\leqslant 40$	100	**77.7**	**41.5**	100	**76.8**	**40.5**
this paper, Negra	$\leqslant 40$	96.7	76.4	39.2	96.3	74.8	38.7
HaNi2008, Tiger	$\leqslant 40$				97.0	75.3	32.6
this paper, Tiger	$\leqslant 40$	97.6	78.7	40.5	97.6	**78.8**	**40.8**
ENGLISH							
†*EvKa2011, WSJ	< 25				100	79.0	
this paper, WSJ	$\leqslant 40$	96.0	**85.2**	28.0	96.6	**85.6**	31.3
DUTCH							
this paper, Alpino	$\leqslant 40$	90.1	74.5	37.2	85.2	65.9	23.1
this paper, Lassy	$\leqslant 40$	94.1	79.0	37.4	94.6	77.0	35.2

Table 3: Discontinuous parsing of three Germanic languages. POS is the part-of-speech tagging accuracy, F1 is the labeled bracketing F1-score, EX is the exact match score. Results marked with * use gold POS tags; those marked with † do not discount the root node and punctuation. NB: KaMa, EvKa, and HaNi use a different test set and length restriction. Key to citations: vanCra: van Cranenburgh (2012a); KaMa: Kallmeyer and Maier (2013); HaNi: Hall and Nivre (2008); EvKa: Evang and Kallmeyer (2011).

Pipeline	F1 %	EX %
Split-PCFG (no LCFRS, no TSG)	65.8	28.0
Split-PCFG ⇒ PLCFRS (no TSG)	65.9	28.6
Split-PCFG ⇒ PLCFRS ⇒ 2DOP	77.7	41.5
Split-PCFG ⇒ Split-2DOP (no LCFRS)	78.1	42.0

Table 4: Parsing discontinuous constituents is possible without LCFRS (Negra dev. set, gold POS tags; results are for final stage).

Schmid, 2006) has recovered long-distance dependencies by producing the traces and co-indexation as in the original annotation; unfortunately the results are not directly comparable because their evaluation method depends on having both traces and antecedents, while our model directly generates discontinuous constituents.

Table 4 shows a comparison of coarse-to-fine pipelines with and without LCFRS, showing that, surprisingly, the use of a formalism that explicitly models discontinuity as an operation does not give any improvement over a simpler model in which discontinuities are only modeled probabilistically by encoding them into labels and fragments. This demonstrates that given the use of tree fragments, discontinuous rewriting through LCFRS comes at a high computational cost without a clear benefit over CFG.

From the results it is clear that a probabilistic tree-substitution grammar is able to provide much better results than a simple treebank grammar. However, it is not obvious whether the improvement is due to the more fine-grained statistics (i.e., weakened independence assumptions), or because of the use of larger chunks. A serendipitous discovery during development of the parser provides insight into this: during an experiment, the frequencies of fragments were accidentally permuted and assigned to different fragments, but the resulting decrease in accuracy was surprisingly low, from 77.7 % to 74.1 % F1—suggesting that most of the improvement over the 65.9 % of the PLCFRS treebank grammar comes from memorizing larger chunks, as opposed to statistical reckoning.

7 Conclusion

We have shown how to parse with discontinuous tree-substitution grammars along with a practical implementation. We have presented a fragment extraction tool that finds recurring structures in treebanks efficiently, and supports discontinuous treebanks. This enables a data-oriented parsing implementation providing a compact, efficient, and accurate model for discontinuous parsing in a single generative model that improves upon previous results for this task.

Surprisingly, it turns out that the formal power of LCFRS to describe discontinuity is not necessary, since equivalent results can be obtained with a probabilistic tree-substitution grammar in which long-distance dependencies are encoded as part of non-terminal labels.

The source code of the parser used in this work has been released as `disco-dop 0.4`, available at: `https://github.com/andreasvc/disco-dop`

Acknowledgments

We are grateful to Kilian Evang for supplying the discontinuous Penn treebank, to Remko Scha for extensive feedback and fruitful discussions, to the reviewers for detailed comments, and to Corina Koolen and Kim Jautze for reading drafts.

This work is part of The Riddle of Literary Quality, a project supported by the Royal Netherlands Academy of Arts and Sciences through the Computational Humanities Program.

References

Mohit Bansal and Dan Klein. 2010. Simple, accurate parsing with an all-fragments grammar. In *Proceedings of ACL*, pages 1098–1107.

Shane Bergsma, Matt Post, and David Yarowsky. 2012. Stylometric analysis of scientific articles. In *Proceedings of NAACL*, pages 327–337. `http://aclweb.org/anthology/N12-1033`.

Ezra Black, John Lafferty, and Salim Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proceedings of ACL*, pages 185–192. `http://aclweb.org/anthology/P92-1024`.

Rens Bod. 1995. The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of EACL*, pages 104–111. `http://aclweb.org/anthology/E95-1015`.

Rens Bod. 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings of ACL*, pages 69–76.

Rens Bod, Remko Scha, and Khalil Sima'an, editors. 2003. *Data-Oriented Parsing*. The University of Chicago Press.

Pierre Boullier. 1998. Proposal for a natural language processing syntactic backbone. Technical Report RR-3342, INRIA-Rocquencourt, Le Chesnay, France. `http://www.inria.fr/RRRT/RR-3342.html`.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The Tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, pages 24–41.

Eugene Charniak. 1996. Tree-bank grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1031–1036.

Eugene Charniak, Mark Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, et al. 2006. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of NAACL-HLT*, pages 168–175.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *The Journal of Machine Learning Research*, 11(Nov):3053–3096.

Amit Dubey and Frank Keller. 2003. Parsing german with sister-head dependencies. In *Proceedings of ACL*, pages 96–103.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of IWPT*, pages 104–116. `http://aclweb.org/anthology/W11-2913`.

Ryan Gabbard, Seth Kulick, and Mitchell Marcus. 2006. Fully parsing the Penn treebank. In *Proceedings of NAACL-HLT*, pages 184–191.

Daniel Gildea. 2010. Optimal parsing strategies for linear context-free rewriting systems. In *Proceedings of NAACL HLT 2010.*, pages 769–776.

Joshua Goodman. 2003. Efficient parsing of DOP with PCFG-reductions. In Bod et al. (2003).

Spence Green, Marie-Catherine de Marneffe, John Bauer, and Christopher D. Manning. 2011. Multiword expression identification with tree substitution grammars: A parsing tour de force with French. In *Proceedings of EMNLP*, pages 725–735.

Johan Hall and Joakim Nivre. 2008. Parsing discontinuous phrase structure with grammatical functions. In Bengt Nordstrm and Aarne Ranta, editors, *Advances in Natural Language Processing*, volume 5221 of *Lecture Notes in Computer Science*, pages 169–180. Springer Berlin / Heidelberg. `http://dx.doi.org/10.1007/978-3-540-85287-2_17`.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of IWPT 2005*, pages 53–64.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of ACL*, pages 136–143.

Laura Kallmeyer and Wolfgang Maier. 2010. Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 537–545.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, volume 1, pages 423–430.

Wolfgang Maier. 2010. Direct parsing of discontinuous constituents in German. In *Proceedings of the SPMRL workshop at NAACL HLT 2010*, pages 58–66.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of Formal Grammar 2008*, pages 61–76.

Matt Post. 2011. Judging grammaticality with tree substitution grammar derivations. In *Proceedings of the ACL-HLT 2011*, pages 217–222.

Matt Post and Daniel Gildea. 2009. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48.

Federico Sangati and Willem Zuidema. 2011. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of EMNLP*, pages 84–95. `http://aclweb.org/anthology/D11-1008`.

Federico Sangati, Willem Zuidema, and Rens Bod. 2010. Efficiently extract recurring tree fragments from large treebanks. In *Proceedings of LREC*, pages 219–226. `http://dare.uva.nl/record/371504`.

Remko Scha. 1990. Language theory and language technology; competence and performance. In Q.A.M. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere, the Netherlands. Original title: Taaltheorie en taaltechnologie; competence en performance. Translation available at `http://iaaa.nl/rs/LeerdamE.html`.

Helmut Schmid. 2006. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proceedings of COLING-ACL*, pages 177–184.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of ACL*, pages 440–448. `http://aclweb.org/anthology/P12-1046`.

Wojciech Skut, Brigitte Krenn, Thorten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of ANLP*, pages 88–95.

Benjamin Swanson and Eugene Charniak. 2012. Native language detection with tree substitution grammars. In *Proceedings of ACL*, pages 193–197. `http://aclweb.org/anthology/P12-2038`.

Andreas van Cranenburgh. 2012a. Efficient parsing with linear context-free rewriting systems. In *Proceedings of EACL*, pages 460–470. `http://aclweb.org/anthology/E12-1047`.

Andreas van Cranenburgh. 2012b. Extracting tree fragments in linear average time. Technical Report PP-2012-18, FNWI/FGw: Institute for Logic, Language and Computation (ILLC). `http://dare.uva.nl/en/record/421534`.

Andreas van Cranenburgh. 2012c. Literary authorship attribution with phrase-structure fragments. In *Proceedings of the NAACL-HLT 2012 Workshop on Computational Linguistics for Literature*, pages 59–63. `http://aclweb.org/anthology/W12-2508`.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. 2011. Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of SPMRL*, pages 34–44. `http://aclweb.org/anthology/W11-3805`.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino dependency treebank. *Language and Computers*, 45(1):8–22.

Gertjan Van Noord. 2009. Huge parsed corpora in Lassy. In *Proceedings of TLT7*. LOT, Groningen, The Netherlands.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of ACL*, pages 104–111. `http://www.aclweb.org/anthology/P87-1015`.

An Efficient Typed Feature Structure Index: Theory and Implementation

Bernd Kiefer and Hans-Ulrich Krieger
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{kiefer,krieger}@dfki.de

Abstract

Many applications (not necessarily only from computational linguistics), involving record- or graph-like structures, would benefit from a framework which would allow to efficiently test a single structure ϕ under various operations $\odot$ against a compact representation $\Im$ of a set of similar structures: $\phi \odot \Im$. Besides a Boolean answer, we would also like to see those structures stored in $\Im$ which are entailed by operation $\odot$. In our case, we are especially interested in $\odot$s that implement feature structure subsumption and unifiability. The urgent need for such a kind of framework is related to our work on the approximation of (P)CFGs from unification-based grammars. We not only define the mathematical apparatus for this in terms of finite-state automata, but also come up with an efficient implementation mostly along the theoretical basis, together with measurements in which we compare our implementation of $\Im$ against a discrimination tree index.

1 Introduction and Motivation

There exist several applications in which a single feature structure (FS) is tested against a large set of structures with operations like *forward and backward subsumption*, *unifiability*, or *containedness*.

One possibility to optimize this task is to separate the set into subsets with disjoint properties that help to quickly cut down the number of structures under consideration. This is usually called *indexing* of feature structures and several proposals have been published on how to achieve this efficiently, namely, (Goetz et al., 2001), (Kiefer and Krieger, 2002), (Ninomiya and Makino, 2002), and (Munteanu, 2003). All approaches use more or less the same strategy: *select a set of paths from the FSs, extract the values/types at the end of these paths, and then build indexing information based on these types*. The approaches differ mostly in the last step, viz., on how the information of the set of types is encoded or exploited, resp.

In cases where the structures originate from applications like constraint-based natural language processing, they are often structurally similar to one another. In graph-based implementations, all the above mentioned operations require a traversal of the graph structure and the execution of appropriate tests at the edges or nodes. It seems natural to exploit this similarity by avoiding multiple traversals, packing the information of the elements such that the tests can be performed for a whole set of structures at once.

With this idea in mind, we propose a new method to store a large set of FSs in a very compact form, which is more memory efficient than storing the set of single structures. The resulting data structure $\Im$ reminds us of *distributed disjunctions* (Maxwell III and Kaplan, 1991). Mathematically, we will characterize $\Im$ and the required operations in terms of finite automata and operations over these automata, will provide implementation details of the packed index data structure, and will compare $\Im$ with the index used in (Kiefer and Krieger, 2004).

The motivation to study this task arose from experiments to approximate a large-coverage constraint grammar, the LinGO ERG (Flickinger, 2011), in terms of CFGs. Besides this, feature structure indexing can and has also been used for

- feature structure operations for extracting context-free grammars (**this paper**);
- lexicon and chart lookup during parsing or generation, or for accessing FS tree banks;
- establishing a lookup structure for a semantic RDF triple repository.

The structure of this paper is as follows. In section 2, the application that motivates this work is described. After that, a mathematical characterization of the data structure and algorithms in terms of finite state automata and operations thereon is presented in section 3. The implementation and its relation to the mathematical apparatus is covered in section 4. Section 5 contains measurements which compare the implementation of the new index to the discrimination tree index from (Kiefer and Krieger, 2004).

2 Our Application: CF Approximations

The need for this data structure arose from an attempt to compute a context-free approximation of the current LinGO ERG HPSG grammar (Flickinger, 2011). This grammar is one of the most complex constraint-based grammars ever written, containing 91934 types, 217 rules, and 36112 lexical types, resulting in a very high coverage of the English language.
The first tests were run using a reimplementation of (Kiefer and Krieger, 2004) in a 32-bit Java environment, all ending in memory overflow. After a reduction of the initial (lexicon) structures, the next test was manually cancelled after two weeks which made the need for a more efficient way of handling the vast amount of FSs and operations obvious. The algorithm outlined below is described in more detail in (Kiefer and Krieger, 2004), including various optimizations.

The Approximation Algorithm. Approximation starts by applying the available rules of a grammar to the already existing lexical entries, thereby generating new FSs which, again, are used as rule arguments in new rule instantiations. This process is iterated, until no further structures are computed, i.e., until a fixpoint is reached. For this process to terminate, it must be ensured that the FSs will not continue to grow forever. This is achieved by two means.

Applying restrictors. Restrictors are functions that delete or transform parts of a feature structure, and are applied to get rid of features that encode the constituent tree, or parts that will only increase the number of structures without imposing constraints during parsing, e.g., terminal string labels in lexicon entries. More than one restrictor may be used, e.g., to take proper care of lexical vs. phrasal structures.

Reducing the set of FSs under subsumption. The set of FSs T may not contain elements which are comparable under subsumption, i.e., there are no FS $\phi, \phi' \in T$ s.t. $\phi \sqsubseteq \phi'$. This implies that a newly generated FS ψ must be checked against T, and in case ψ is subsumed by some element from T, it is discarded; conversely, all elements that are subsumed by ψ must be found and removed from T.

The informal algorithm from above makes use of an index structure $\Im$ to efficiently implement T, using the following operations thereon:

1. find potential unifiable members for an input feature structure during rule instantiations;
2. check if a new FS is subsumed by a structure in the index after a new structure has been created;
3. add a non-subsumed structure, and remove all current members which are subsumed by the new one.

It is worth noting that operation 1 is only an imperfect filter in all implementations, i.e., full unification still has to be performed on all structures that are returned, and there will still be unification failures for some structures. Contrary to other approaches, the subsumption operations in our implementation return all and only all correct answers.

3 A FSA Characterization of the Index

We start this section with some recap in order to motivate our decisions why we have deviated from some of the standard definitions. This includes a special definition of typed feature structures (TFS) as deterministic finite state automata (deterministic FSA or DFSA) which replaces the type decoration of nodes by additional type labels, attached to new edges.
Given the DFSA characterization of typed feature structures, we define TFS unification and subsumption in terms of operations on the corresponding automata and their recognized languages.
We then present the finite-state characterization of the index, focussing on the integration of new TFSs.
After that, we describe typed feature structure subsumption of an input query (a potentially underspecified TFS) against the index. This includes the definition of an effective procedure that constructs answer automata with their enclosing result structures.

3.1 Edge-Typed Feature Structures

In line with early work by Shieber on the PATR-II system (Shieber, 1984) and work by Kasper & Rounds (Kasper and Rounds, 1986; Rounds and Kasper, 1986) for the untyped case, Carpenter defines a *typed feature structure* (without disjunctions or sets) as a kind of *deterministic finite state automaton* (Hopcroft and Ullman, 1979) with the following signature (Carpenter, 1992, p. 36):

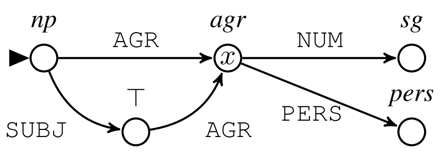
Definition 1 (TYPED FEATURE STRUCTURE)
A (conjunctive) typed feature structure (TFS) over a

finite set of types $\mathcal{T}$ and a *finite* set of features $\mathcal{F}$ is a quadruple $\langle Q, q_0, \theta, \delta \rangle$, such that

- Q is a *finite* set of nodes,
- $q_0 \in Q$ is *the* root node,
- $\theta : Q \to \mathcal{T}$ is a *total* node typing function, and
- $\delta : Q \times \mathcal{F} \to Q$ a *partial* feature value function.

The use of θ and δ thus allows us to attach labels (types and features) to nodes and edges of an automaton, representing a TFS. We note here that this definition does *not* employ a distinguished set of *final* states F. When extending the TFS model in Definition 3, we will assume that F always refers to the set of all *leaf nodes*, nodes that do not have outgoing edges. This will turn out important when we characterize TFS unification and subsumption as operations over the languages recognized by the FSA. The following TFS represents an underspecified singular NP.

Example 1 (FEATURE STRUCTURE)



We often use an alternative representation to depict TFSs, so-called *attribute-value matrices* (AVMs). *Reentrancies* in the above automaton are expressed through logical variables in the AVM (here: x).

Example 2 (ATTRIBUTE-VALUE MATRIX)

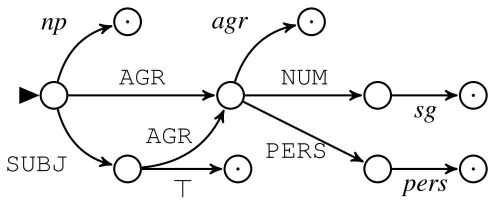
$$\begin{bmatrix} \textit{np} & \\ \texttt{AGR} \; \boxed{x} & \begin{bmatrix} \textit{agr} \\ \texttt{NUM} \; \textit{sg} \\ \texttt{PERS} \; \textit{pers} \end{bmatrix} \\ \texttt{SUBJ|AGR} \; \boxed{x} & \end{bmatrix}$$

In the following, we no longer distinguish between TFSs and AVMs as they denote the same set of objects (there exists a trivial bijection between TFSs and AVMs). We also make use of the notion of a *type hierarchy*, underlying a typed feature structure and operations thereon, such as typed feature structure *unification* and *subsumption*.

Definition 2 (TYPE HIERARCHY)
Let $\mathcal{T}$ be a finite set of types and let $\leq$ be a binary relation over $\mathcal{T}$, called *type subsumption*. A decidable partial order $\langle \mathcal{T}, \leq \rangle$ then is called a type hierarchy (or an inheritance hierarchy). $\mathcal{T}$ contains two special symbols: $\top$ is called the *top type* (or the most general type) and $\bot$ is called the *bottom type* (or the most specific type), according to $\leq$.

We will now deviate from the TFS definition in (Carpenter, 1992) in that we move away from the *node-oriented* type representation to an *edge-based* implementation whose set of features is now given by $\mathcal{F} \uplus \mathcal{T}$. Thus, we have to make sure that $\mathcal{T}$ and $\mathcal{F}$ are disjoint (as can be easily realized via renaming). This extension results in *deterministic* automata with *directly interpretable* edge labels and *unlabeled* nodes. The following TFS depicts this modification when applied to the TFS from Example 1. Note that we have not renamed the types here, but have used a different style (*italics*) in order to distinguish them from the original features (`typewriter`).

Example 3 (FEATURE STRUCTURE, EXTENDED)



Given the above edge-based representation, we can now define an *edge-typed feature structure* (ETFS), mirroring exactly this modification.

Definition 3 (EDGE-TYPED FS)
An edge-typed feature structure (ETFS) over a *finite* set of types $\mathcal{T}$ and a *finite* set of features $\mathcal{F}$ is a quintuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, such that

- Q is a *finite* set of nodes,
- $\Sigma = \mathcal{F} \uplus \mathcal{T}$ is a *finite* input alphabet,
- $\delta : Q \times \Sigma \to Q$ is a *partial* feature value function,
- $q_0 \in Q$ is *the* root node, and
- $F = \{q \in Q \mid \delta(q, a) \uparrow,$ for all $a \in \Sigma\}$ is the *finite* set of final states.

We notice here that F corresponds exactly to the set of *leaf* nodes, nodes *without* outgoing edges, and that the total typing function θ is no longer needed.

We finally define a stronger class of ETFSs by enforcing *acyclicity*, meaning that these structures are only able to recognize *finite* languages. This assumption makes the definition of the index together with index unification and subsumption easy to understand. It is also worth noting that the applications in which we are interested (parsing with HPSG grammars (Pollard and Sag, 1994) and grammar approximation (Kiefer and Krieger, 2004)) do forbid such cyclic TFS. In case cyclic structure result from a unification of two TFSs, the Tomabechi-style unification engines that we were using in our systems will signal a failure in the final copy phase.

Definition 4 (ACYCLIC ETFS)
An acyclic edge-typed feature structure is an ETFS which does not allow for infinite paths:

$$\nexists q \in Q, \nexists f \in \mathcal{F}^* \,.\, \hat{\delta}(q, f) = q$$

Recall, $\Sigma = \mathcal{F} \uplus \mathcal{T}$, so when we say *path* here, we usually refer to elements from $\mathcal{F}^*$ (but *not* from $\mathcal{F}^*\mathcal{T}$, for which we use the term *extended path*). Note that $\hat{\delta}$ above refers to the usual extension of δ, when moving from Σ to Σ^* (Hopcroft and Ullman, 1979, p. 17):

- $\hat{\delta}(q, \epsilon) := q$
- $\hat{\delta}(q, wa) := \delta(\hat{\delta}(q, w), a)$, for $q \in Q, w \in \Sigma^*$, and $a \in \Sigma$

3.2 TFS Unification and Subsumption

Given Definition 3, we are now able to define *ETFS subsumption* $\sqsubseteq$ and *ETFS unification* $\sqcap$ of two ETFSs ϕ and ψ in terms of the languages, recognized by ϕ and ψ. Since the types, represented by the typed edges, need to be interpreted against an inheritance hierarchy $\langle \mathcal{T}, \leq \rangle$, we first define the *recognized pre-language* of an ETFS. Due to space requirements, we restrict ourselves in the paper (but not in the oral presentation) to *coreference-free ETFSs*, as a proper handling of coreferences would require a further modification of the ETFS definition (edges in the DFSA need to be labelled with sets of elements from $\mathcal{F}^*$, expressing equivalence classes).

Definition 5 (ETFS PRE-LANGUAGE)
The pre-language $\mathcal{L}^-(\phi)$ recognized by an ETFS ϕ is defined as follows:

$$\mathcal{L}^-(\phi) := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

Now, the type hierarchy $\langle \mathcal{T}, \leq \rangle$ comes into play when defining the *recognized language* of an ETFS ϕ. Essentially, we "expand" type t at the end of each word from $\mathcal{L}^-$ by $\{s \mid s \leq t\}$.

Definition 6 (ETFS LANGUAGE)
The language $\mathcal{L}(\phi)$ recognized by an ETFS ϕ is defined as follows:

$$\mathcal{L}(\phi) := \{vs \in \Sigma^* \mid vt \in \mathcal{L}^-(\phi) \textit{ and } s \leq t\}$$

Note that the languages we are dealing with are not only regular, but even *finite*, due to the acyclicity assumption, imposed on ETFSs (see Definition 4).

In order to define ETFS subsumption and unification, we need a further definition that introduces an operator Π that basically *chops off* the types at the end of extended paths.

Definition 7 (STRIPPED-DOWN FS)
A stripped-down FS can be obtain from an ETFS ϕ by applying Π to $\mathcal{L}(\phi)$, where

$$\Pi(\mathcal{L}(\phi)) := \{w \in \mathcal{F}^* \mid ws \in \mathcal{L}(\phi) \textit{ and } s \in \mathcal{T}\}$$

We are now almost ready to define the usual ETFS operations. Before doing so, need to talk about the *unique paths* (depicted by $\mathcal{L}_=$) and *shared paths* (depicted by $\mathcal{L}_{\neq}$), relative to ϕ and ψ.

Definition 8 (UNIQUE AND SHARED PATHS)
Given two ETFSs ϕ and ψ, we deconstruct their corresponding languages as follows:

$$\mathcal{L}(\phi) := \mathcal{L}_=(\phi) \uplus \mathcal{L}_{\neq}(\phi)$$
$$\mathcal{L}(\psi) := \mathcal{L}_=(\psi) \uplus \mathcal{L}_{\neq}(\psi)$$

such that

$$\Pi(\mathcal{L}_=(\phi)) = \Pi(\mathcal{L}_=(\psi))$$

Thus

$$\mathcal{L}_{\neq}(\phi) = \mathcal{L}(\phi) \setminus \mathcal{L}_=(\phi)$$
$$\mathcal{L}_{\neq}(\psi) = \mathcal{L}(\psi) \setminus \mathcal{L}_=(\psi)$$

Definition 9 (ETFS SUBSUMPTION/UNIFICATION)

$$\phi \sqsubseteq \psi :\Longleftrightarrow \mathcal{L}_=(\phi) \subseteq \mathcal{L}_=(\psi) \textit{ and } \mathcal{L}_{\neq}(\psi) = \emptyset$$

$$\phi \sqcap \psi = \pi :\Longleftrightarrow \mathcal{L}(\pi) = \mathcal{L}_{\neq}(\phi) \cup \mathcal{L}_{\neq}(\psi) \cup \{wu \mid ws \in \mathcal{L}_=(\phi), wt \in \mathcal{L}_=(\phi), \textit{ and } u = s \wedge t\}$$

It is worth noting that the unification operation directly above assumes that the underlying type hierarchy is GLB-completed as it assumes a unique (and not many) u, resulting from taking the GLB $s \wedge t$. We further note that if $s \wedge t$ fails at any point (i.e., returns $\bot$), π is supposed to be inconsistent.

3.3 A TFS Index

In this subsection, we will establish a typed feature structure index $\Im$ that arises from a collection of ETFSs $\{\phi_1, \ldots, \phi_n\}$ by taking the *union* of the corresponding automata, an operation that can be *effectively* constructed, resulting in, what we call later, an *answer automaton*:

$$\Im = \bigsqcup_{i=1}^{n} \phi_i$$

With *effectively*, we mean here and in the following that there exists an algorithm that directly constructs the resulting DFSA (the *answer* FSA) from the input DFSAs. When an ETFS ϕ has already been inserted in $\Im$, we write

$$\phi \in \Im$$

In order to address all this properly, we need to slightly extend the signature of an ETFS by a total indexing function ι defined on F, enumerating those ETFSs $\phi \in \iota(q)$ at a final node $q \in F$, such that $\hat{\delta}(q_0, w) = q$ *and* $w \in \mathcal{L}^-(\phi)$. Thus, ι records TFSs at q that share an *extended path* $w = ft$ from root node q_0 to q ($f \in \mathcal{F}^*, t \in \mathcal{T}$).

As we we will see in a moment, ι will be utilized to return *all* and *only all* TFSs recorded in the index $\Im$ that are more specific (or equal) than or which are unifiable with a given *query* TFS. Due to space requirements, we restrict ourself here to coreference-free TFSs as this simplifies the formal description of the index.

Definition 10 (ETFS, REVISED)
An edge-typed feature structure (ETFS) is a sextuple $\langle Q, \Sigma, \delta, q_0, F, \iota\rangle$, where Q, Σ, δ, q_0 and F is given in Definition 3 and ι defined as follows:

- $\iota : F \to 2^I$ is a *total* indexing function.

As we see from this definition, ι does not access the feature structures directly. Instead, it utilizes a set of IDs I that identify the corresponding ETFSs Φ from the index through the use of an additional *bijective* function, viz., $id : I \to \Phi$ and $id^{-1} : \Phi \to I$.

This strategy gives an implementation the freedom to relocate the TFSs to a separate table in RAM or even to store them in an external file system. Since it is possible to *reconstruct* feature structures from the index $\Im$, given a specific ID from I, we can even use a memory-bounded internal cache to limit the memory footprint of $\Im$, which is the method of choice in the implementation described in section 2.

When entering a TFS ϕ to the index, a brand-new identifier from I is obtained and added to every final node of ϕ. This assignment is important when we define subsumption and unification over the index in Section 3.4. In the end, the index $\Im$ is still a ETFS/DFSA, but its ι-function differs from a single TFS in that it does *not* return a singleton set, but usually a set with more than one ID.

Before moving on, let us take an example that explains the ideas, underlying the index. The example will display the index after three ETFSs have been added. To ease the pictures here, we will not use the DFSA model of ETFSs, but instead employ the corresponding equivalent AVM description from Example 2. As already explained, types occur as "ordinary" features, and the AVM for the index usually comes with more than one outgoing type features at every node in the DFSA. The set of IDs that refer to those TFSs "entailed" by a current extended path at a final node in the index are attached (via ':') to the type features in the AVMs below.

Example 4 (INDEX WITH THREE ETFSS)
We assume a type hierarchy with the following six types ("lower" depicted types are more specific) and will add the following three ETFSs to an empty index

$$\begin{tikzcd}[column sep=small, row sep=small] & \top \arrow[dl, dash] \arrow[dr, dash] & & \\ s \arrow[d, dash] & & bool \arrow[dl, dash] \arrow[dr, dash] & \\ t & + & & - \end{tikzcd}
\qquad
\begin{bmatrix} s : \{1\} \\ \text{A} \ \begin{bmatrix} bool : \{1\} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} t : \{2\} & \\ \text{A} & \begin{bmatrix} + : \{2\} \end{bmatrix} \\ \text{B} & \begin{bmatrix} + : \{2\} \end{bmatrix} \end{bmatrix}
\qquad
\begin{bmatrix} t : \{3\} & \\ \text{A} & \begin{bmatrix} + : \{3\} \end{bmatrix} \\ \text{B} & \begin{bmatrix} - : \{3\} \end{bmatrix} \end{bmatrix}$$

The resulting index $\Im$, integrating 1, 2, and 3, is:

$$\Im \equiv \begin{bmatrix} s : \{1\} & \\ t : \{2, 3\} & \\ \text{A} & \begin{bmatrix} bool : \{1\} \\ + : \{2, 3\} \end{bmatrix} \\ \text{B} & \begin{bmatrix} + : \{2\} \\ - : \{3\} \end{bmatrix} \end{bmatrix}$$

What this example shows is that the index does in fact result from taking the *union* of the three DFSAs/ETFSs. We also see that the associated ID sets at the final nodes of the index are extended by the IDs of the ETFSs that are going to be inserted.

In order to obtain the *complete* set of ETFSs Φ stored in the index, we need to walk over the set of final nodes and take the union of *all* ID sets (actually, the TFSs associated with the indices):

$$\Phi = \bigcup_{q \in F} \bigcup_{i \in \iota(q)} \{id(i)\}$$

3.4 Querying the Index

We will now define two further natural operations w.r.t. index $\Im$ and a query ETFS ψ, and will present a construction procedure for one of them, viz., $\Im \sqsubseteq \psi$:

1. *return all ϕ_i which are equal or more specific than query ψ:*
   $\Im \sqsubseteq \psi :\Longleftrightarrow \cup_i\, \phi_i$ *s.t.* $\phi_i \sqsubseteq \psi$ *and* $\phi_i \in \Im$

2. *return all ϕ_i which are unifiable with query ψ:*
   $\psi \sqcap \Im :\Longleftrightarrow \cup_i \, \phi_i$ *s.t.* $\phi_i \sqcap \psi \neq \bot$ *and* $\phi_i \in \Im$

(1.) essentially reduces to the construction of a subautomaton, whereas (2.) results in the construction of an intersecting FSA, again two operations that can be *effectively* constructed. By this, as before, we mean that we can directly construct a new ETFS/DFSA from the *signatures* (see Definition 10) of $\Im$ and ψ. Note that we let the indexing function ι^ψ for query ψ always map to the empty set, as it is a query, and *not* a TFS that is part eof the index, i.e., $\iota^\psi : F^\psi \to \emptyset$.
Since the result of these two operations are again DFSAs, we call them *answer automata*, as the subsumed or unifiable structures can be obtained through the use of ι and *id*—the resulting FSA, *as such*, is *not* the answer.

Before presenting the construction procedure for $\Im \sqsubseteq \psi$, we need to report on two important observations. Firstly, the following equality relations always hold for the resulting structure $\cup_i \, \phi_i$ w.r.t query ψ:

$$\forall i \, . \, \Pi(\mathcal{L}^-_=(\psi)) = \Pi(\mathcal{L}^-_=(\phi_i))$$

In other words, the subsumed ETFSs ϕ_i from $\Im$ must at least contain the *same* paths from $\mathcal{F}^*$ than query ψ, and perhaps come up with more specific type labels (and, of course, additional *own* unique paths).

Secondly, if $\phi \in \Im$ *and* $w, w' \in \mathcal{L}^-(\phi)$ s.t. $w = ft$ and $w' = ff't'$ ($f \in \mathcal{F}^*; f' \in \mathcal{F}^+; t, t' \in \mathcal{T}$), then $id^{-1}(\phi) \in \iota(q)$ *and* $id^{-1}(\phi) \in \iota(q')$, given $\hat{\delta}(q_0, w) = q$ and $\hat{\delta}(q_0, w') = q'$ (q_0 being the root node of $\Im$). I.e., a recorded ETFS ϕ in $\Im$ with index $id^{-1}(\phi)$ under path f will also be recorded with the same index under the *longer* path ff'.

Given these two remarks, it thus suffices to construct an answer automaton $\Im \sqsubseteq \psi$ that is *structural equivalent* to ψ and whose ι-function is no longer empty, but instead constructed from ι of $\Im$ w.r.t. a type hierarchy.

Algorithm 1 (INDEX SUBSUMPTION)
Given an index $\Im = \langle Q^\Im, \Sigma^\Im, \delta^\Im, q_0^\Im, F^\Im, \iota^\Im \rangle$, a query $\psi = \langle Q^\psi, \Sigma^\psi, \delta^\psi, q_0^\psi, F^\psi, \iota^\psi \rangle$, and a type hierarchy $\langle \mathcal{T}, \leq \rangle$, we define the *subsumed answer automaton*

$$\Im \sqsubseteq \psi := \langle Q, \Sigma, \delta, q_0, F, \iota \rangle$$

where $Q := Q^\psi, \Sigma := \Sigma^\psi, \delta := \delta^\psi, q_0 := q_0^\psi$, and $F := F^\psi$. Now let $\hat{\delta}(q_0, ft) = q \in F$, where $f \in \mathcal{F}^*$, $t \in \mathcal{T}$, and $ft \in \mathcal{L}^-(\psi)$. ι then is given by the following definition ($q \in F$):

$$\iota(q) := \bigcup_{s \leq t} \begin{cases} \emptyset, & \textit{if } \hat{\delta}^\Im(q_0^\Im, fs)\uparrow \\ \iota^\Im(\hat{\delta}^\Im(q_0^\Im, fs)), & \textit{otherwise} \end{cases}$$

The set $\Phi = \cup_i \, \phi_i$ of subsumed ETFSs ϕ_i w.r.t. ψ is finally given by

$$\Phi = \bigcap_{q \in F} \bigcup_{i \in \iota(q)} \{id(i)\}$$

It is worth noting that the transition function for fs is not necessarily defined in $\Im$, since query ψ might employ a feature from $\mathcal{F}$ and/or use a type labeling from $\mathcal{T}$ that is not literally present in $\Im$.

Let us now present a further example, showing how $\Im \sqsubseteq \psi$ looks for two queries w.r.t the index depicted in Example 4.

Example 5 (ANSWER FA FOR TWO QUERIES)
Given the index from Example 4 and queries

$$\psi_1 \equiv \begin{bmatrix} s & \\ \texttt{A} & + \\ \texttt{B} & \textit{bool} \end{bmatrix} \text{ and } \psi_2 \equiv \begin{bmatrix} s & \\ \texttt{A} & \textit{bool} \\ \texttt{C} & \textit{bool} \end{bmatrix}$$

the answer automata have the following structure:

$$\Im \sqsubseteq \psi_1 \equiv \begin{bmatrix} s : \{1\} \cup \{2,3\} = \underline{\{1,2,3\}} & \\ \texttt{A} & \begin{bmatrix} + : \underline{\{2,3\}} \end{bmatrix} \\ \texttt{B} & \begin{bmatrix} \textit{bool} : \emptyset \cup \{2\} \cup \{3\} = \underline{\{2,3\}} \end{bmatrix} \end{bmatrix}$$

$$\Im \sqsubseteq \psi_2 \equiv \begin{bmatrix} s : \{1\} \cup \{2,3\} = \underline{\{1,2,3\}} & \\ \texttt{A} & \begin{bmatrix} \textit{bool} : \{1\} \cup \{2,3\} \cup \emptyset = \underline{\{1,2,3\}} \end{bmatrix} \\ \texttt{C} & \begin{bmatrix} : \emptyset \cup \emptyset \cup \emptyset = \underline{\emptyset} \end{bmatrix} \end{bmatrix}$$

The indices *Id* for the ETFSs from $\Im$ hidden in these answer automata are given by the intersection of the ID sets associated with the final nodes (see Algor. 1):

$$\textit{Id}(\Im \sqsubseteq \psi_1) = \{1,2,3\} \cap \{2,3\} \cap \{2,3\} = \underline{\{2,3\}}$$

$$\textit{Id}(\Im \sqsubseteq \psi_2) = \{1,2,3\} \cap \{1,2,3\} \cap \emptyset = \underline{\emptyset}$$

I.e., ETFS 2 and 3 from Example 4 are subsumed by ψ_1, whereas *no* ETFS can be found in $\Im$ for ψ_2.

Due to space limitations, we are not allowed here to describe the answer automata for index construction $\phi \sqcup \Im$, for the inverse case of index subsumption $\psi \sqsubseteq \Im$, and for index unifiability $\psi \sqcap \Im$. This will be addressed in the oral presentation, where we will also indicate how coreferences in the theoretical description of the index are represented.

4 Implementing the Index

The proposed data structure exploits the similarity of the structures by putting all information into the tree structure of $\Im$, which contains all paths occurring in the stored TFSs. Now, ι is *implemented* by attaching *bit vectors* to the nodes, which is very compact since the member indices are themselves small integer numbers. In contrast to the mathematical description, they encode not only the presence of a type under a specific path, but also the presence or absence of other values, namely features and coreferences.

Our implementation can straightforwardly be used in a parallel execution environment, which allows to scale up the target application more easily with modern machines, as can be seen in the next section. Subsumption and generalization in our implementation always return correct results, while other indexing techniques require the full test to be applied on the results, thus thread-safe versions of the FS operations have to be provided by them. Since almost all unification engines draw their efficiency from invalidating intermediate results by increasing a global counter, concurrent evaluation is only possible at higher costs.[1] Our implementation provides multiple-read/single-write, beneficial for the application described in Section 2, as the amount of queries by far surmounts that of adding or removing structures.

To perform the operations, the index structure is traversed in a canonical order which in turn implies an order over all occurring paths and establishes a one-to-one correspondence between index nodes and nodes in the query feature structure. Because of this, when we subsequently talk about the *current node*, we always mean the pair of corresponding index and query structure node.

Boolean bit vector operations are employed to exclude invalid structures. As a starting point for traversal, a bit vector a is used, where all the bits, representing the current index feature structures, are set. When checking unifiability, for example, we collect at every current node all those defined types t which are incompatible with type s in the query structure ψ at the same current node q. We then use the stored bit vectors b_t^q that encode the index feature structures bearing t to remove them from the set of valid candidates a by executing $a \wedge (\bigwedge_t \neg b_t^q)$.

We describe the implementation of *Id*$(\Im \sqsubseteq \psi)$ and *Id*$(\psi \sqsubseteq \Im)$ in detail below, that is, determining TFSs from the index that are subsumed by (resp. subsuming) query structure ψ. The currently implemented unifiability method only deals with type constraints, and ignores coreferences.

Every member from $\Im$, whose type t is *not subsumed (not subsuming, resp.)* by type s in the current node of the query ψ, is removed. For the generalization case, all contexts with outgoing features missing in the query are removed. After that, the coreference constraints are evaluated. Coreference constraints (sets of paths leading to the same node in an FS) establish equivalence classes of index nodes (exploiting the one-to-one correspondence between query FS nodes and index nodes). We write $\mathcal{E}_\psi^q$ for the equivalence class at node q of the query structure ψ. These sets can be collected during traversal. A valid member $\phi \in \Im$ for a subsumption query ψ in terms of coreferences is one where $\mathcal{E}_\psi^q \subseteq \mathcal{E}_\phi^q$, for every q. This condition has to be checked only at nodes where a new coreference is introduced, whereas the information that nodes are coreferent has to be stored also at paths that reach beyond such an introduction node.

In the index, the sets $\mathcal{E}_\phi^q$ are only encoded implicitly. Every non-trivial equivalence class (non-singleton set) is replaced by its *representative* which is the lowest node in the canonical order that belongs to this class. Analogous to the bit vectors for types, there are bit vectors b_r^q for all members, using r as the representative at node q.

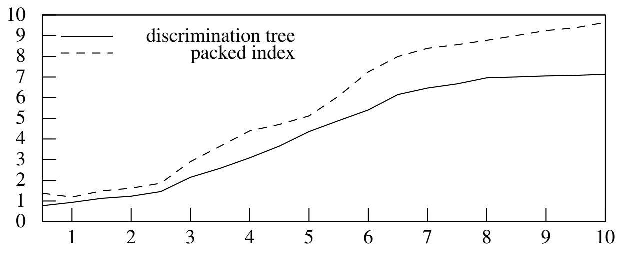
To test the subset condition above, we have to make sure that all members of $\mathcal{E}_\psi^q$ point to the same representative for some index member. Thus for the valid contexts, we have to compute $\bigvee_{r\in\mathcal{E}_\phi^q}(\bigwedge_{q'\in\mathcal{E}_\psi^q} b_r^{q'})$.

For the generalization case, computing that the equivalence class of the index is a subset of that in the query, we have to check if a representative node r in $\mathcal{E}_\phi^q$ is not an element of $\mathcal{E}_\psi^q$, and to remove all members where this holds by computing $a \wedge (\bigwedge_{r\in\mathcal{E}_\phi^q \wedge r\notin\mathcal{E}_\psi^q} \neg b_r^q)$.

5 Measurements

To be able to compare the performance of the two index structures, we have designed the following syn-

[1]We plan to implement a thread-safe unifier in the near future, which will enable us to assess a parallel version of the discrimination, tree. For the approximation, however, the packed index outperforms the discrimination tree even in sequential mode.



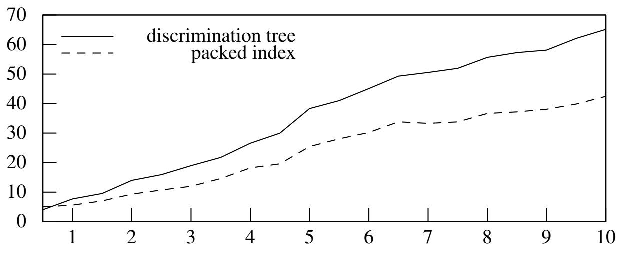


Figure 1: Time to get all subsumed (left) and all unifiable members (right) from the index. The graphs show the time needed for 10,000 operations (in 100 ms steps on the vertical axis) in relation to the index size ($\times 1,000$ members). Experiments were executed on a 2.4 GHz Quad-Core Opteron with 64GB main memory which only ran the test process. We would like to draw the reader's attention to the difference in scale on the vertical axes.

thetic experiment: 20 million feature structures generated during the approximation were dumped, from which three random sets were selected, such that indices of up to 10,000 TFSs are created, and three random sets of 10,000 query structures. The numbers from the resulting nine experiments were averaged to remove effects that are due to characteristics of the data set. Every index and query set was used to perform two of the operations executed in the CF approximation: (1) determining the set of all subsumed elements in the index and (2) returning all unifiable elements (see Figure 1). Where the index only acts as a filter, the time to compute the correct result is included, i.e., the full unification or subsumption. The indexing method we are comparing against is described in detail in (Kiefer and Krieger, 2004) and is an instance of discrimination tree indexing.

The performance of the subsumed members operation is slightly worse, while the unifiability test is superior as it avoids many of the costly full unification operations. At first sight, the graphs in figure 1 don't seem to indicate a large improvement. However, we ran these synthetic experiments to demonstrate the raw performance; When applying it to grammar approximation, the picture is quite different. The unifiability test is the first and therefore most important step, and together with the potential that it can be used almost effortlessly with low locking overhead in concurrent environments, the packed index by far outperforms the discrimination tree. To show this, we ran the approximation in three different setups, with the discrimination tree, and the packed index without and with parallelization. The numbers below show the *real time* needed to complete the third iteration of the fixpoint computation of the CF approximation:

1. discrimination tree: 139,279 secs
2. packed sequential: 49,723 secs (2.8× faster)
3. packed parallel: **15,309** secs (9.1× faster)

To measure the *space requirements*, we used the 59,646 feature structures at the end of this third iteration and, running the system in a profiling environment, stored it in both implementations. This gave us, subtracting the 144 MB that the profiler showed after loading the grammar, 103 MB for the packed and 993 MB for the discrimination tree index, a factor of **9.64**.
As is true for most techniques that optimize feature structure operations, the effectiveness strongly depends on the way they are used in the application, e.g., the number of executions of the different operations, the implementation of basic functions like type unification or subsumption, etc. This means that the presented method, while very effective for our application, may have to be adapted by others to produce a significant gain. And there is still a lot of room for improvement, e.g., by combining the tree and packed index, or tuning the implementation of the bit set operations, which will often operate on sparse sets.

6 Summary and Outlook

In this paper, we have described a new indexing method for typed feature structures that deviates from the index techniques mentioned in the introduction. Our measurements have shown that the new method outperforms the discrimination tree index, at least when applied to our approximation experiments.
We note here that the new methods might also be important to other areas in computational linguistics, such as lexicon lookup for a large lexical data base or tree bank, unification-based parsing under packing, or even chart-based generation. Other areas involving record-like structures would also benefit from our approach and we envisage semantic repositories for storing RDF graphs, as similar operations to unifiability and subsumption are of importance to OWL.

Acknowledgements

The research described here was financed by the German Federal Ministry of Education and Research (BMBF) through the project *∂eepen∂ance* (contract no. 01IW11003) and by the EU FP7-ICT Programme projects ALIZ-E (grant agreement no. 248116) and TrendMiner (grant agreement no. 287863). The authors would like to thank the reviewers for their comments.

References

Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.

Dan Flickinger. 2011. Accuracy vs. robustness in grammar engineering. In E.M. Bender and J.E. Arnold, editors, *Language from a Cognitive Perspective: Grammar, Usage, and Processing*, pages 31–50. CSLI Publications, Stanford.

Thilo Goetz, Robin Lougee-Heimer, and Nicolas Nicolov. 2001. Efficient indexing for typed feature structures. In *Proceedings of Recent Advances in Natural Language Processing, Tzigov Chark*.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Robert T. Kasper and William C. Rounds. 1986. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, ACL-86*, pages 257–266.

Bernd Kiefer and Hans-Ulrich Krieger. 2002. A context-free approximation of Head-Driven Phrase Structure Grammar. In S. Oepen, D. Flickinger, J. Tsuji, and H. Uszkoreit, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, pages 49–76. CSLI Publications.

Bernd Kiefer and Hans-Ulrich Krieger. 2004. A context-free superset approximation of unification-based grammars. In H. Bunt, J. Carroll, and G. Satta, editors, *New Developments in Parsing Technology*, pages 229–250. Kluwer Academic Publishers.

John T. Maxwell III and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, pages 173–190. Kluwer. Also available as report SSL-90-06, XEROX, System Sciences Laboratory, Palo Alto, 1990.

Cosmin Munteanu. 2003. Indexing methods for efficient parsing with typed feature structure grammars. In *Proceedings of the 6th Pacific Association for Computational Linguistics Conference*.

Takashi Ninomiya and Takaki Makino. 2002. An indexing scheme for typed feature structures. In *In Proceedings of the 19th International Conference on Computational Linguistics (COLING-02)*, pages 1248–1252.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, Chicago.

William C. Rounds and Robert T. Kasper. 1986. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual Symposium of the IEEE on Logic in Computer Science*.

Stuart M. Shieber. 1984. The design of a computer language for linguistic information. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 362–366.

Learning Bilingual Categories in Unsupervised Inversion Transduction Grammar Induction

Markus SAERS and **Dekai WU**
Human Language Technology Center
Dept. of Computer Science and Engineering
Hong Kong University of Science and Technology
{masaers|dekai}@cs.ust.hk

Abstract

We present the first known experiments incorporating unsupervised bilingual nonterminal category learning within end-to-end fully unsupervised transduction grammar induction using matched training and testing models. Despite steady recent progress, such induction experiments until now have not allowed for learning differentiated nonterminal categories. We divide the learning into two stages: (1) a *bootstrap* stage that generates a large set of categorized short transduction rule hypotheses, and (2) a *minimum conditional description length* stage that simultaneously prunes away less useful short rule hypotheses, while also iteratively segmenting full sentence pairs into useful longer categorized transduction rules. We show that the second stage works better when the rule hypotheses have categories than when they do not, and that the proposed conditional description length approach combines the rules hypothesized by the two stages better than a mixture model does. We also show that the compact model learned during the second stage can be further improved by combining the result of different iterations in a mixture model. In total, we see a jump in BLEU score, from 17.53 for a standalone minimum description length baseline with no category learning, to 20.93 when incorporating category induction on a Chinese–English translation task.

1 Introduction

Even simple lexical translations are surprisingly context-dependent, in this paper we aim to learn a translation model that can base contextual translation decision on more than lexical n-grams, both in the input and output language. In a syntactic translation system such as inversion transduction grammars (ITGs), this can be achieved with unsupervised bilingual category induction. Surface-based and hierarchical models only use output language n-grams, and syntactic model typically choose the categories from either the input or the output language, or attempts to heuristically synthesize a set of bilingual categories from the two monolingual sets. In contrast, we attempt to learn a set of bilingual categories without supervision, which gives a unique opportunity to strike a good balance between the two approaches.

The specific translation of words and segments depend heavily on the context. A grammar-based translation model can model the context with nonterminal categories, which allows (a) moving beyond n-grams (as a compliment to the language model prior which is typically preserved), and (b) taking both the input and output language context into account. Typical syntactic MT systems either ignore categories (bracketing ITGs and hierarchical models), or derive the categories from tree-banks, which relies on choosing the set of categories from either language, or heuristically synthesize it from both; both approaches eliminates the full benefits of (b). In contrast, unsupervised induction of a bilingual category set has the potential to fully take advantage of (b).

Recent work has seen steady improvement in translation quality for completely unsupervised transduction grammar induction under end-to-end purely matched training and testing model conditions. In this paper, we take a further step along this line of research by incorporating unsupervised bilingual category induction into the learning process. To our knowledge, no previous attempt has been made to incorporate bilingual categories under such conditions. Matching the training and testing models as closely as possible is a fundamental principle taken for granted in most applications of machine learning, but for machine translation it has been the norm to see very different assumptions during training and testing, which makes it difficult to assess the effects of changing or tweaking the model—the observed effect may not be repeatable. By matching training and testing conditions, this risk is minimized.

A bilingual category is similar to a monolingual category in that it is realized as the left-hand side label of a (transduction) grammar rule, but differ in what it represents. A monolingual category only encodes how something relates to other parts of the language, a bilingual category should encode how a translation equivalence relates to other translation equivalences. It needs to account for the relationship between two languages as well as the relationship between the parts of the individual languages. This makes the usage of existing tagging schemes problematic. It would be possible to use the categories from either of the languages (assuming they are languages with enough resources) and impose these on the other language. This could work for closely related languages, but we are translating between English and Chinese: two very different languages, and we know that the category sets of either language is a poor fit for the other. Another possibility is to take the cross-product of the monolingual category sets, but handling such a large set of categories becomes unwieldy in ITG induction, a process which is resource intensive as is, without exploding the set of nonterminals. Instead, we opt for unsupervised learning of the bilingual categories during induction of the ITG itself.

The novel learning method we propose consists of an initial hypothesis generator that proposes (a) short lexical translations and (b) nonterminal categories, screened by a mechanism that (c) verifies the usefulness of the hypotheses while (d) uses them to further generate longer transduction rules. For convenience, our implementation breaks this into two stages: one that generates a large set of short transduction rule hypotheses, and another that iteratively segments long transduction rules (initialized from the sentence pairs in the training data) by trying to reuse a minimal subset of the hypotheses while chipping away at the long sentence pair rules until the conditional description length is minimized.

The paper is structured so that, after giving the background situated within the context of relevant related research (Section 2), we define the proposed conditional description length approach, which represents the ideal model search (Section 3). We then detail the two stages of our proposed learning algorithm, which represents our approximation of the search problem (Sections 4 and 5). After the theory we detail the particular experiments we conducted (Section 6) and the results from those experiments (Section 7). Finally, we offer some conclusions (Section 8).

2 Background

Description length has been used before to drive iterative segmenting ITG learning (Saers et al., 2013). We will use their algorithm as our baseline, but the simple mixture model we used then works poorly with our ITG with categories. Instead, we propose a tighter incorporation, where the rule segmenting learning is biased towards rules that are present in the categorized ITG.

We refer to this objective as minimizing *conditional description length*, since technically, the length of the ITG being segmented is *conditioned* on the categorized ITG. Conditional description length (CDL) is detailed in Section 3. The minimum CDL (MCDL) objective differs from the simple mixture model in that it separates the rule hypotheses into two groups: the ones that are used during segmentation and therefor carries over to the final induced ITG, and those that do not and are effectively filtered out. As we will see, MCDL far outperforms the mixture model when one of the ITGs has categories and the other does not.

A problem with the description length family of learning objectives is that they tend to commit to a segmentation when it would be wise to keep the unsegmented rule *as well*—a significant part of the success of phrase-based translation models comes from their approach to keep all possible segmental translations (that do not violate the prerequisite word alignment). We will show that we can counter this by combining different iterations of the same segmentation process into a single grammar, which gives a significant bump in BLEU scores.

By insisting on the fundamental machine learning principle of matching the training model to the testing model, we do forfeit the short term boost in BLEU that is typically seen when embedding a learned ITG in the midst of the common heuristics employed in statistical machine translation. For example, Cherry and Lin (2007), Zhang et al. (2008), Blunsom et al. (2008), Blunsom et al. (2009), Haghighi et al. (2009), Saers and Wu (2009), Blunsom and Cohn (2010), Burkett et al. (2010), Riesa and Marcu (2010), Saers et al. (2010), Saers and Wu (2011), Neubig et al. (2011), and Neubig et al. (2012) all plug some aspect of the ITGs they learn into training pipelines for existing, mismatched decoders, typically in the form of the word alignment that an ITG imposes on a parallel corpus as it is biparsed. Our own past work has also taken similar approaches, but it is not necessary to do so—instead, any ITG can be used for decoding by directly parsing with the input sentence as a hard constraint, as we do in this paper. Although it allows you to tap into the vast engineering efforts that have gone into perfecting existing decoders, it also prevents you from surpassing them in the long run. The motivation for our present series of experiments is that, as a field we are well served by tackling the fundamental questions as well, and not exclusively focusing on engineering short term incremental BLEU score boosts where the quality of an induced ITG itself is obscured because it is embedded within many other heuristic algorithms.

When the structure of an ITG is induced without supervision, it is possible to get an effect that resembles MDL. Zhang et al. (2008) impose a sparsity prior over the rule probabilities to prevent the search from having to consider all the rules found in the Viterbi biparses. Blunsom et al. (2008), Blunsom et al. (2009), Blunsom and Cohn (2010), Neubig et al. (2011), and Neubig et al. (2012) use Gibbs sampling to learn ITGs with priors over the rule structures that serve a similar purpose to the model length component of description length. All of the above evaluate their models by biparsing the training data and feeding the imposed word alignment into an existing, mismatched SMT learning pipeline.

Transduction grammars can also be induced with supervision from treebanks, which cuts down the search space by enforcing external constraints (Galley et al., 2006). Although this constitutes a way to borrow nonterminal categories that help the translation model, it complicates the learning process by adding external constraints that are bound to match the translation model poorly.

3 Conditional description length

Conditional description length (CDL) is a general method for evaluating a model and a dataset given a preexisting model. This makes it ideal for augmenting an existing model with a variant model of the same family. In this paper we will apply this to augment an existing inversion transduction grammar (ITG) with rules that are found with a different search strategy. CDL is similar to description length (Solomonoff, 1959; Rissanen, 1983), but the length calculations are subject to additional constraints. When minimum CDL (MCDL) is used as a learning objective, all the desired properties of minimum description length (MDL) are retained: the model is allowed to become less certain about the data provided that the it shrinks sufficiently to compensate for the loss in precision. MDL is a good way to prevent over-fitting, and MCDL retains this property, but for the task of inducing a model that is specifically tailored toward augmenting an existing model. Formally, the conditional description length is:

$$DL\left(\Phi, D|\Psi\right) = DL\left(D|\Phi, \Psi\right) + DL\left(\Phi|\Psi\right)$$

where Ψ is the fixed preexisting model, Φ is the model being induced, and D is the data. The total unconditional length is:

$$\begin{aligned} &DL\left(\Psi, \Phi, D\right) = \\ &\qquad DL\left(D|\Phi, \Psi\right) + DL\left(\Phi|\Psi\right) + DL\left(\Psi\right) \end{aligned}$$

In minimizing CDL, we fix $DL\left(\Psi\right)$ instead of allowing Ψ to vary as we would in full MCDL; to be precise, we seek:

$$\begin{aligned} &\underset{\Phi}{\operatorname{argmin}}\, DL\left(\Psi, \Phi, D\right) \\ &= \underset{\Phi}{\operatorname{argmin}}\, DL\left(D|\Phi, \Psi\right) + DL\left(\Phi|\Psi\right) + DL\left(\Psi\right) \\ &= \underset{\Phi}{\operatorname{argmin}}\, DL\left(\Phi, D|\Psi\right) \\ &= \underset{\Phi}{\operatorname{argmin}}\, DL\left(D|\Phi, \Psi\right) + DL\left(\Phi|\Psi\right) \end{aligned}$$

To measure the CDL of the data, we turn to information theory to count the number of bits needed to encode the data given the two models under an optimal encoding (Shannon, 1948), which gives:

$$DL\left(D|\Phi, \Psi\right) = -\lg P\left(D|\Phi, \Psi\right)$$

To measure the CDL of the model, we borrow the encoding scheme for description length presented in Saers et al. (2013), and define the conditional description length as:

$$DL\left(\Phi|\Psi\right) \equiv DL\left(\Phi\right) - DL\left(\Phi \cap \Psi\right)$$

To determine whether a model Φ has a shorter conditional description length, than another model Φ', it is sufficient to be able to subtract one length from the other. For the model length, this is trivial as we merely have to calculate the length of the difference between the two models in our theoretical encoding. For data length, we need to solve:

$$\begin{aligned} & DL\left(D|\Phi', \Psi\right) - DL\left(D|\Phi, \Psi\right) \\ & \quad = -\lg P\left(D|\Phi', \Psi\right) - -\lg P\left(D|\Phi, \Psi\right) \\ & \quad = -\lg \frac{P\left(D|\Phi', \Psi\right)}{P\left(D|\Phi, \Psi\right)} \end{aligned}$$

4 Generating rule hypotheses

In the first stage of our learning approach, we generate a large set of possible rules, from which the second stage will choose a small subset to keep. The goal of this stage is to keep the *recall* high with respect to a theoretical "optimal ITG", *precision* is achieved in the second stage. We rely on chunking and category splitting to generate this large set of rule hypotheses.

To generate these high-recall ITGs, we will follow the bootstrapping approach presented in Saers et al. (2012), and start with a finite-state transduction grammar (FSTG), do the chunking and category splitting within the FSTG framework before transferring the resulting grammar to a corresponding ITG. This is likely to produce an ITG that performs poorly on its own, but may be informative in the second stage.

5 Segmenting rules

In the second stage of our learning approach, we segment rules explicitly representing the entire training data, into smaller—more general—rules, reusing rules from the first stage whenever we can. By driving the segmentation-based learning with a minimum description length objective, we are learning a very concise ITG, and by conditioning the description length on the rules hypothesized in the first stage, we separate the good rule hypotheses from the bad: the good rules—along with their categorizing left-hand sides—are reused and the bad are not.

In this work, we are only considering segmentation of lexical rules, which keeps the ITG in normal form, greatly simplifying processing without altering the expressivity. A lexical ITG rule has the form $A \rightarrow e_{0..T}/f_{0..V}$, where A is the left-hand side nonterminal—the category, $e_{0..T}$ is a sequence of T (from position 0 up to but not including position T) L_0 tokens and $f_{0..V}$ is a sequence of V (from position 0 up to but not including position V) L_1 tokens. When segmenting this rule, three new rules are produced which take one of the following forms depending on whether the segmentation is inverted or not:

$$\begin{array}{rcr} A \rightarrow [BC] & & A \rightarrow \langle BC \rangle \\ B \rightarrow e_{0..S}/f_{0..U} & \text{or} & B \rightarrow e_{0..S}/f_{U..V} \\ C \rightarrow e_{S..T}/f_{U..V} & & C \rightarrow e_{S..T}/f_{0..U} \end{array}$$

All possible splits of the terminal rule can be accounted for by choosing the identities of B, C, S and U, as well as whether the split it straight or inverted.

Algorithm 1 Iterative rule segmenting learning driven by minimum conditional description length.

```
Φ                                          ▷ The ITG being induced
Ψ                         ▷ The ITG the learning is conditioned on
repeat
    $\delta_{sum} \leftarrow 0$
    $bs \leftarrow$ collect_biaffixes($\Phi$)
    $b\delta \leftarrow []$
    for all $b \in bs$ do
        $\delta \leftarrow$ eval_cdl($b, \Psi, \Phi$)
        if $\delta < 0$ then
            $b\delta \leftarrow [b\delta, \langle b, \delta \rangle]$
    sort_by_delta($b\delta$)
    for all $\langle b, \delta \rangle \in b\delta$ do
        $\delta' \leftarrow$ eval_cdl($b, \Psi, \Phi$)
        if $\delta' < 0$ then
            $\Phi \leftarrow$ make_segmentations($b, \Phi$)
            $\delta_{sum} \leftarrow \delta_{sum} + \delta'$
until $\delta_{sum} \geq 0$
return $\Phi$
```

The pseudocode for the iterative rule segmenting learning algorithm driven by minimal conditional description length can be found in Algorithm 1. It uses the methods `collect_biaffixes`, `eval_cdl`, `sort_by_delta` and `make_segmentations`. These methods collect all biaffixes in the rules of an ITG, evaluate the difference in conditional description length, sorts candidates by these differences, and commits to a given set of candidates, respectively. To evaluate the CDL of a proposed set of candidate segmentations, we need to calculate the difference in CDL between the current model, and the model that would result from committing to the candidate segmentations:

$$\begin{aligned} &DL\left(D, \Phi'|\Psi\right) - DL\left(D, \Phi|\Psi\right) \\ &\quad = DL\left(D|\Phi', \Psi\right) - DL\left(D|\Phi, \Psi\right) \\ &\qquad + DL\left(\Phi'|\Psi\right) - DL\left(\Phi|\Psi\right) \end{aligned}$$

The model lengths are trivial, as we merely have to encode the rules that are removed and inserted according to our encoding scheme and plug in the summed lengths in the above equation. This leaves the length of the data, which would be:

$$DL\left(D|\Phi', \Psi\right) - DL\left(D|\Phi, \Psi\right) = -\lg \frac{P\left(D|\Phi', \Psi\right)}{P\left(D|\Phi, \Psi\right)}$$

For the sake of convenience in efficiently calculating this probability, we make the simplifying assumption that:

$$P\left(D|\Phi, \Psi\right) \approx P\left(D|\Phi\right) = P\left(D|\theta\right)$$

where θ represents the model parameters, which reduces the difference in data CDL to

$$-\lg \frac{P\left(D|\theta'\right)}{P\left(D|\theta\right)}$$

which lets us determine the probability through biparsing with the model being induced. Biparsing is, however, a very expensive operation, and we are making relatively small changes to the ITG, so we will further assume that we can estimate the CDL difference in closed form based on the model parameters. Given that we are splitting the rule r_0 into the three rules r_1, r_2 and r_3, and that the probability mass of r_0 is distributed uniformly over the new rules, the new grammar parameters θ' will be identical to θ, except that:

$$\begin{aligned} \theta'_{r_0} &= 0 \\ \theta'_{r_1} &= \theta_{r_1} + \frac{1}{3}\theta_{r_0} \\ \theta'_{r_2} &= \theta_{r_2} + \frac{1}{3}\theta_{r_0} \\ \theta'_{r_3} &= \theta_{r_3} + \frac{1}{3}\theta_{r_0} \end{aligned}$$

We estimate the CDL of the corpus given this new parameters to be:

$$-\lg \frac{P\left(D|\theta'\right)}{P\left(D|\theta\right)} \approx -\lg \frac{\theta'_{r_1}\theta'_{r_2}\theta'_{r_3}}{\theta_{r_0}}$$

To generalize this to a set of rule segmentations, we construct the new parameters θ' to reflect all the changes in the set in a first pass, and then sum the differences in CDL for all the rule segmentations with the new parameters in a second pass.

6 Experimental setup

The learning approach we chose has two stages, and in this section we describe the different ways of using these two stages to arrive at a final ITG, and how we intend to evaluate the quality of those ITGs.

For the first stage, we will use the technique described in Saers et al. (2012) to start with a finite-state transduction grammar (FSTG) and perform chunking before splitting the nonterminal categories and moving the FSTG into ITG form. We will perform one round of chunking, and two rounds of category splitting (resulting in 4 nonterminals and 4 preterminals, which becomes 8 nonterminals in the ITG form). Splitting all categories is guarnteed to at least double the size of the grammar, which makes is impractical to repeat more times. At each stage, we run a few iterations of expectation maximization using the algorithm detailed in Saers et al. (2009) for biparsing. For comparison we also bootstrap a comparable ITG that has not had the categories split. Before using either of the bootstrapped ITGs, we eliminate all rules that do not have a probability above a threshold that we fixed to 10^{-50}. This eliminates the highly unlikely rules from the ITG.

For the second stage, we use the iterative rule segmentation learning algorithm driven by minimum conditional description length that we introduced in Section 5. We will try three different variants on this algorithm: one without an ITG to condition on, one conditioned on the chunked ITG, and one conditioned on the chunked ITG with categories. The first variant is completely independent from the chunked ITGs, so we will also try to create mixture models with it and the chunked ITGs.

Since the MCDL objective tends to segment large rules and count on them being recreatable when needed, many of the longer rules that would be good

Table 1: Experimental results. *Chunked* is the base model, which has categories added to produce *chunked w/categories*. *Segmented* corresponds to the second learning stage, which can be done in isolation (*only*), *mixed with* a base model, or *conditioned on* a base model.

Model	**BLEU**	**NIST**	**Categories**
Chunked ITG only	3.76	0.0119	1
Chunked ITG w/categories only	9.39	0.7481	8
Segmented ITG only	17.53	4.5409	1
Segmented ITG mixed with chunked ITG	10.23	0.2886	1
Segmented ITG mixed with chunked ITG w/categories	12.06	1.1415	8
Segmented ITG conditioned on chunked ITG	17.04	4.4920	1
Segmented ITG conditioned on chunked ITG w/categories	19.02	4.6079	8
... with iterations combined	20.20	4.8287	8
... and improved search parameters	20.93	4.8426	8

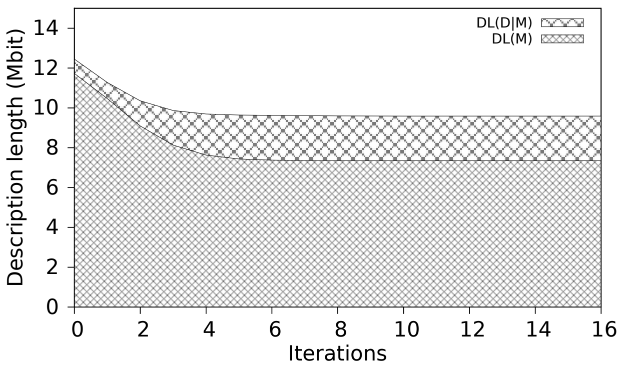


Figure 1: Description length in bits over the different iterations of segmenting search. The lower portion represents the conditional description length of the model, $DL\left(\Phi|\Psi\right)$, and the upper portion represents the conditional description length of the data given the model, $DL\left(D|\Phi,\Psi\right)$.

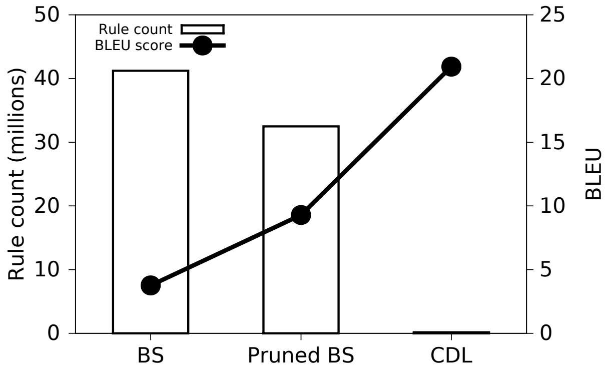


Figure 2: Rule count versus BLEU scores for the bootstrapped ITG, the pruned bootstrapped ITG and the segmented ITG conditioned on the pruned bootstrapped ITG.

to have when translating are not explicitly in the grammar. This is potentially a source of translation mistakes, and to investigate this, we create a mixture model from iterations of the segmenting learning process leading up to the learned ITG.

All the above outlined ITGs are trained using the IWSLT07 Chinese–English data set (Fordyce, 2007), which contains 46,867 sentence pairs of training data, and 489 Chinese sentences with 6 English reference translations each as test data; all the sentences are taken from the traveling domain. Since the Chinese is written without whitespace, we use a tool that tries to clump characters together into more “word like” sequences (Wu, 1999).

To test the learned ITGs, we use them as translation systems with our in-house ITG decoder. The decoder uses a CKY-style parsing algorithm (Cocke, 1969; Kasami, 1965; Younger, 1967) and cube pruning (Chiang, 2007) to integrate the language model scores. For language model, we use a trigram language model trained with the SRILM toolkit (Stolcke, 2002) on the English side of the training corpus. To evaluate the resulting translations, we use BLEU (Papineni et al., 2002) and NIST (Doddington, 2002).

7 Results

In this section we present the empirical results: bilingual categories help translation quality under the experimental conditions detailed in the previous section. The results are summarized in Table 1. As predicted the base *chunked only* ITG fares poorly, while the categories help a great deal in the *chunked w/categories* *only* ITG—though the scores are not very reliable when in this low range.

The trade-off between model and data size during segmentation conditioned on the ITG with categories is illustrated in Figure 1. It starts out with most of the total description being used to describe the model, and very little to describe the data. This is the degenerate situation where every sentence pair is its own lexical rule. Then there is a sharp drop in model size with a slight increase in data size. This is where the most dramatic generalizations take place. It levels off fairly quickly, and the minor adjustments that take place on the plateau still represent valid generalizations, they just have a very small effect on the over-all description length of either the model or the data.

That the chunked ITG with split categories suffers from having too many irrelevant rules is clearly seen in Figure 2, where we plotted the number of rules contrasted to the BLEU score. Merely pruning to a threshold helps somewhat, but the sharper improvement—both in terms of model size and BLEU score—is seen with the filtering that MCDL represents.

A number of interesting lessons emerge from the results, as follows.

7.1 Minimum CDL outperforms mixture modeling

The segmenting approach works as expected (*segmented only*), essentially reproducing the results reported by Saers et al. (2013) for this style of bilingual grammar induction.

Interestingly, however, where they had success with the mixture model combining the base ITGs with the ITG learned through the segmenting approach (*segmented mixed with...*), we see a significant drop in translation quality. This may be because we have categories in our base ITG and they do not.

7.2 Category induction strongly improves minimum CDL learning

When we use the base ITGs to condition the segmenting approach, we see something interesting. The base ITG that has categories causes a sharp 1.5 BLEU point rise in translation quality (compare *segmented only* to *segmented conditioned on chunked w/categories*).

In contrast, the base ITG that does not have categories causes a slight 0.5 BLEU point *fall* in translation quality (compare *segmented only* to *segmented conditioned on chunked*).

7.3 Redundant segmental rule granularities help

As mentioned, the minimum description length objective may be theoretically nice, but it also relies on the learned ITG being able to reassemble segmented rules with fairly high fidelity at decoding time. To demand that all transduction rules are reduced to exactly a single right level of granularity may be a bit of a tall order.

Our way to test this was to uniformly mix the ITGs at different iterations though the segmenting process. By mixing the ITG after each iteration up to the one labeled *segmented conditioned on chunked w/categories*, we get the same model labeled *...with iterations combined*, which secures an additional 1.18 BLEU points.

7.4 Tuning search parameters

Lastly, for the best approach, we further experimented with adjusting the parameters somewhat. Pruning the base grammar harder (a threshold of 10^{-10} instead of 10^{-50}), and allowing for a wider beam (100 items instead of 25) during the parsing part of the segmenting learning approach, we see the BLEU score rise to 20.93.

7.5 Analysis of learned rules

A manual inspection of the content of the categories learned reveals that the main nonterminal contains mainly structural rules, segments that it could not segment further. The latter type of rules varies from full clauses such as that ' s a really beautiful dress/真是件漂亮的衣服 to reasonable translation units such as Kazuo Yamada/Ｋ ａ ｚ ｕ ｏ Ｙ ａ ｍ ａ ｄ ａ , which is really hard to capture because each Latin character on the Chinese side is its own individual token whereas the English side has whole names as individual tokens.

A second nonterminal category contains punctuation such as full stop and question mark, along with , sir/，先生, which can be considered as a form of punctuation in the domain of the training data.

A third nonterminal category contains personal pronouns in subject form (I, we, he, and also ambiguous pronouns that could be either subject or object form such as you and it) paired up with their respective Chinese translations. It also contains please/请, which—like pronouns in subject form—occurs frequently in the beginning of sentence pairs.

A fourth nonterminal category contains pairs such as can/吗, do you/吗, is/吗, could you/吗 and will you/吗 — instances where Chinese typically makes a statement, possibly eliding the pronoun, and adds the question particle (吗) to the end, and where English prefixes that statement with a verb; both languages use a question mark in the particular training data we used. The main nonterminal learned that this category typically was used in inverted rules, and the other translation equivalences conform to that pattern. They include where/在哪, where the Chinese more literally translates to on/at which, what/什么 which is a good translation, and have/了, where the English auxiliary verb corresponds well to the Chinese particle signaling *perfect aspect*—that the action described in the preceeding clause is finished.

Other categories appear to still be consolidating, with a mix of nouns, verbs, adjectives, and adverbials. Chinese words and phrases typically can function as any of these, so it is possible that differentiating them may require increased emphasis on the English half of the rules.

Although the well-formed categories are few and somewhat trivial, it is very encouraging to see them emerging without any form of human supervision. Future work will expand to continue learning an even wider range of categories.

8 Conclusions

We have presented the first known experiments for incorporating bilingual category learning within completely unsupervised transduction grammar induction under end-to-end matched training and testing model conditions. The novel approach employs iterative rule segmenting driven by a minimum conditional description length learning objective, conditioned on a prior defined by a stochastic ITG containing automatically induced bilingual categories. We showed that this learning objective is superior to the previously used mixture model, when bilingual categories are involved. We also showed that the segmenting learning algorithm may be committing too greedily to segmentations since combining the ITGs with different degrees of segmentation gives better scores than any single point in the segmentation process; this points out an interesting avenue of future research. We further saw that the segmenting minimization of conditional description length can pick up some of the signal in categorization that was buried in noise in the base ITG the induction was conditioned on, leading to an ITG with much clearer categories. In total we have seen an improvement of 3.40 BLEU points due to the incorporation of unsupervised category induction.

Acknowledgements

This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under BOLT contract no. HR0011-12-C-0016, and GALE contract nos. HR0011-06-C-0022 and HR0011-06-C-0023; by the European Union under the FP7 grant agreement no. 287658; and by the Hong Kong Research Grants Council (RGC) research grants GRF620811, GRF621008, and GRF612806. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the EU, or RGC.

References

Phil Blunsom and Trevor Cohn. 2010. Inducing synchronous grammars with slice sampling. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010)*, pages 238–241, Los Angeles, California, June.

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. Bayesian synchronous grammar induction. In *Advances in Neural Information Processing Systems 21 (NIPS 21)*, Vancouver, Canada, December.

Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles Osborne. 2009. A Gibbs sampler for phrasal synchronous grammar induction. In *Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP 2009)*, pages 782–790, Suntec, Singapore, August.

David Burkett, John Blitzer, and Dan Klein. 2010. Joint parsing and alignment with weakly synchronized grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010)*, pages 127–135, Los Angeles, California, June.

Colin Cherry and Dekang Lin. 2007. Inversion transduction grammar for joint phrasal translation modeling. In *Syntax and Structure in Statistical Translation (SSST)*, pages 17–24, Rochester, New York, April.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

John Cocke. 1969. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *The second international conference on Human Language Technology Research (HLT '02)*, pages 138–145, San Diego, California.

C. S. Fordyce. 2007. Overview of the IWSLT 2007 evaluation campaign. In *International Workshop on Spoken Language Translation (IWSLT 2007)*, pages 1–12.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL 2006)*, pages 961–968, Sydney, Australia, July.

Aria Haghighi, John Blitzer, John DeNero, and Dan Klein. 2009. Better word alignments with supervised ITG models. In *Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP 2009)*, pages 923–931, Suntec, Singapore, August.

Tadao Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-00143, Air Force Cambridge Research Laboratory.

Graham Neubig, Taro Watanabe, Eiichiro Sumita, Shinsuke Mori, and Tatsuya Kawahara. 2011. An unsupervised model for joint phrase alignment and extraction. In *49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL HLT 2011)*, pages 632–641, Portland, Oregon, June.

Graham Neubig, Taro Watanabe, Shinsuke Mori, and Tatsuya Kawahara. 2012. Machine translation without words through substring alignment. In *50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pages 165–174, Jeju Island, Korea, July.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 311–318, Philadelphia, Pennsylvania, July.

Jason Riesa and Daniel Marcu. 2010. Hierarchical search for word alignment. In *48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 157–166, Uppsala, Sweden, July.

Jorma Rissanen. 1983. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, June.

Markus Saers and Dekai Wu. 2009. Improving phrase-based translation via word alignments from stochastic inversion transduction grammars. In *Third Workshop on Syntax and Structure in Statistical Translation (SSST-3)*, pages 28–36, Boulder, Colorado, June.

Markus Saers and Dekai Wu. 2011. Principled induction of phrasal bilexica. In *15th Annual Conference of the European Association for Machine Translation (EAMT-2011)*, pages 313–320, Leuven, Belgium, May.

Markus Saers, Joakim Nivre, and Dekai Wu. 2009. Learning stochastic bracketing inversion transduction grammars with a cubic time biparsing algorithm. In *11th International Conference on Parsing Technologies (IWPT'09)*, pages 29–32, Paris, France, October.

Markus Saers, Joakim Nivre, and Dekai Wu. 2010. Word alignment with stochastic bracketing linear inversion transduction grammar. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010)*, pages 341–344, Los Angeles, California, June.

Markus Saers, Karteek Addanki, and Dekai Wu. 2012. From finite-state to inversion transductions: Toward unsupervised bilingual grammar induction. In *24th International Conference on Computational Linguistics (COLING 2012)*, pages 2325–2340, Mumbai, India, December.

Markus Saers, Karteek Addanki, and Dekai Wu. 2013. Combining top-down and bottom-up search for unsupervised induction of transduction grammars. In *Seventh Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-7)*, pages 48–57, Atlanta, Georgia, June.

Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October.

Ray J. Solomonoff. 1959. A new method for discovering the grammars of phrase structure languages. In *International Federation for Information Processing Congress (IFIP)*, pages 285–289.

Andreas Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *7th International Conference on Spoken Language Processing (ICSLP2002 - INTERSPEECH 2002)*, pages 901–904, Denver, Colorado, September.

Zhibiao Wu. 1999. LDC Chinese segmenter.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Hao Zhang, Chris Quirk, Robert C. Moore, and Daniel Gildea. 2008. Bayesian learning of non-compositional phrases with synchronous parsing. In *46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, pages 97–105, Columbus, Ohio, June.

Comparative Evaluation of Argument Extraction Algorithms in Discourse Relation Parsing

Evgeny A. Stepanov and **Giuseppe Riccardi**
Department of Information Engineering and Computer Science,
University of Trento, Trento, Italy
{stepanov,riccardi}@disi.unitn.it

Abstract

Discourse relation parsing is an important task with the goal of understanding text beyond the sentence boundaries. One of the subtasks of discourse parsing is the extraction of argument spans of discourse relations. A relation can be either intra-sentential – to have both arguments in the same sentence – or inter-sentential – to have arguments span over different sentences. There are two approaches to the task. In the first approach the parser decision is not conditioned on whether the relation is intra- or inter-sentential. In the second approach relations are parsed separately for each class. The paper evaluates the two approaches to argument span extraction on Penn Discourse Treebank explicit relations; and the problem is cast as token-level sequence labeling. We show that processing intra- and inter-sentential relations separately, reduces the task complexity and significantly outperforms the single model approach.

1 Introduction

Discourse analysis is one of the most challenging tasks in Natural Language Processing, that has applications in many language technology areas such as opinion mining, summarization, information extraction, etc. (see (Webber et al., 2011) and (Taboada and Mann, 2006) for detailed review). With the availability of annotated corpora, such as Penn Discourse Treebank (PDTB) (Prasad et al., 2008), statistical discourse parsers were developed (Lin et al., 2012; Ghosh et al., 2011; Xu et al., 2012).

PDTB adopts non-hierarchical binary view on discourse relations: Argument 1 (*Arg1*) and Argument 2 (*Arg2*), which is syntactically attached to a discourse connective. Thus, PDTB-based discourse parsing can be roughly partitioned into discourse relation detection, argument position classification, argument span extraction, and relation sense classification. For discourse relations signaled by a connective (explicit relations), discourse relation detection is cast as classification of connectives as discourse and non-discourse. Argument position classification involves detection of the location of *Arg1* with respect to *Arg2*: usually either the same sentence (SS) or previous ones (PS).[1] Argument span extraction, on the other hand, is extraction (labeling) of text segments that belong to each of the arguments. Finally, relation sense classification is the annotation of relations with the senses from PDTB.

Since arguments of explicit discourse relations can appear in the same sentence or in different ones (i.e. relations can be intra- or inter-sentential); there are two approaches to argument span extraction. In the first approach the parser decision is not conditioned on whether the relation is intra- or inter-sentential (e.g. (Ghosh et al., 2011)). In the second approach relations are parsed separately for each class (e.g. (Lin et al., 2012; Xu et al., 2012)). In the former approach argument span extraction is applied right after discourse connective detection, while the latter approach also requires argument position classification.

The decision on argument span can be made on different levels: from token-level to sentence-level. In (Ghosh et al., 2011) the decision is made on token-level, and the problem is cast as sequence labeling using conditional random fields (CRFs) (Lafferty et

[1]We use the term *inter-sentential* to refer to a set of relations that includes both previous sentence (*PS*) and following sentence (*FS*) *Arg1*. *Intra-sentential* and same sentence (*SS*) relations, on the other hand, are the same set.

al., 2001). In this paper we focus on argument span extraction, and extend the token-level sequence labeling approach of (Ghosh et al., 2011) with the separate models for arguments of intra-sentential and inter-sentential explicit discourse relations. To compare to the other approaches (i.e. (Lin et al., 2012) and (Xu et al., 2012)) we adopt the immediately previous sentence heuristic to select a candidate *Arg1* sentence for the inter-sentential relations. Additionally to the heuristic, we train and test CRF argument span extraction models to extract *exact* argument spans.

The paper is structured as follows. In Section 2 we briefly present the corpus that was used in the experiments – Penn Discourse Treebank. Section 3 describes related works. Section 4 defines the problem and assesses its complexity. In Section 5 we describe argument span extraction cast as the token-level sequence labeling; and in Section 6 we present the evaluation of the two approaches – either single or separate processing of intra- and inter-sentential relations. Section 7 provides concluding remarks.

2 The Penn Discourse Treebank

The Penn Discourse Treebank (PDTB) (Prasad et al., 2008) is a corpus that contains discourse relation annotation on top of WSJ corpus; and it is aligned with Penn Treebank (PTB) syntactic tree annotation. Discourse relations in PDTB are binary: *Arg1* and *Arg2*, where *Arg2* is an argument syntactically attached to a discourse connective. With respect to *Arg2*, *Arg1* can appear in the same sentence (SS case), one of the preceding (PS case) or following (FS case) sentences.

A discourse connective is a member of a well defined list of 100 connectives and a relation expressed via such connective is an *Explicit* relation. There are other types of discourse and non-discourse relations annotated in PDTB; however, they are out of the scope of this paper. Discourse relations are annotated using 3-level hierarchy of senses. The top level (level 1) senses are the most general: *Comparison*, *Contingency*, *Expansion*, and *Temporal* (Prasad et al., 2008).

3 Related Work

Pitler and Nenkova (2009) applied machine learning methods using lexical and syntactic features and achieved high classification performance on discourse connective detection task (F_1: 94.19%, 10 fold cross-validation on PDTB sections 02-22). Later, Lin et al. (2012) achieved an improvement with additional lexico-syntactic and path features (F_1: 95.76%).

After a discourse connective is identified as such, it is classified into relation senses annotated in PDTB. Pitler and Nenkova (2009) classify discourse connectives into 4 top level senses – *Comparison*, *Contingency*, *Expansion*, and *Temporal* – and achieve accuracy of 94.15%, which is slightly above the inter-annotator agreement. In this paper we focus on the parsing steps after discourse connective detection; thus, we use gold reference connectives and their senses as features.

The approaches used for the argument position classification even though useful, are incomplete as they do not make decision on argument spans. (Wellner and Pustejovsky, 2007) and (Elwell and Baldridge, 2008), following them, used machine learning methods to identify head words of the arguments of explicit relations expressed by discourse connectives. (Prasad et al., 2010), on the other hand, addressed a more difficult task of identification of sentences that contain *Arg1* for cases when arguments are located in different sentences.

Dinesh et al. (2005) and Lin et al. (2012) approach the problem of argument span extraction on syntactic tree node-level. In the former, it is a rule based system that covers limited set of connectives; whereas in the latter it is a machine learning approach with full PDTB coverage. Both apply syntactic tree subtraction to get argument spans. Xu et al. (2012) approach the problem on a constituent-level: authors first decide whether a constituent is a valid argument and then whether it is *Arg1*, *Arg2*, or neither. Ghosh et al. (2011) (and further (Ghosh et al., 2012a; Ghosh et al., 2012b)), on the other hand, cast the problem as token-level sequence labeling. In this paper we follows the approach of (Ghosh et al., 2011).

4 Problem Definition

In the introduction we mentioned Immediately Previous Sentence Heuristic for *Arg1* of inter-sentential explicit relations and Argument Position Classification as a prerequisite for processing intra- and inter-sentential relations separately. In this section we analyze PDTB to assess the complexity and potential accuracy of the heuristic and the classification task.

	SingFull		SingPart		MultFull		MultPart		Total	
ARG1										
IPS	**3,192**	(44.2%)	**1,880**	(26.0%)	370	(5.1%)	107	(1.5%)	5,549	(76.8%)
NAPS	993	(13.8%)	551	(7.6%)	71	(1.0%)	51	(0.7%)	1,666	(23.1%)
FS	2	(0.0%)	0	(0.0%)	1	(0.0%)	5	(0.0%)	8	(0.1%)
Total	4,187	(58.0%)	2,431	(33.7%)	442	(6.1%)	163	(2.3%)	7,223	(100%)
ARG2										
SS/Total	5,181	(71.7%)	1,936	(26.8%)	84	(1.2%)	22	(0.3%)	7,223	(100%)

Table 1: Distribution of *Arg1* with respect to the location (rows) and extent (columns) (partially copied from (Prasad et al., 2008)); and distribution of *Arg2* with respect to extent **in inter-sentential explicit discourse relations**.
SS = same sentence as the connective; IPS = immediately previous sentence; NAPS = non-adjacent previous sentence; FS = some sentence following the sentence containing the connective; SingFull = Single Full sentence; SingPart = Part of single sentence; MultFull = Multiple full sentences; MultPart = Parts of multiple sentences.

4.1 Immediately Previous Sentence Heuristic

According to Prasad et al. (2008)'s analysis of explicit discourse relations annotated in PDTB, out of 18,459 relations, 11,236 (60.9%) have both of the arguments in the same sentence (SS case), 7,215 (39.1%) have *Arg1* in the sentences preceding the *Arg2* (PS case), and only 8 instances have *Arg1* in the sentences following *Arg2* (FS case). Since FS case has too few instances it is usually ignored. For the PS case, the *Arg1* is located either in Immediately Previous Sentences (IPS: 30.1%) or in some Non-Adjacent Previous Sentences (NAPS: 9.0%).

CRF-based discourse parser of Ghosh et al. (2011), which processes SS and PS cases with the same model, uses ± 2 sentence window as a hypothesis space (5 sentences: 1 sentence containing the connective, 2 preceding and 2 following sentences). The window size is motivated by the observation that it entirely covers arguments of 94% of all explicit relations. The authors also report that the performance of the parser on inter-sentential relations (i.e. mainly PS case) has F-measure of 36.0. However, since in 44.2% of inter-sentential explicit discourse relations *Arg1* fully covers the sentence immediately preceding *Arg2* (see Table 1 partially copied from (Prasad et al., 2008)), the heuristic that selects the immediately previous sentence and tags all of its tokens as *Arg1* already yields F-measure of 44.2 over all PDTB (the performance on the test set may vary).

The same heuristic is mentioned in (Lin et al., 2012) and (Xu et al., 2012) as a majority classifier for the relations with *Arg1* in previous sentences.

Compared to the ± 2 window, the heuristic covers *Arg1* of only 88.4% explicit discourse relations (60.9% SS + 27.5% PS); since it ignores all the relations with *Arg1* in Non-Adjacent Previous Sentences (NAPS) (9.0% of all explicit relations), and does not accommodate *Arg1* spanning multiple immediately preceding sentences (2.6% of all explicit relations). Nevertheless, 70.2% of all PS explicit relations have *Arg1* entirely inside the immediately previous sentence. Thus, the integration of the heuristic is expected to improve the argument span extraction performance for inter-sentential *Arg1*.

In 98.5% of all PS cases *Arg2* is within the sentence containing the connective (remaining 1.5% are multi-sentence *Arg2*); and in 71.7% of all PS cases it fully covers the sentence containing the discourse connective (see Table 1). Thus, similar heuristic for *Arg2* is to tag all the tokens of the sentence except the connective as *Arg2*.

For the heuristics to be applicable, a discourse connective has to be classified as requiring its *Arg1* in the same sentence (SS) or the previous ones (PS), i.e. it requires argument position classification.

4.2 Argument Position Classification

Explicit discourse connectives, annotated in PDTB, belong to one of the three syntactic categories: (1) subordinating conjunctions (e.g. *when*), (2) coordinating conjunctions (e.g. *and*), and (3) discourse adverbials (e.g. *for example*). With few exceptions, a discourse connective belongs to a single syntactic category (see Appendix A in (Knott, 1996)). Each of these syntactic categories has a strong preference on the po-

	Sentence Initial				Sentence Medial			
	SS		PS		SS		PS	
Coordinating	10	(0.05%)	2,869	(15.54%)	3,841	(20.81%)	202	(1.09%)
Subordinating	1,402	(7.60%)	114	(0.62%)	5,465	(29.61%)	83	(0.45%)
Discourse Adverbial	13	(0.07%)	1,632	(8.84%)	495	(2.68%)	2,325	(12.60%)

Table 2: Distribution of discourse connectives in PDTB with respect to syntactic category (rows) and position in the sentence (columns) and the location of *Arg1* as in the same sentence (SS) as the connective or the previous sentences (PS). The case when *Arg1* appears in some following sentence (FS) is ignored, since it has only 8 instances.

sition of *Arg1*, depending on whether the connective appears sentence-initially or sentence-medially. Here, a connective is considered sentence-initial if it appears as the first sequence of words in a sentence. Table 2 presents the distribution of discourse connectives in PDTB with respect to the syntactic categories, their position in the sentence, and having *Arg1* in the same or previous sentences. The distribution of sentence-medial discourse adverbials, which is the most ambiguous class, between SS and PS cases is 17.5% to 82.5%; for all other classes it higher than 90% to 10%. Thus, the overall accuracy of the SS vs. PS majority classification using just syntactic category and position information is already 95.0%.

When analyzed on per connective basis, the observation is that some connectives require *Arg1* in the same or previous sentence irrespective of their position in the sentence. For instance, sentence-initial subordinating conjunction *so* always has its *Arg1* in the previous sentence; and the parallel sentence-initial subordinating conjunction *if..then* in the same sentence. Others, such as sentence-medial adverbials *however* and *meanwhile* mainly require their *Arg1* in the previous sentence. Even though low, there is still an ambiguity: e.g. for sentence-medial adverbials *also*, *therefore*, *still*, *instead*, *in fact*, etc. *Arg1* appears in SS and PS cases evenly. Consequently, assigning the position of the *Arg1* considering the discourse connective, together with its syntactic category and its position in the sentence, for PDTB will be correct in more than 95% of instances.

In the literature, the task of argument position classification was addressed by several researchers (e.g. (Prasad et al., 2010), (Lin et al., 2012)). Lin et al. (2012), for instance, report F_1 of 97.94% for a classifier trained on PDTB sections 02-21, and tested on section 23. The task has a very high baseline and even higher performance on supervised machine learning,

Feature	**ABBR**	**Arg2**	**Arg1**
Token	TOK	Y	Y
POS-Tag	POS		
Lemma	LEM	Y	Y
Inflection	INFL	Y	Y
IOB-Chain	IOB	Y	Y
Connective Sense	CONN	Y	Y
Boolean Main Verb	BMV		Y
Prev. Sent. Festure	PREV		Y
Arg2 Label	ARG2		Y

Table 3: Feature sets for *Arg2* and *Arg1* argument span extraction in (Ghosh et al., 2011)

which is an additional motivation to process intra- and inter-sentential relations separately.

5 Parsing Models

We replicate and evaluate the discourse parser of (Ghosh et al., 2011), then modify it to process intra- and inter-sentential explicit relations separately. This is achieved by integrating Argument Position Classification and Immediately Previous Sentence heuristic into the parsing pipe-line.

Since the features used to train argument span extraction models for both approaches are the same, we first describe them in Subsection 5.1. Then we proceed with the description of the single model discourse parser (our baseline) and separate models discourse parser, Subsections 5.2 and 5.3, respectively.

5.1 Features

The features used to train the models for *Arg1* and *Arg2* are given in Table 3. Besides the token itself (*TOK*), the rest of the features is described below.

Lemma (*LEM*) and *inflectional* affixes (*INFL*) are extracted using *morpha* tool (Minnen et al., 2001), that requires token and its POS-tag as input. For instance, for the word *flashed* the lemma and infection features are '*flash*' and '*+ed*', respectively.

IOB-Chain (*IOB*) is the path string of the syntactic tree nodes from the root node to the token, prefixed with the information whether a token is at the beginning (B-) or inside (I-) the constituent. The feature is extracted using the *chunklink* tool (Buchholz, 2000). For example, the IOB-Chain '*I-S/B-VP*' indicates that a token is the first word of the verb phrase (*B-VP*) of the main clause (*I-S*).

PDTB Level 1 Connective sense (*CONN*) is the most general sense of a connective in PDTB sense hierarchy: one of *Comparison*, *Contingency*, *Expansion*, or *Temporal*. For instance, a discourse connective *when* might have the CONN feature '*Temporal*' or '*Contingency*' depending on the discourse relation it appears in, or '*NULL*' in case of non-discourse usage. The value of the feature is '*NULL*' for all tokens except the discourse connective.

Boolean Main Verb (*BMV*) is a feature that indicates whether a token is a main verb of a sentence or not (Yamada and Matsumoto, 2003). For instance in the sentence *Prices collapsed when the news flashed*, the main verb is *collapsed*; thus, its BMV feature is '1', whereas for the rest of tokens it is '0'.

Previous Sentence Feature (*PREV*) signals if a sentence immediately precedes the sentence starting with a connective, and its value is the first token of the connective (Ghosh et al., 2011). For instance, if some sentence *A* is followed by a sentence *B* starting with discourse connective *On the other hand*, all the tokens of the sentence *A* have the *PREV* feature value '*On*'. The feature is similar to a heuristic to select the sentence immediately preceding a sentence starting with a connective as a candidate for *Arg1*.

Arg2 Label (*ARG2*) is an output of *Arg2* span extraction model, and it is used as a feature for *Arg1* span extraction. Since for sequence labeling we use IOBE (Inside, Out, Begin, End) notation, the possible values of *ARG2* are IOBE-tagged labels, i.e. '*ARG2-B*' – if a word is the first word of *Arg2*, '*ARG2-I*' – if a word is inside the argument span, '*ARG2-E*' – if a word is in the last word of *Arg2*, and '*O*' otherwise.

CRF++[2] – conditional random field implementation we use – allows definition of feature templates. Via templates these features are enriched with n-grams: tokens with 2-grams in the window of ±1 tokens, and the rest of the features with 2 & 3-grams in the window of ±2 tokens.

[2]https://code.google.com/p/crfpp/

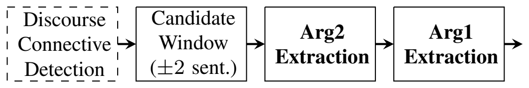


Figure 1: Single model discourse parser architecture of (Ghosh et al., 2011). CRF argument span extraction models are in bold.

For instance, labeling a token as *Arg2* is an assignment of one of the four possible labels: ARG2-B, ARG2-I, ARG2-E and O (ARG2 with IOBE notation). The feature set (token, lemma, inflection, IOB-chain and connective sense (see Table 3)) is expanded by CRF++ via template into 55 features ($5 * 5$ unigrams, 2 token bigrams, $4 * 4$ bigrams and $4 * 3$ trigrams of other features).

5.2 Single Model Discourse Parser

The discourse parser of (Ghosh et al., 2011) is a cascade of CRF models to sequentially label *Arg2* and *Arg1* spans (since *Arg2* label is a feature for *Arg1* model) (see Figure 1). There is no distinction between intra- and inter-sentential relations, rather the single model jointly decides on the position and the span of an argument (either *Arg1* or *Arg2*, not both together) in the window of ±2 sentences (the parser will be further abbreviated as *W5P* – Window 5 Parser).

The single model parser achieves F-measure of 81.7 for *Arg2* and 60.3 for *Arg1* using CONNL evaluation script. The performance is higher than (Ghosh et al., 2011) – *Arg2*: F_1 of 79.1 and *Arg1*: F_1 of 57.3 – due to improvements in feature and instance extraction, such as the treatment of multi-word connectives. These models are the baseline for comparison with separate models architecture. However, we change the evaluation method (see Section 6).

5.3 Separate Models Discourse Parser

Figure 2 depicts the architecture of the discourse parser processing intra- and inter-sentential relations separately. It is a combination of argument position classification with specific CRF models for each of the arguments of SS and PS cases, i.e. there are 4 CRF models – SS *Arg1* and *Arg2*, and PS *Arg1* and *Arg2* (following sentence case (FS) is ignored). SS models are applied in a cascade and, similar to the

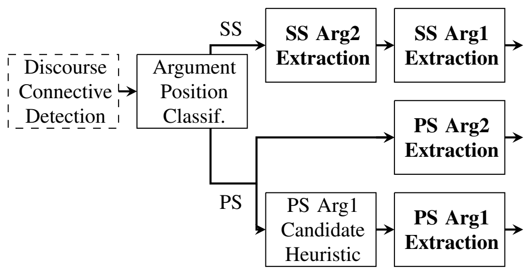

SS
Discourse Connective Detection
Argument Position Classif.
SS Arg2 Extraction
SS Arg1 Extraction
PS Arg2 Extraction
PS
PS Arg1 Candidate Heuristic
PS Arg1 Extraction

Figure 2: Separate models discourse parsing architecture. CRF argument span extraction models are in bold.

baseline single model parser, *Arg2* label is a feature for *Arg1* span extraction. These SS models are trained using exactly the same features, with the exception of *PREV* feature: since we consider only the sentence containing the connective, it naturally falls out.

For the PS case, we apply a heuristic to select candidate sentences. Based on the observation that in PDTB for the PS case *Arg2* span is fully located in the sentence containing the connective in 98.5% of instances; and *Arg1* span is fully located in the sentence immediately preceding *Arg2* in 71.7% of instances; we select sentences in these positions to train and test respective CRF models. The feature set for *Arg2* remains the same, whereas, from *Arg1* feature set we remove *PREV* and *Arg2* label (since in PS case *Arg2* is in different sentence, the feature will always have the same value of 'O').

For *Argument Position Classification* we train unigram BoosTexter (Schapire and Singer, 2000) model with 100 iterations[3] on PDTB sections 02-22 and test on sections 23-24; and, similar to other researchers, achieve high results: $F_1 = 98.12$. The features are connective surface string, POS-tags, and IOB-chains. The results obtained using automatic features ($F_1 = 97.87$) are insignificantly lower (McNemar's $\chi^2(1, 1595) = 0.75, p = 0.05$); thus, this step will not cause deterioration in performance with automatic features. Here we used Stanford Parser (Klein and Manning, 2003) to obtain POS-tags and automatic constituency-based parse trees.

Since both argument span extraction approaches are equally affected by the discourse connective detection step, we use gold reference connectives. As an alternative, discourse connectives can be detected with high accuracy using addDiscourse tool (Pitler and Nenkova, 2009).

[3]The choice is based on the number of discourse connectives defined in PDTB.

In the separate models discourse parser, the steps of the process to extract argument spans given a discourse connective are as follows:

1. Classify connective as SS or PS;
2. If classified as SS:
   (a) Use SS Arg2 CRF model to label the sentence tokens for *Arg2*;
   (b) Use SS Arg1 CRF model to label the sentence tokens for *Arg1* using *Arg2* label as a feature;
3. If classified as PS
   (a) Select the sentence containing the connective and use PS Arg2 CRF model to label *Arg2* span;
   (b) Select the sentence immediately preceding the *Arg2* sentence and use PS Arg1 CRF model to label *Arg1* span.

The separate model parser with CRF models will be further abbreviated as *SMP*; and with the heuristics for PS case as *hSMP*.

6 Experiments and Results

We first describe the evaluation methodology. Then present evaluation of PS case CRF models against the heuristic. In subsection 6.3 we compare the performance of the single and separate model parsers on SS and PS cases of the test set separately and together. Finally, we compare the results of the separate model parser to (Lin et al., 2012) and (Xu et al., 2012).

6.1 Evaluation

There are two important aspects regarding the evaluation. First, in this paper it is different from (Ghosh et al., 2011); thus, we first describe it and evaluate the difference. Second, in order to compare the baseline single and separate model parsers, the error from argument position classification has to be propagated for the latter one; and the process is described in 6.1.2.

Since both versions of the parser are affected by automatic features, the evaluation is on gold features only. The exception is for *Arg2* label; since it is generated within the segment of the pipeline we are interested in. Unless stated otherwise, all the results for *Arg1* are reported for automatic *Arg2* labels as a feature. Following (Ghosh et al., 2011) PDTB is split as Sections 02-22 for training, 00-01 for development, and 23-24 for testing.

6.1.1 CONLL vs. String-based Evaluation

Ghosh et al. (2011) report using CONLL-based evaluation script. However, it is not well suited for the evaluation of argument spans because the unit of evaluation is a chunk – a segment delimited by any out-of-chunk token or a sentence boundary. However, in PDTB arguments can (1) span over several sentences, (2) be non-contiguous in the same sentence. Thus, CONLL-based evaluation yields incorrect number of test instances: Ghosh et al. (2011) report 1,028 SS and 617 PS test instances for PDTB sections 23-24 (see caption of Table 7 in the original paper), which is 1,645 in total; whereas there is only 1,595 explicit relations in these sections.

In this paper, the evaluation is string-based; i.e. an argument span is correct, if it matches the whole reference string. Following (Ghosh et al., 2011) and (Lin et al., 2012), argument initial and final punctuation marks are removed; and precision (p), recall (r) and F_1 score are computed using the equations 1 – 3.

$$p = \frac{\text{Exact Match}}{\text{Exact Match} + \text{No Match}} \quad (1)$$

$$r = \frac{\text{Exact Match}}{\text{References in Gold}} \quad (2)$$

$$F_1 = \frac{2 * p * r}{p + r} \quad (3)$$

In the equations, *Exact Match* is the count of correctly tagged argument spans; *No Match* is the count of argument spans that do not match the reference string exactly (even one token difference is counted as an error); and *References in Gold* is the total number of arguments in the reference.

String-based evaluation of the single model discourse parser with gold features reduces F_1 for *Arg2* from 81.7 to 77.8 and for *Arg1* from 60.33 to 55.33.

6.1.2 Error Propagation

Since the single model parser applies argument span extraction right after discourse connective detection, whereas in the separate model parser there is an additional step of argument position classification; for the two to be comparable an error from the argument position classification is propagated. Even though, the performance of the classifier is very high (98.12%) there are still some misclassified instances. These instances are propagated to the counts of *Exact Match* and *No Match* of the argument span extraction. For example, if the argument position classifier misclassified an SS connective as PS; in the SS evaluation its *Arg1* and *Arg2* are considered as not recalled regardless of argument span extractor's decision (i.e. neither *Exact Match* nor *No Match*); and in the PS evaluation, they are both considered as *No Match*.

The separate model discourse parser results are reported without error propagation for in-class comparison of the heuristic and CRF models, and with error propagation for cross-class comparison with the single model parser.

	Arg2			**Arg1**		
	P	**R**	**F1**	**P**	**R**	**F1**
hSMP	74.19	74.19	74.19	39.19	39.19	39.19
SMP	78.61	78.23	78.42	46.81	37.90	41.89

Table 4: Argument span extraction performance of the heuristics (*hSMP*) and the CRF models (*SMP*) on inter-sentential relations (PS case). Results are reported as precision (**P**), recall (**R**) and F-measure (**F1**)

6.2 Heuristic vs. CRF Models

The goal of this section is to assess the benefit of training CRF models for the extraction of exact argument spans of PS *Arg1* and *Arg2* on top of the heuristics. The performance of the heuristics (immediately previous sentence for *Arg1* and the full sentence except the connective for *Arg2*) and the CRF models is reported in Table 4. CRF models perform significantly better for *Arg2* (McNemar's $\chi^2(1, 620) = 7.48$, $p = 0.05$). Even though, they perform 2.7% better for *Arg1*, the difference is insignificant (McNemar's $\chi^2(1, 620) = 0.66$, $p = 0.05$). For both arguments, the CRF model results are lower than expected.

6.3 Single vs. Separate Models

To compare the single and the separate model parsers, the results of the former must be split into SS and PS cases. For the latter, on the other hand, we propagate

	Arg2			**Arg1**		
	P	**R**	**F1**	**P**	**R**	**F1**
W5P	87.57	84.51	86.01	71.73	62.97	67.07
SMP	90.36	87.49	88.90	70.27	66.67	68.42

Table 5: Performance of the single ±2 window (*W5P*) and separate model (*SMP*) parsers on argument span extraction of SS relations; reported as precision (**P**), recall (**R**) and F-measure (**F1**). For the *SMP* results are with error propagation from argument position classification.

	Arg2			**Arg1**		
	P	**R**	**F1**	**P**	**R**	**F1**
W5P	71.12	59.19	64.61	40.06	22.74	29.01
hSMP	74.67	72.23	73.94	38.98	38.23	38.60
SMP	79.01	77.10	78.04	46.23	36.61	40.86

Table 6: Performance of the single model parser (*W5P*) and the separate model parser with the heuristics (*hSMP*) and CRF models (*SMP*) on argument span extraction of PS relations; reported as precision (**P**), recall (**R**) and F-measure (**F1**). For the separate model parsers, results include error propagation from argument position classification.

error from the argument position classification step. For the PS case we also report the performance of the heuristic with error propagation.

Table 5 reports the results for the SS case, and Table 6 reports the results for the PS case. In both cases the separate model parser with error propagation from argument position classification step significantly outperforms the single model parser.

The performance of the separate model parsers (reported in Table 7) with heuristics and CRF models on all relations (SS + PS) both are significantly better than the performance of single ±2 window model parser (for *SMP* McNemar's $\chi^2(1, 1595) = 17.75$ for *Arg2* and $\chi^2(1, 1595) = 19.82$ for *Arg1*, $p = 0.05$).

	Arg2			**Arg1**		
	P	**R**	**F1**	**P**	**R**	**F1**
W5P	81.47	74.42	77.79	61.90	46.96	53.40
hSMP	84.21	81.94	83.06	57.86	55.61	56.71
SMP	85.93	83.45	84.67	61.94	54.98	58.25

Table 7: Performance of the single model parser (*W5P*) and the separate model parser with the heuristics (*hSMP*) and CRF models (*SMP*) on argument span extraction of all relations; reported as precision (**P**), recall (**R**) and F-measure (**F1**). For the separate model parsers, results include error propagation from argument position classification.

	Arg2	**Arg1**
Lin et al. (2012)	82.23	59.15
Xu et al. (2012)	81.00	**60.69**
hSMP	80.04	54.37
SMP	**82.35**	57.26

Table 8: Comparison of the separate model parsers (with heuristics (*hSMP*) and CRFs (*SMP*)) to (Lin et al., 2012) and (Xu et al., 2012) reported as F-measure (**F1**). Trained on PDTB sections 02-21, tested on 23.

6.4 Comparison of Separate Model Parser to (Lin et al., 2012) and (Xu et al., 2012)

The separate model parser allows to compare argument span extraction cast as token-level sequence labeling to the syntactic tree-node level classification approach of (Lin et al., 2012) and constituent-level classification approach of (Xu et al., 2012); since now the complexity and the hypothesis spaces are equal. For this purpose we train models on sections 02-21 and test on 23.

Unfortunately, the authors do not report the results on SS and PS cases separately, but only the combined results that include the heuristic. Moreover, the performance of the heuristic is mentioned to be 76.9% instead of 44.2% for the exact match (see IPS x SingFull cell in Table 1 or Table 1 in (Prasad et al., 2008)). Thus, the comparison provided here is not definite. Since all systems have different components up the pipe-line, the only possible comparison is without error propagation.

From the results in Table 8, we can observe that all the systems perform well on *Arg2*. As expected, for the harder case of *Arg1*, performances are lower.

7 Conclusion

In this paper we compare two strategies for the argument span extraction: to process intra- and inter-sentential explicit relations by a single model, or separate ones. We extend the approach of (Ghosh et al., 2011) to argument span extraction cast as token-level sequence labeling using CRFs and integrate argument position classification and immediately previous sentence heuristic. The evaluation of parsing strategies on the PDTB explicit discourse relations shows that the models trained specifically for intra- and inter-sentential relations significantly outperform the single ±2 window models.

References

Sabine Buchholz. 2000. Readme for perl script chunklink.pl.

Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Rashmi Prasad, Aravind Joshi, and Bonnie Webber. 2005. Attribution and the (non)-alignment of syntactic and discourse arguments of connectives. In *Proceedings of the ACL Workshop on Frontiers in Corpus Annotation II: Pie in the Sky*.

Robert Elwell and Jason Baldridge. 2008. Discourse connective argument identification with connective specific rankers. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2008)*.

Sucheta Ghosh, Richard Johansson, Giuseppe Riccardi, and Sara Tonelli. 2011. Shallow discourse parsing with conditional random fields. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP 2011)*.

Sucheta Ghosh, Richard Johansson, Giuseppe Riccardi, and Sara Tonelli. 2012a. Improving the recall of a discourse parser by constraint-based postprocessing. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*.

Sucheta Ghosh, Giuseppe Riccardi, and Richard Johansson. 2012b. Global features for shallow discourse parsing. In *Proceedings of the SIGDIAL 2012 Conference, The 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*.

Dan Klein and Christopher D. Manning. 2003. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*.

Alistair Knott. 1996. *A Data-Driven Methodology for Motivating a Set of Coherence Relations*. Ph.D. thesis, University of Edinburgh.

John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of 18th International Conference on Machine Learning*.

Ziheng Lin, Hwee Tou Ng, and Min-Yen Kan. 2012. A pdtb-styled end-to-end discourse parser. *Natural Language Engineering*, 1:1 – 35.

Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of english. *Natural Language Engineering*.

Emily Pitler and Ani Nenkova. 2009. Using syntax to disambiguate explicit discourse connectives in text. In *Proceedings of the ACL-IJCNLP Conference*.

Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind Joshi, and Bonnie Webber. 2008. The penn discourse treebank 2.0. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*.

Rashmi Prasad, Aravind Joshi, and Bonnie Webber. 2010. Exploiting scope for shallow discourse parsing. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC-2010)*.

Robert E. Schapire and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

Maite Taboada and William C. Mann. 2006. Applications of rhetorical structure theory. *Discourse Studies*, (8):567–88.

Bonnie L. Webber, Markus Egg, and Valia Kordoni. 2011. Discourse structure and language technology. *Natural Language Engineering*, pages 1 – 54.

Ben Wellner and James Pustejovsky. 2007. Automatically identifying the arguments of discourse connectives. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*.

Fan Xu, Qiao Ming Zhu, and Guo Dong Zhou. 2012. A unified framework for discourse argument identification via shallow semantic parsing. In *Proceedings of 24th International Conference on Computational Linguistics (COLING 2012): Posters*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of 8th International Workshop on Parsing Technologies*.

Improved Chinese Parsing Using Named Entity Cue

Dongchen Li, Xiantao Zhang and Xihong Wu
Key Laboratory of Machine Perception and Intelligence,
Speech and Hearing Research Center,
Peking University, China
{lidc,Zhangxt,wxh}@cis.pku.edu.cn

Abstract

Parsing and named entity recognition are two standalone techniques in natural language processing community. We expect that these two types of annotations should provide useful information to each other, and that modeling them jointly should improve performance and produce consistent outputs. Employing more fine-grained named entity annotations helps to parse complex named entity structures correctly. Thus, we integrate parsing and named entity recognition in a unified framework: 1. Through a joint representation of syntactic and named entity structures, we annotate named entity information to Penn Chinese Treebank5.0 (CTB5.0); 2. We annotate the nested structures for all nested named entities; 3. A latent annotation probabilistic context-free grammar (PCFGLA) model is trained on the data with joint representation. Experiment results demonstrate the mutual benefits for both Chinese parsing and named entities recognition tasks.

1 Why Exploit Named Entity Cue for Chinese Parsing?

Chinese parsing and named entity recognition are two basic Chinese NLP technologies. They play an important role in the Chinese information extraction, machine translation and question answering systems.

However, to the best of our knowledge, previous researches generally regard them as two standalone processes. One of the reasons is that the Treebank for training a parser has not been annotated with adequate named entity information. We argue that it will be beneficial to utilize named entity cue in parsing. Because one of the main difficulties in parsing Chinese is bracketing phrases with complex structure, and many complex phrases are named entities.

In Chinese there are a large number of named entities. Named entities (NEs) can be generally divided into three types: entity names, temporal expressions, and number expressions. They are “unique identifiers” of entities (organizations, persons, locations), time (date, times), and quantities (monetary values, percentages). According to Chinese Treebank fifth edition (CTB5.0) (Xue et al., 2002), every sentence contains over 1.5 entity names. Table 1 shows the distribution of these named entities in CTB5.0.

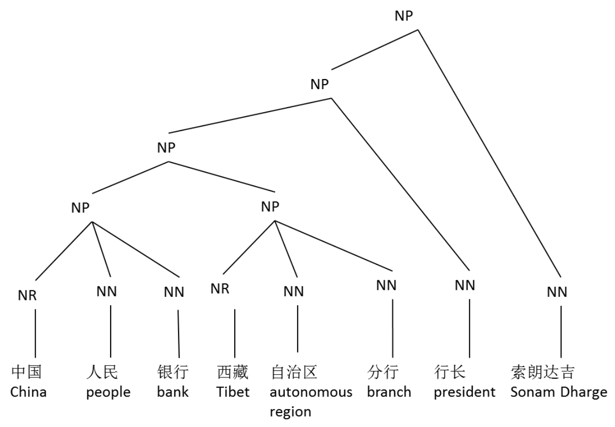


Figure 1: A named entity example with complex structures.

Different types of named entity phrases have their distinct structure patterns. However, all noun phrases including named entities get an identical label, say, noun phrase (NP). Computational processing of Chinese is typically based on the coarse syntactic tags. For example, in Figure 1, the structures of 中国人民银行“People’s Bank of China” and 中国人民银行西藏自治区分行行长索朗达吉“Sonam Dharge, the president of the Tibet Autonomous Region branch of

NE-Types	SubTypes	Description	Percent	Example
Entity Names	GPE	geographical / social / political entities	44.95	上海“Shanghai”, 山东省“Shandong Province”
	PERSON	person	29.34	理查德・尼克松 “Richard Nixon”
	ORG	organization	21.62	深圳市教育局 “Shenzhen Education Bureau”
	LOC	location(non-GPE locations)	8.67	淮河“Huai River”
Temporal expressions	DATE	date	14.74	一九九九年“the year of 1999”
	TIME	time	0.44	12时“12:00”
Number Expressions	NUM	number	52.58	172, 1.5
	ORD	ordinal number	6.78	第一“first”
	FRACTION	traction	4.06	百分之七十“70%”
	CODE	code	2.28	AK-47
	EVENT	event	0.83	五四“May Fourth Movement”
	TEMPERATURE	temperature	0.10	十二度“12 ℃”
	RATIO	ratio, score	0.08	0:05
	TEL	telephone number	0.05	23482192
	MONEY	money	0.02	三百万元“three million Yuan”

Table 1: The distribution of the named entity types (#sentences = 18789)

the People's Bank of China" is quite different, but they get the identical label NP in CTB. A parser trained on these annotations is messy and hard to discriminate these complex structures correctly. Much work has illustrated that training the parser with manually annotated fine-grained labels and structures could help disambiguate parsing structure and improve parsing accuracy (Li, 2011; Li and Wu, 2012).

Thus, it is necessary to introduce these named entities in syntactic structure and integrate their recognition in the parsing process.

We integrate syntactic and named entity information in a unified framework through a joint representation. We add these named entity annotations into the syntactic structures in CTB5.0, with special care for nested named entity. Then we validate our annotations in parsing and named entity recognition tasks. This joint representation improves Chinese parsing accuracy significantly. Furthermore, the accuracies of the named entity recognition of our joint model outperform CRF-based NER system.

The rest of this paper is organized as follows. Section 2.1 reviews previously established Chinese Treebank (Penn Chinese Treebank) and Chinese corpus annotated with named entities (OntoNotes). Section 3 represents our joint representation of syntactic structures and named entities. In section 4 we perform experiments to illustrate the effectiveness of our joint representation.

2 Related Work

Penn Chinese Treebank (CTB) is the most widely used treebank for parsing Chinese. OntoNotes is a corpus annotated with both syntactic structure and named entities. We first review the annotations in these two corpora. Then, a brief introduction of Chinese parsing on character-level is given. Finally, we reviews the previous work on utilizing named entity cue in parsing.

Length of word	#NEs	#All	Percent
1	10276	166881	6.16
2	21843	222539	9.82
3	13588	30436	44.64
4	2532	6287	40.27
5	2300	2454	93.72
6	704	772	91.19
7	283	325	87.08
8	283	307	92.18
9	83	103	80.58
10	32	38	84.21
11	14	16	87.5
12	2	4	50
13	5	6	83.33

Table 2: Statistics of NEs' percent in different words' length

2.1 Penn Chinese Treebank and OntoNotes

CTB is a segmented, part-of-speech tagged, and fully bracketed corpus that currently has 500 thousand words (over 824K Chinese characters). There are totally 890 files in CTB5.0.

Parsing of Chinese is typically based on coarse part-of-speech tags and syntactic tags in CTB. In CTB, named entity phrase is simply labeled as a noun phrase (NP) without distinction of their diverse types (some of them may be labeled with an extra function tag P-N). Similarly, named entity words are simply labeled as a proper noun (NR), cardinal number (CD), ordinal number (OD) or temporal noun (NT), and they correspond to words in the parse trees without annotation of their internal word structure.

OntoNotes Release 4.0 (LDC2011T03) is a large, manually annotated corpus that contains various text genres and annotations (Hovy et al., 2006). It is also a corpus with annotation of entity names in Chinese. It contains 403 files which are also in CTB5.0, including the test set and development set in the standard parsing evaluation setup. Entity names in OntoNotes4.0 are annotated with 18 types of entity names, including PERSON, ORGANIZATION, GPE, LOC, PRODUCT and so on.

Many named entities contain other named entities inside them. However, works on named entity recognition (NER) and the annotation of OntoNotes have almost entirely ignored nested entities and instead chosen to focus on the outermost entities.

2.2 Parsing

Most high-performance parsers is based on probabilistic context-free grammars (PCFGs). They all refine grammar labels to capture more syntactic characteristic, ranging from full lexicalization and intricate smoothing (Collins, 1999; Charniak, 2000) to category refinement (Johnson, 1998; Klein and Manning, 2003). Latent annotation probabilistic context-free grammar (PCFG-LA) method in Matsuzaki et al. (2005) and Petrov and Klein (2007) automatically refines syntactic and lexical tags in an unsupervised manner, and has achieved state-of-the-art performance on both English and Chinese.

In recent years, there has been much work on character-level Chinese parsing. Qian and Liu (2012) trained three individual models of Chinese segmentation, POS tagging and Parsing separately during training, and incorporated them together in a discriminative framework. Zhang et al. (2013) integrated character-structure features in the joint model based on the discriminative shift-reduce parser of Zhang and Clark (2009) and Zhang and Clark (2011)Zhang and Clark (2009; 2011).

In spite of the convenience of its totally automatic learning process, the main defect of the latent factor models lies in that the training process is completely data-driven and suffers from data sparseness. To alleviate this problem, we leverage named entity cue, in the form of explicit annotation.

2.3 Named Entity Cue in Parsing

There is a large body of work on parsing and named entity recognition (Bikel and Chiang, 2000; Sekine and Nobata, 2004; Klementiev and Roth, 2006; Singh et al., 2010) separately. The sequence labeling approach has been shown to perform well on the task of Chinese NER (Chen et al., 2006; Yu et al., 2008). Finkel and Manning (2009a) and Finkel and Manning (2009b) paid special attention to the entity names in paring English. They gave a joint NER and parsing model with a discriminative parser, and improved accuracy for both tasks. We take advantage of named entity cue in character-level Chinese parsing, and further exploiting nested named entities in parsing.

Some existing work investigates the number expressions in parsing. Harper and Huang (2009) addressed this issue for achieving better parsing performance. Our work is not to verbalize sequences of digits; we annotate the entire constituent with fine label, such as DATE, NUM, TIME, FRACTION.

3 Our Approach

However, the completely data-driven state-split approach is prone to overfit the training data. Because the training data is always extremely sparse, and the automatically split categories might not be adequate. To improve parsing accuracy, Li (2011) manually annotated the internal structure of words, /citeli2012conjuncting manually annotated fine-grained labels for function words.

In our approach, all these types of named entity information are annotated to CTB5.0 through a joint representation in both word-level and character-level. Then we train a PCFG-LA parser on the corpus, and validate that named entity cue helps to improve parsing and NER accuracy simultaneously.

3.1 Named Entity Representation in Syntactic Tree

We argue that syntactic information and named entity information are mutual beneficial, so we enrich the annotations of the parse tree with fine-grained named entity labels to achieve the joint representation.

It is an important issue of how to define the types of Named entities. OntoNotes Release 4.0 (LDC2011T03) has annotated eighteen types of entity names. Some of these entity types do not occur frequently and are not always useful in practice, such as `works of art`, `product` and `law`, so we discard them in this study. In addition, we annotate the types of code, ratio and tel. All the named entity types are explained in Table 1.

There are totally 890 files in CTB5.0, and 403 of them have already been annotated with entity names in OntoNotes4.0. The test set and development set are setup as in the standard parsing evaluation. We annotated the left 487 files with previously mentioned types of named entities following the guideline of OntoNotes4.0.

3.2 Nested Named Entities Annotations

One of the main challenges for named entity recognition task is dealing with nested named entities. For example, Figure 1 contains nested named entities 中国人民银行西藏自治区分行“the Tibet Autonomous Region branch of the People’s Bank of China”, 中国人民银行“the People’s Bank of China”, and 索朗达吉“Sonam Dharge”. Tradition sequence labeling methods, such as CRF, treat the text as a linear sequence and have great difficulty in handling nested named entities, if not impossible.

We adopt a novel solution to explicitly represent nested named entities naturally in the syntactic structure. Nested named entities are exhaustively labelled in the syntactic tree structure, and each corresponds to one node in the tree.

Next, we will discuss the annotation process in detail. We refine the label of named entities’ components. As shown in Figure 1, 中国人民银行西藏自治区分行“People’s Bank of China branch of the Tibet Autonomous Region” is labeled as “NP_ORG”, and its two children in the tree are also labeled as “NP_ORG”. All the words’ structures are not changed; we just add a finer label to replace the original coarse label.

Further, we annotate the internal structure of a word that represents a nested named entity. There are three types of nested named entities: GPE, PERSON and temporal expression. We handle them respectively as follows.

For GPE, we split the GPE name and its geographical unit apart in a tree structure. This annotation style has the advantage of generalizing the common GPE composition structure. For example, 深圳市教育局“Shenzhen Education Bureau” is a ORG, but 深圳“Shenzhen” and 深圳市“Shenzhen city” are both GPE. The character 市“city” will obtain a special label. [1] In this case, we get a derivation which includes GPE $\rightarrow$ GPE GPEend. The experiment results in the next section show that the parser benefits a lot from this derivation. This example is shown in Figure 2.

We also distinguish the Chinese and foreign name by the entity name labels NR_PERSONF (Foreign

[1]When annotating the internal word structure, We do not need to distinguish an original word (e.g., Shenzhen City: NR_GPE) from an internal sub-word (e.g., Shenzhen: NR_GPE) explicitly. Because the internal sub-word can always be located by the geographical unit which is tagged by ”end”.

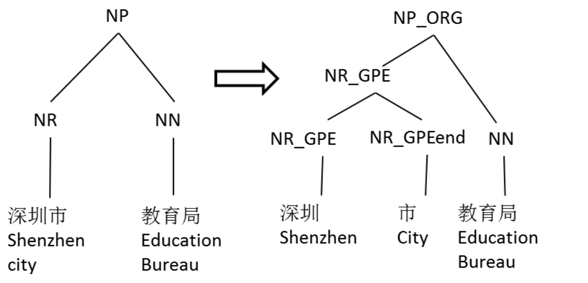


Figure 2: An example annotation for the phrase 深圳市教育局"Shenzhen Education Bureau"

Person Name) and NR_PERSONC (Chinese Person Name). It is obvious that a name containing the character '・' is a foreign name. Using this cue, it is easy to recognize the foreign names. See Figure 3 for an illustration.

For temporal expressions, the nested structure is bracketed into number expressions and temporal unit. For instance, the word 十五日"the 15th day in a month" will be split with 十五-NUM and 日-Day. Figure 4 gives a detailed example.

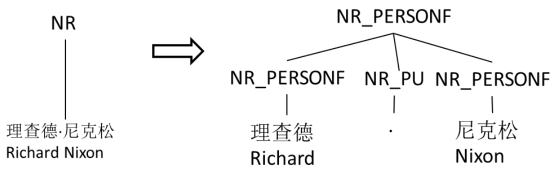


Figure 3: An example of annotation for the foreign name 理查德・尼克松"Richard Nixon"

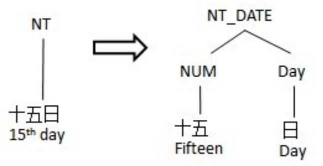


Figure 4: An example of nested annotation for the temporal expression 十五日"the 15th day in a month"

3.3 Our Annotation Method

The process of annotating named entity labels is as follows: Firstly, sentences also in OntoNotes (with file number from 1 to 325 and 1001 to 1078) will be selected, resulting in a small treebank with named entity annotations. A PCFG-LA parser is trained on the small treebank. Then the parser is used to label the rest of the sentences (with file number from 400 to 931 and 1100 to 1151). After that, the parsed sentences are manually corrected. Two persons marked the correct tags to each named entity independently. Manual correction is necessary, so can we avoid the danger of low-recall. Both persons should agree on a single tag when differences occurred.

The size of our new corpus is shown in Table 3.

CTB files	#Files	#Sens.	#NE	#NestedNE
1-325 1001-1078	403	8971	28344	1754
400-931 1100-1151	487	9778	28149	1144

Table 3: Statistics of the annotated corpus

3.4 Parsing Model

PCFG-LA in Petrov et al. (2006) used a hierarchical state-split approach to refine the original grammars. Starting with the basic non-terminal nodes, this method repeats the split-merge (SM) cycle to increase the complexity of grammars. Specifically, it splits every symbol into two, and then re-merge some new subcategories which cause little or less loss in likelihood incurred when removing it. In other words, the parser introduces latent annotations to refine the syntactic categories.

We employ Berkeley parser[2] in this study. We have re-implemented and enhance the Berkeley parser to handle Chinese character involved in nested named entity words efficiently and robustly. Especially, when the input is character not the word, we will change the strategy to deal the unknown character accordingly .

4 Experiments

In this section, we examine the effect of named entity cue in parsing Chinese. At the same time, the parser output an NER result. For the sake of comparison, here we also train a CRF model for NER as a baseline.

[2]http://code.google.com/p/berkeleyparser/

4.1 Experimental Setup

We present experimental results on Chinese Treebank (CTB) 5.0 with annotation of the named entity information. We adapted the standard data allocation and split the corpus as follows: files from CHTB_001.fid to CHTB_270.fid, and files from CHTB_400.fid to CHTB_1151.fid were used as training set. The development set includes files from CHTB_301.fid to CHTB_325.fid, and the test set includes files CHTB_271.fid to CHTB_300.fid. All traces and functional tags were stripped.

For comparison, we also trained a baseline BerkeleyParser without the cue, and a CRF model for named entity recognition. Our CRFs were implemented based on the CRF++ package [3], and the features used were mentioned in (Wan et al., 2011).

With regard to the parser from (Petrov et al., 2006), all the experiments were carried out after six cycles of split-merge.

4.2 Evaluation Metric

Three metrics were used for the evaluation of syntactic parsing: precision (P), recall (R) and F1-measure (F1) which is defined as 2PR/(P+R).

In the evaluation using the EVALB parseval, the additional named entity labels are also ignored. For instance, the label ‘NP_ORG’ and ‘NR_ORG’ will be replaced as ’NP’ and ’NR’ separately. The internal structure of nested named entity words are discarded by rules to make the results comparable to previous work.

We tested the significance of our results using Dan Bikel’s randomized parsing evaluation comparator[4], and validate the improvement in F1-measure is statistically significant.

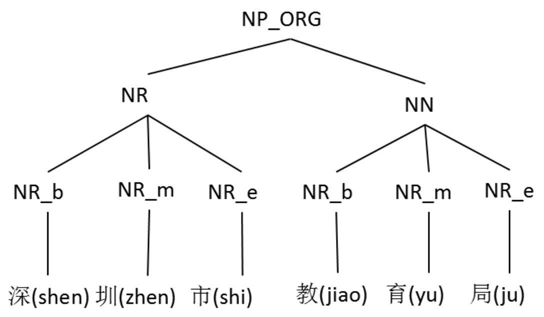
4.3 Results on Parsing

In this section, we examine the effect of joint learning of syntactic structure and named entity cues for parsing.

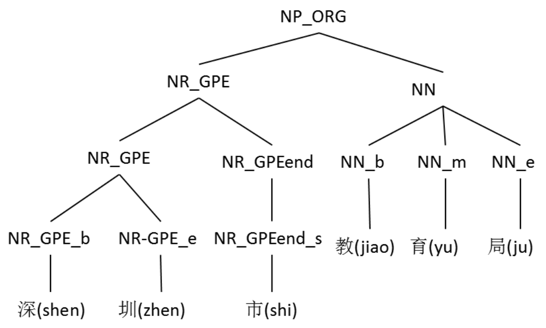
Using the same data set setup and evaluation metric as the previous experiments, our parser achieves performance of 84.43 in F1-measure on the test data. Table 4 lists a few state-of-the-art word-level parser performance, showing that our system is competitive with all the others.

[3]http://crfpp.sourceforge.net/

[4]http://www.cis.upenn.edu/dbikel/software.html



(a) A tree without nested annotation



(b) A tree with nested annotation

Figure 5: Not nested and Nested named entity annotation in the character-level tree for 深(shen) 圳(zhen) 市(shi) 教(jiao) 育(yu) 局(ju) “Shenzhen Education Bureau”

Experiment results show that named entity cue is useful for parsing. PCFG-LA method refines the syntactic categories by latent annotations, whereas, we introduce the fine-grained subcategorizations in the form of explicit annotations. The completely data-driven approach is prone to overfit, and the introduction of named entity cue by manual annotations is a more reliable way than unsupervised clustering.

System	**P**	**R**	**F1**
Petrov ’07	84.8	81.9	83.3
Qian’12	84.57	**83.68**	84.13
This paper	**85.53**	83.34	**84.43**

Table 4: Comparisons of our word-level parsing results with state-of-the-art systems

4.4 Examining the Effectiveness of These Annotations for NER

The above experiments demonstrate that syntactic parsing benefits from our integrated approach. In this section, we exploit the effect on named entity recognition of joint learning.

For comparison to previous work, we convert word-level trees into character-level trees according to some rules. Then, the trained grammar has the ability to parse on characters and output syntactic structure and named entity labels. The simple rules used in this conversion are as follows:

- All part-of-speech tags in Word-level become constituent labels in character-level trees. Then a new node for each character if cerated, and we assign a new label for each new node. The new label consists of the POS tag of its word and its position in its word('b' for starting position, 'e' for end position, and 'm' for others). For example , the character 教"Jiao" in NN-教育局"Jiao Yu Ju", will be labeled as ‘NNb’. [5]

- All the characters underlying the NUM node will replace with “#NUM#”.

In Table 5, we show the NER result of our joint model. In the named entity evaluation, only the named entities with the correct boundaries and the correct categories are regarded as a correct recognition.

Model	GPE	PER	ORG	LOC
CRF	86.98	88.56	48.79	67.28
Parsing+NotNested	85.61	85.63	40.63	54.73
Parsing+NestedNR	**89.64**	**89.97**	**63.44**	**73.07**

Table 5: NER F1 results using different models

There is a great performance improvement on named entity recognition, especially on the recognition for ORG. On one hand, the internal structure of the named entity helps to determine the boundary of the entity. For instance, the organization phrase 中国华侨国际文化交流促进会"China International Cultural Exchange Association of the overseas Chinese" can be recognized . But the CRF model cannot capture the long-distance structure. On the other hand, the structural context in which it appears can help determine the type of the entity. As illustrated in Figure 6, the structure "NP_ORG CC NP_ORG" is a pattern, and the noun phrases on both sides of the 与"and" should be of the same type.

[5]This rule is the same as in Luo (2003) and Li (2011)

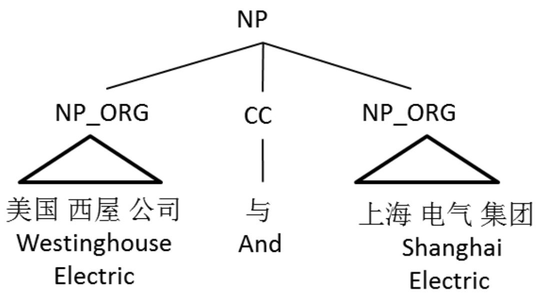


Figure 6: An example parsing result on the phrase 美国西屋公司与上海电气集团"Westinghouse Electric and Shanghai Electric"

5 Conclusion and Future Work

In this paper, we exploit the named entity cue in a unified framework for parsing. We annotate this cue in CTB5.0 through a joint representation of syntactic and named entity structures. Furthermore, we annotate nested named entity structure for all entity names, temporal expressions and number expressions. A PCFG-LA parser is then trained on the corpus. The evaluation shows that, introducing the named entity cue when training a parser help to recognize the complex named entity structures.

This preliminary investigation could be extended in several ways. First, it is natural to introduce other cues together, such as verbal subcategories and function word subcategories. Second, we would like to adopt discriminative parsing to integrate named entity cue into parsing.

Acknowledgment

This research was supported in part by the National Basic Research Program of China (973 Program) under grant 2013CB329304, the Key National Social Science Foundation of China under grant 12&ZD119, the National Natural Science Foundation of China under grant 91120001.

References

Daniel M Bikel and David Chiang. 2000. Two statistical parsing models applied to the chinese treebank. In *Proceedings of the second workshop on Chinese language processing: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 12*, pages 1–6. Association for Computational Linguistics.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics.

Aitao Chen, Fuchun Peng, Roy Shan, and Gordon Sun. 2006. Chinese named entity recognition with conditional probabilistic models. In *5th SIGHAN Workshop on Chinese Language Processing, Australia*.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Jenny Rose Finkel and Christopher D Manning. 2009a. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 326–334. Association for Computational Linguistics.

Jenny Rose Finkel and Christopher D Manning. 2009b. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 141–150. Association for Computational Linguistics.

Mary Harper and Zhongqiang Huang. 2009. Chinese statistical parsing. *Gale Book*.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

Mark Johnson. 1998. Pcfg models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Dan Klein and Christopher D Manning. 2003. A parsing: fast exact viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 40–47. Association for Computational Linguistics.

Alexandre Klementiev and Dan Roth. 2006. Weakly supervised named entity transliteration and discovery from multilingual comparable corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 817–824. Association for Computational Linguistics.

Dongchen Li and Xihong Wu. 2012. Parsing tct with split conjunction categories. In *Proceedings of the Second CIPS-SIGHAN Joint Conference on Chinese Language Processing*, pages 174–178. Association for Computational Linguistics and Chinese Information Processing Society of China.

Zhongguo Li. 2011. Parsing the internal structure of words: A new paradigm for chinese word segmentation. In *ACL*, pages 1405–1414.

Xiaoqiang Luo. 2003. A maximum entropy chinese character-based parser. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 192–199. Association for Computational Linguistics.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic cfg with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human language technologies 2007: the conference of the North American chapter of the Association for Computational Linguistics*, pages 404–411.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Xian Qian and Yang Liu. 2012. Joint chinese word segmentation, pos tagging and parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 501–511. Association for Computational Linguistics.

Satoshi Sekine and Chikashi Nobata. 2004. Definition, dictionaries and tagger for extended named entity hierarchy. In *LREC*.

Sameer Singh, Dustin Hillard, and Chris Leggetter. 2010. Minimally-supervised extraction of entities from text advertisements. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 73–81. Association for Computational Linguistics.

Xiaojun Wan, Liang Zong, Xiaojiang Huang, Tengfei Ma, Houping Jia, Yuqian Wu, and Jianguo Xiao. 2011. Named entity recognition in chinese news comments on the web. In *IJCNLP*, pages 856–864.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated chinese corpus. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–8. Association for Computational Linguistics.

Xiaofeng Yu, Wai Lam, Shing-Kit Chan, Yiu Kei Wu, and Bo Chen. 2008. Chinese ner using crfs and logic for the fourth sighan bakeoff. In *IJCNLP*, pages 102–105.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. *51st Annual Meeting of the Association for Computational Linguistics*.

Improving a symbolic parser through partially supervised learning

Éric Villemonte de la Clergerie
INRIA - Rocquencourt - B.P. 105
78153 Le Chesnay Cedex, FRANCE
Eric.De_La_Clergerie@inria.fr

Abstract

Recently, several statistical parsers have been trained and evaluated on the dependency version of the French TreeBank (FTB). However, older symbolic parsers still exist, including FRMG, a wide coverage TAG parser. It is interesting to compare these different parsers, based on very different approaches, and explore the possibilities of hybridization. In particular, we explore the use of partially supervised learning techniques to improve the performances of FRMG to the levels reached by the statistical parsers.

1 Introduction

Most stochastic parsers are trained and evaluated on the same source treebank (for instance the Penn TreeBank), which, by definition, avoid all problems related to differences between the structures returned by the parsers and those present in the treebank. Some symbolic or hybrid parsers are evaluated on a treebank specifically designed for their underlying formalism, possibly by converting and hand-correcting the treebank from some other annotation scheme (as done in (Hockenmaier and Steedman, 2007)). Besides the cost of the operation, an issue concerns the comparison with other parsers. By contrast, the most problematic case remains the evaluation of a parser on an unrelated treebank and scheme (Sagae et al., 2008).

This situation arose for French with the recent emergence of several statistical parsers trained and evaluated on the French TreeBank (FTB) (Abeillé et al., 2003), in particular under its dependency version (Candito et al., 2010b) represented in CONLL format (Nivre et al., 2007). On the other hand, older parsing systems still exist for French, most of them not based on statistical approaches and not related to FTB. For instance, FRMG is a wide coverage symbolic parser for French (de La Clergerie, 2005), based on Tree Adjoining Grammars (TAGs), that has already participated in several parsing campaigns for French. It was important to be able to compare it with statistical parsers on their native treebank, but also possibly to extend the comparison for other treebanks.

A first necessary step in this direction was a conversion from FRMG's native dependency scheme into FTB's dependency scheme, a tedious task highlighting the differences in design at all levels (segmentation, parts of speech, representation of the syntactic phenomena, etc.). A preliminary evaluation has shown that accuracy is good, but largely below the scores reached by the statistical parsers.

A challenge was then to explore if training on the FTB could be used to improve the accuracy of a symbolic parser like FRMG. However, the main difficulty arises from the fact that FTB's dependency scheme has little in common with FRMG's underlying grammar, and that no reverse conversion from FTB to FRMG structures is available. Such a conversion could be investigated but would surely be difficult to develop. Instead, we tried to exploit directly FTB data, using only very minimal assumptions, nevertheless leading to important gains and results close to those obtained by the statistical parsers. The interest is that the technique should be easily adaptable for training data with different annotation schemes. Furthermore, our motivation was not just to improve the performances on the FTB and for the annotation scheme of FTB, for instance by training a reranker (as often done for domain adaptation), but to exploit the FTB to achieve global improvement over all kinds of corpora and for FRMG native annotation scheme.

Section 2 provides some background about FRMG. We expose in Section 3 how partially supervised learning may be used to improve its performances. Section 4 briefly presents the French TreeBank and several other corpora used for training and evaluation. Evaluation results are presented and discussed in Section 5 with a preliminary analysis of the differences between FRMG and the other statistical parsers.

2 FRMG, a symbolic TAG grammar

FRMG (de La Clergerie, 2005) denotes (a) a French meta-grammar; (b) a TAG grammar (Joshi et al., 1975) generated from the meta-grammar; and (c) a chart-like parser compiled from the grammar. As a parser, FRMG parses DAGs of words, built with SxPipe (Sagot and Boullier, 2008), keeping all potential segmentation ambiguities and with no prior tagging. The parser tries to get *full parses* covering the whole sentence, possibly relaxing some constraints (such as number agreement between a subject and its verb); if not possible, it switches to a *robust mode* looking for a sequence of *partial parses* to cover the sentence.

All answers are returned as shared TAG derivation forests, which are then converted into dependency shared forests, using the anchors of the elementary trees as sources and targets of the dependencies. Some elementary trees being not anchored, pseudo empty words are introduced to serve as source or target nodes. However, in most cases, by a simple transformation, it is possible to reroot all edges related to these pseudo anchors to one of their lexical child.

Finally, the dependency forests are disambiguated using heuristic rules to get a tree. The local *edge rules* assign a positive or negative weight to an edge e, given information provided by e (form/lemma/category/...for the source and target nodes, edge label and type, anchored trees, ...), by neighbouring edges, and, sometimes, by competing edges. A few other *regional rules* assign a weight to a governor node G, given a set of children edges forming a valid derivation from G. The disambiguation algorithm uses dynamic programming techniques to sum the weights and to return the best (possibly non-projective) dependency tree, with maximal weight.

Several conversion schemes may be applied on FRMG's native dependency trees. A recent one returns dependency structures following the annotation scheme used by the dependency version of the French TreeBank and represented using the column-based CONLL format (Nivre et al., 2007). The conversion process relies on a 2-stage transformation system, with constraints on edges used to handle non-local edge propagation, as formalized in (Ribeyre et al., 2012). Figure 1 illustrates the native FRMG's dependency structure (top) and, on the lower side, its conversion to FTB's dependency scheme (bottom). One may observe differences between the two dependency trees, in particular with a (non-local) displacement of the root node. It may be noted that FTB's scheme only considers projective trees, but that the conversion process is not perfect and may return non projective trees, as shown in Figure 1 for the `p_obj` edge.

Let's also mention an older conversion process from FRMG dependency scheme to the EASy/Passage scheme, an hybrid constituency/dependency annotation scheme used for the first French parsing evaluation campaigns (Paroubek et al., 2009). This scheme is based on a set of 6 kinds of chunks and 14 kinds of dependencies.

3 Partially supervised learning

The set of weights attached to the rules may be seen as a statistical model, initially tailored by hand, through trials. It is tempting to use training data, provided by a treebank, and machine learning techniques to improve this model. However, in our case, the "annotation schemes" for the training data (FTB) and for FRMG are distinct. In other words, the training dependency trees cannot be immediately used as oracles as done in most supervised learning approaches, including well-known perceptron ones. Still, even partial information extracted from the training data may help, using partially supervised learning techniques.

Figure 2 shows the resulting process flow. Learning is done, using the disambiguated dependency trees produced by FRMG on training sentences, with (partial) information about the discarded alternatives. The resulting statistical model may then be used to guide disambiguation, and be improved through iterations. Actually, this simple process may be completed with the construction and use of (imperfect) oracles adapted to FRMG. The learning component can produce such an oracle but can also exploit it. Even better, the oracle can be directly used to guide the disam-

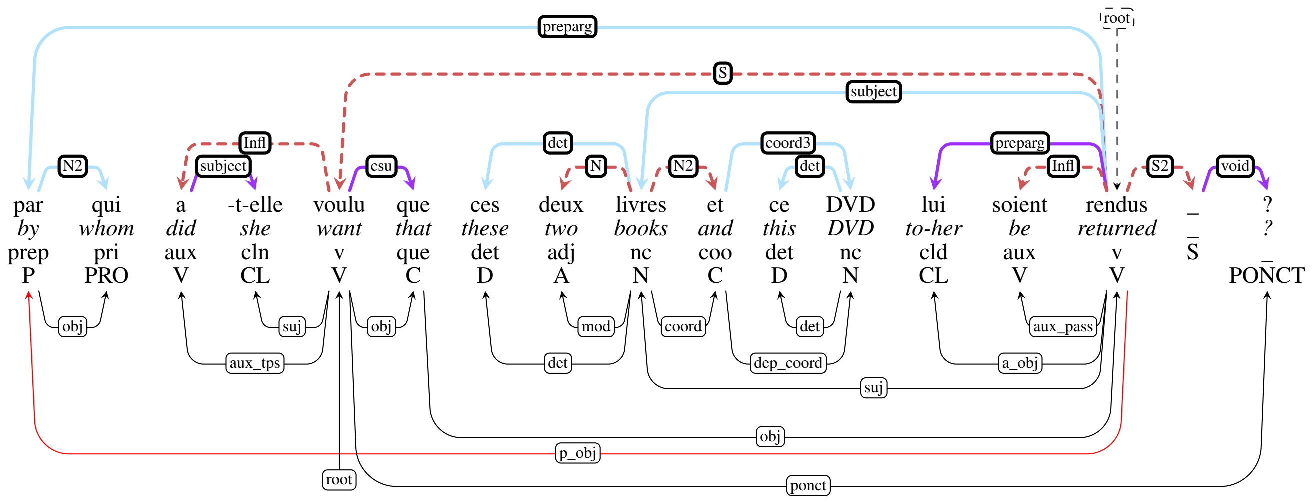


Figure 1: Sample of disambiguated FRMG output, without conversion (top) and with FTB conversion (bottom)

biguation process. Again, by iterating the process, one can hopefully get an excellent oracle for the learning component, useful to get better models.

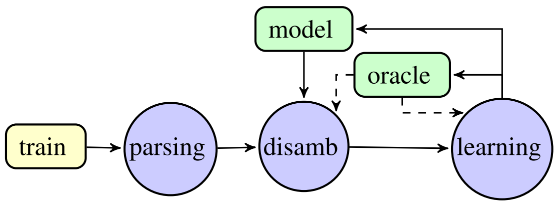


Figure 2: Parsing and partially supervised learning with imperfect oracles

The minimal information we have at the level of a word w is the knowledge that the its incoming dependency d proposed by conversion is correct or not, leading to 4 situations as summarized in Table 1.

	d is correct	d is not correct
selected D	favor r	penalize r'
competitor D'	penalize r'	??

Table 1: Handling weight updates for rules

For the FTB conversion, when d is correct, we can generally assume that the FRMG incoming dependency D for w is also correct and that disambiguation is correct in selecting D. We can then consider than any edge disambiguation rule r applicable on D should then be favored by (slightly) increasing its weight and rules applying on competitors of D should see their weight decrease (to reinforce the non-selection of D).

On the other hand, when d is not correct, we should penalize the rules r applying on D and try to favor some competitor D' of D (and favor the rules r' applying to D'). However, we do not know which competitor should be selected, except in cases where there is only one possible choice. By default, we assume that all competitors have equal chance to be the correct choice and favor/penalize in proportion their rules. If we have n competitors, we can say that it is a bad choice not to keep D' in $\frac{1}{n}$ cases (and should favor rules r') and it is a bad choice to keep D' in $\frac{n-1}{n}$ cases (and should penalize rules r'). So, practically, the dependency D' falling in the problematic case is distributed between the keep/bad case (with weight $\frac{1}{n}$) and the discard/bad case (with weight $\frac{n-1}{n}$). These proportions may be altered if we have more precise information about the competitors, provided by an oracle (as hinted in Figure 2), weights, ranks, or other elements. For instance, if we known that d is not correct but has the right dependency label or the right governor, we use this piece of information to discard some competitors and rerank the remaining ones.

Of course, the update strategy for the problematic case will fail in several occasions. For instance, maybe D is the right FRMG dependency to keep, but the conversion process is incorrect and produces a bad

FTB dependency d. Maybe FRMG is incomplete and has no correct source dependency D. Finally, maybe d (with target word w) derives from some source dependency $D_{w'}$ for some other target word w'. We assume that these cases remain limited and that improving edge selection for the easy cases will then guide edge selection for the more complex cases.

The learning algorithm could be used online, adjusting the weights when processing a sentence. However, we have only implemented an offline version where the weights are updated after considering all training sentences (but discarding some long sentences and sentences with low accuracy scores).

More formally, given the parses for the training sentences, for any edge disambiguation rule r and value tuple $\mathbf{v}$ for a feature template $\mathbf{f}$, we compute the number $n_{r,\mathbf{f}=\mathbf{v}}$ of occurrences of r in context $\mathbf{f} = \mathbf{v}$, and $\text{keep}^{\text{ok}}_{r,\mathbf{f}=\mathbf{v}}$ the number of occurrences where the edge was selected and it was a correct choice. Similarly, but taking into account the above-mentioned redistribution, we compute $\text{discard}^{\text{ok}}_{r,\mathbf{f}=\mathbf{v}}$, $\text{keep}^{\text{bad}}_{r,\mathbf{f}=\mathbf{v}}$, and $\text{discard}^{\text{bad}}_{r,\mathbf{f}=\mathbf{v}}$.

These figures are used to compute an adjustment $\delta_{r,\mathbf{f}=\mathbf{v}}$ added to the base weight w_r of r for context $\mathbf{f} = \mathbf{v}$, using Eq (1), where θ denotes a *temperature*:

$$(1)\quad \delta_{r,\mathbf{f}=\mathbf{v}} = \theta . a_{r,\mathbf{f}=\mathbf{v}} . \begin{cases} \text{discard}^{\text{bad}}_{r,\mathbf{f}=\mathbf{v}} & \text{if } a_{r,\mathbf{f}=\mathbf{v}} > 0 \\ \text{keep}^{\text{bad}}_{r,\mathbf{f}=\mathbf{v}} & \text{otherwise} \end{cases}$$

The $a_{r,\mathbf{f}=\mathbf{v}}$ factor is related to the direction and force of the expected change[1], being positive when selecting an edge thanks to r tends to be a good choice, and negative otherwise (when the edge should rather be discarded), as expressed in the following formula:

$$a_{r,\mathbf{f}=\mathbf{v}} = \frac{\text{keep}^{\text{ok}}}{\text{keep}^{\text{ok}} + \text{keep}^{\text{bad}}} - \frac{\text{discard}^{\text{ok}}}{\text{discard}^{\text{ok}} + \text{discard}^{\text{bad}}}$$

The last factor in Eq (1) is the number of edges whose status (selected or discarded) should ideally change.

The process is iterated, reducing the temperature at each step, and we keep the best run. At each iteration, the edges found to be correctly kept or discarded are added to an oracle for the next iteration.

[1]It may be noted that the interpretation of $a_{r,\mathbf{f}=\mathbf{v}}$ may sometimes be unclear, when both $\text{keep}^{\text{ok}}_{r,\mathbf{f}=\mathbf{v}}$ and $\text{discard}^{\text{ok}}_{r,\mathbf{f}=\mathbf{v}}$ are low (i.e., when neither keeping or discarding the corresponding edges is a good choice). We believe that these cases signal problems in the conversion process or the grammar.

We use standard features such as form, lemma, pos, suffixes, sub-categorization information, morphosyntactic features, anchored TAG trees for words (dependency heads and targets, plus adjacent words); and dependency distances, direction, type, label, and rank for the current dependency and possibly for its parent. For smoothing and out-of-domain adaptation, we add a *cluster* feature attached to forms and extracted from a large raw textual corpus using Brown clustering (Liang, 2005). It may noted that the name of a disambiguation rule may be considered as the value of a `rule` feature. Each feature template includes the label and type for the current FRMG dependency.

It seems possible to extend the proposed learning mechanism to adjust the weight of the regional rules by considering (second-order) features over pairs of adjacent sibling edges (for a same derivation). However, preliminary experiments have shown an explosion of the number of such pairs, and no real gain.

4 The corpora

The learning method was tried on the French TreeBank(Abeillé et al., 2003), a journalistic corpus of 12,351 sentences, annotated in morphology and constituency with the Penn TreeBank format, and then automatically converted into projective dependency trees, represented in the CONLL format (Candito et al., 2010a). For training and benchmarking, the treebank is split into three parts, as summarized in Table 2.

		Cover.	**Time (s)**	
Corpus	**#sent.**	**(%)**	**Avg**	**Median**
FTB train	9,881	95.9	1.04	0.26
FTB dev	1,235	96.1	0.88	0.30
FTB test	1,235	94.9	0.85	0.30
Sequoia	3,204	95.1	1.53	0.17
EASyDev	3,879	87.2	0.87	0.14

Table 2: General information on FTB and other corpora

To analyze the evolution of the performances, we also consider two other corpora. The **Sequoia** corpus (Candito and Seddah, 2012) also uses the FTB dependency scheme (at least for its version 3), but covers several styles of documents (medical, encyclopedic, journalistic, and transcription of political discourses). The **EASyDev** corpus also covers various styles (journalistic, literacy, medical, mail, speech, . . .), but was annotated following the EASy/Passage scheme for evaluation campaigns (Paroubek et al., 2006).

Table 2 shows that coverage (by full parses) is high for all corpora (slightly lower for **EASyDev** because of the mail and speech sub-corpora). Average time per sentence is relatively high but, as suggested by the much lower median times, this is largely due to a few long sentences and due to a large timeout.

5 Results and discussions

Table 3 shows evaluation results for different versions of FRMG on each corpus. On **FTB** and **Sequoia**, we use Labelled Attachment Scores (LAS) without taking into account punctuation, and, on **EASyDev**, F1-measure on the dependencies[2]. The *init* system corresponds to a baseline version of FRMG with a basic set of rules and hand-tailored weights. The *+restr* version of FRMG adds *restriction rules*, exploiting attachment preferences and word (semantic) similarities extracted from a very large corpus parsed with FRMG, using Harris distributional hypothesis[3]. This version shows that unsurpervised learning methods already improve significantly the performances of a symbolic parser like FRMG for all corpora. The *+tuning* version of FRMG keeps the restriction rules and adds the partially supervised learning method. We observe large improvements on the FTB dev and test parts (between 4 and 5 points), but also on **Sequoia** (almost 3 points) on different styles of documents. We also get similar gains on **EASyDev**, again for a large diversity of styles, and, more interestingly, for a different annotation scheme and evaluation metric.

The bottom part of Table 3 lists the accuracy of 3 statistical parsers on FTB as reported in (Candito et al., 2010b). The Berkeley parser (BKY) is a constituent-based parser whose parses are then converted into FTB dependencies (using the same tool used to convert the FTB). MALT parser is a greedy transition-based parser while MST (*maximum spanning tree*) globally extracts the best dependency tree from all possible ones. We see that FRMG (with tuning) is better than BKY on the test part (but not on the dev part), close to MALT, and still below MST. Clearly, tuning allows FRMG to be more competitive with statistical parsers even on their native treebank.

[2]F1-measures on chunks are less informative.

[3]We used a 700Mwords corpus composed of AFP news, French Wikipedia, French Wikisource, etc.. The attachment weights are used for handling PP attachments over verbs, nouns, adjectives, but also for relatives over antecedents, or for filling some roles (subject, object, attribute). Similarities between words are used for handling coordination.

	FTB			other corpora	
system	train	dev	test	Sequoia	EASy
init	79.95	80.85	82.08	81.13	65.92
+restr	80.67	81.72	83.01	81.72	66.33
+tuning	86.60	85.98	87.17	84.56	69.23
BKY	–	86.50	86.80	–	–
MALT	–	86.90	87.30	–	–
MST	–	87.50	88.20	–	–

Table 3: Performances of various systems on French data

system	emea-test	ftb-test	loss
BKY (evalb)	80.80	86.00	5.20
FRMG+tuning (LAS)	84.13	87.17	3.04

Table 4: Evolution for an out-of-domain medical corpus

We do not have results for the 3 statistical parsers on the **Sequoia** corpus. However, (Candito and Seddah, 2012) reports some results for Berkeley parser on constituents for the medical part of **Sequoia**, listed in Table 4. The metrics differ, but we observe a loss of 5.2 for BKY and only of 3.04 for FRMG, which tends to confirm the stability of FRMG across domains, possibly due to the constraints of its underlying linguistically-motivated grammar (even if we observe some over-fitting on FTB).

Figure 3 shows the evolution of accuracy on the 3 components of FTB during the learning iterations. We observe that learning is fast with a very strong increase at the first iteration, and a peak generally reached at iterations 3 or 4. As mentioned in Section 3, rule names may be seen as feature values and it is possible to discard them, using only a single *dummy* rule. This *dummy* edge rule checks nothing on the edges but only acts as a default value for the `rule` feature. However, old experiments showed a LAS on FTB dev of 84.31% keeping only a dummy rule and of 85.00% with all rules, which seems to confirm the (global) pertinence of the hand-crafted disambiguation rules. Indeed, these rules are able to consult additional information (about adjacent edges and alternative edges) not available through the other features. [4]

As suggested in Figure 2, the oracle built by the learning component on FTB train may be used during disambiguation (on FTB train) by setting a very high weight for the edges in the oracle and a very low weight for the others. The disambiguation process is then strongly encouraged to select the edges of the oracle (when possible). Iterating the process, we reach an accuracy of 89.35% on FTB train, an interesting first step in direction of a FRMG version of the FTB[5].

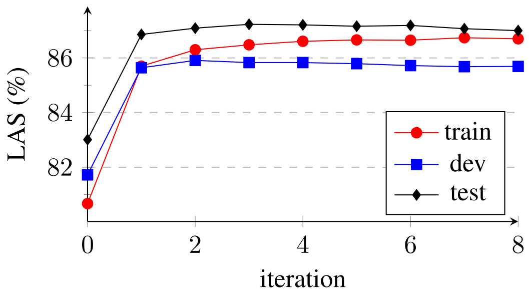


Figure 3: LAS evolution on FTB train per iteration

One reason still explaining the differences between FRMG and the statistical parsers arises from the conversion process to FTB annotation scheme being not perfect. For instance, FRMG and FTB do not use the same list of multi-word expressions, leading to problems of mapping between words and of dependency attachments, in particular for complex prepositions and conjunctions. The segmenter SXPIPE also recognizes named entities such as *Communauté européenne* (*European Community*), *5 millions*, or *Mardi prochain* (*next Tuesday*) as single terms whereas FTB adds internal dependencies for these expressions. During the conversion phase, most of the missing dependencies are added leading to an accuracy of 75.38% on the specific dependencies in FTB train (around 3.5% of all dependencies), still largely below the global accuracy (86.6%). There are also 1259 sentences in FTB train (12.7%) where FRMG produces non-projective trees when FTB expects projective ones.[6]

[4]However, it is clear that some disambiguation rules are redundant with the other features and could be discarded.

[5]The problem is that the treebank would have to be regenerated to follow the evolution of FRMG.

[6]It does not mean that so many FRMG trees are non-projective, just that the conversion builds non-projective trees, because of edge movement. A quick investigation has shown that many cases were related to punctuation attachment.

Then, following (McDonald and Nivre, 2007), we tried to compare the performance of FRMG, MST, and MALT with respect to several properties of the dependencies. Figure 4(a) compares the recall and precision of the systems w.r.t. the distance of the dependencies (with, in background, the number of gold dependencies). We observe that all systems have very close recall scores for small distances, then MST is slightly better, and, at long distance, both MST and MALT are better. On the other hand, FRMG has a much better precision than MALT for long distance dependencies. One may note the specific case of null distance dependencies actually corresponding to root nodes, with lower precision for FRMG. This drop corresponds to the extra root nodes added by FRMG in robust mode when covering a sentence with partial parses.

As shown in Figure 4(b), the recall curves w.r.t. dependency depths are relatively close, with FRMG slightly below for intermediate depths and slightly above for large depths. Again, we observe a precision drop for root nodes (depth=0) which disappears when discarding the sentences in robust mode.

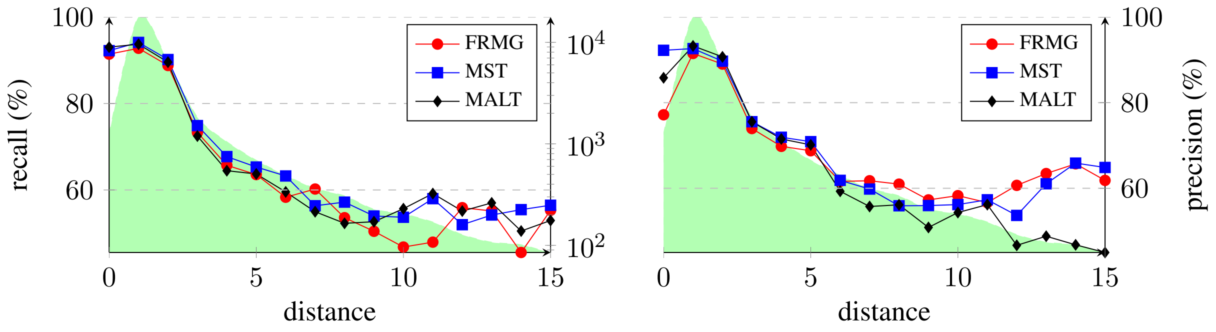
In Figure 4(c), we get again a lower recall for large numbers of sibling edges with, surprisingly, a much higher precision for the same values.

Figure 4(d) compares recall and precision w.r.t. dependency rank[7], with again the lower precision due to the extra root nodes (rank=0) and again a lower recall and higher precision for large absolute ranks.

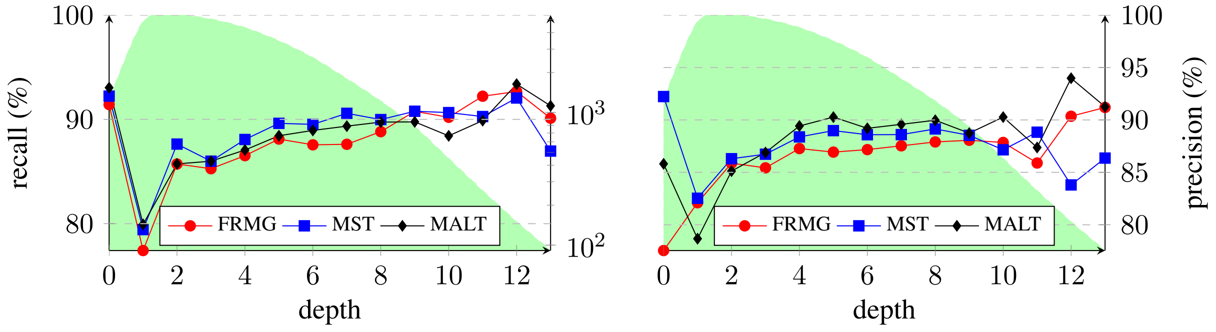
More generally, FRMG tends to behave like MST rather than like MALT. We hypothesize that it reflects than both systems share a more global view of the dependencies, in particular thanks to the domain locality provided by TAGs for FRMG.

Figure 5 shows recall wrt some of the dependency labels. The most striking point is the weak recall for coordination by all systems but, nevertheless, the better score of FRMG. We observe a lower recall of FRMG for some verbal prepositional arguments (`a_obj`, `de_obj`) that may be confused with verb modifiers or attached to a noun or some other verb. Verbal modifiers (`mod`), a category covering many different syntactic phenomena, seem also difficult, partly due to the handling of prepositional attachments. On

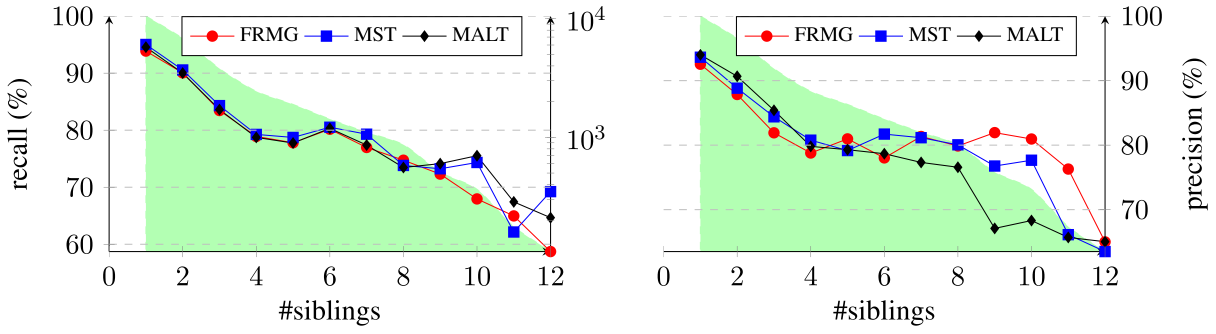
[7]defined as the number of siblings (plus 1) between a dependant and its head, counted positively rightward and negatively leftward.



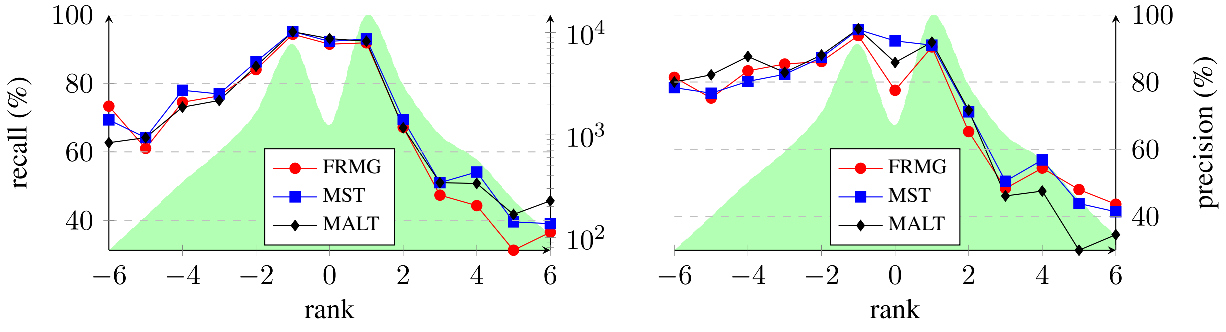
(a) w.r.t. dependency distance



(b) w.r.t. dependency depth



(c) w.r.t. number of siblings



(d) w.r.t. dependency rank

Figure 4: System comparison

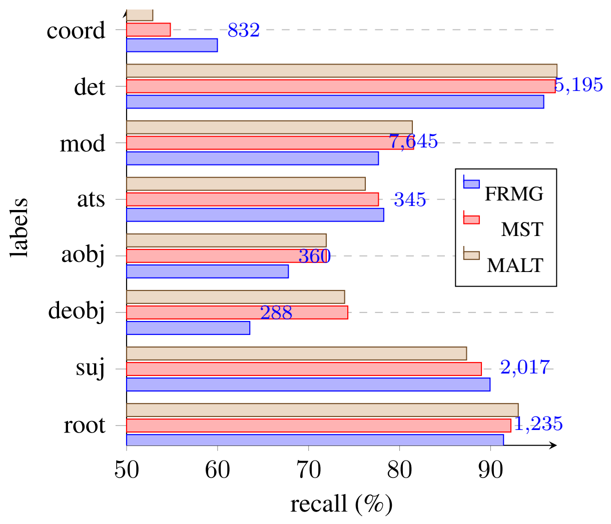


Figure 5: System comparison w.r.t. dependency labels

the other hand, FRMG has a better recall for subjects, possibly because the grammar accepts a large range of positions and realization for subjects.

6 Conclusion

We have presented a new partially supervised learning approach exploiting the information of a training treebank for tuning the disambiguation process of FRMG, a symbolic TAG-based parser. Even considering minimal assumptions for transferring oracle information from the training treebank, we strongly improve accuracy, allowing FRMG to be on par with statistical parsers on their native treebank, namely the French TreeBank. Even if the gains are important, several extensions of the learning algorithm have still to be explored, in particular to build and exploit better oracles, and to incorporate more higher order features, such as sibling features.

The approach explored in this paper, even if tried in the specific context of FRMG, is susceptible to be adapted for other similar contexts, in particular when some imperfect annotation conversion process takes place between a disambiguation process and a training treebank. However, some work remains be done to get a better characterization of the learning algorithm, for instance w.r.t. perceptrons.

We are aware that some of the data collected by the learning algorithm could be used to track problems either in the conversion process or in FRMG grammar (by exploring the cases where neither selecting or discarding an edge seems to be a good choice). We would like to fix these problems, even if most of them seem to have very low frequencies. The conversion process could also be improved by allowing some non-deterministic choices, again controlled by probabilistic features. However, it is not yet clear how we can couple learning for the disambiguation process and learning for the conversion process.

More investigations and comparisons are needed, but some hints suggest that an underlying linguistically-motivated grammar ensures a better robustness w.r.t. document styles and domains. On the other hand, the evaluation shows that the choices made in FRMG to handle lack of full coverage using partial parses should be improved, maybe by using some guiding information provided by a statistical parser to handle the problematic areas in a sentence.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In Anne Abeillé, editor, *Treebanks*. Kluwer, Dordrecht.

Marie Candito and Djamé Seddah. 2012. Le corpus sequoia: annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In *TALN 2012-19e conférence sur le Traitement Automatique des Langues Naturelles*.

Marie Candito, Benoît Crabbé, and Pascal Denis. 2010a. Statistical french dependency parsing: treebank conversion and first results. In *Proceedings of the 7th Language Resources and Evaluation Conference (LREC'10)*, La Valette, Malte.

Marie Candito, Joakim Nivre, Pascal Denis, and Enrique Henestroza Anguiano. 2010b. Benchmarking of statistical dependency parsers for french. In *Proceedings of COLING'2010 (poster session)*, Beijing, China.

Éric de La Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT'05 (poster)*, pages 190–191, Vancouver, Canada.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Aravind K. Joshi, Leon Levy, and Makoto Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Science 10*, 10(1):136–163.

Percy Liang. 2005. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology.

Ryan T. McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*, pages 122–131.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *The CoNLL 2007 shared task on dependency parsing*.

Patrick Paroubek, Isabelle Robba, Anne Vilnat, and Christelle Ayache. 2006. Data, Annotations and Measures in EASy, the Evaluation Campaign for Parsers of French. In *Proceedings of the 5th international conference on Language Resources and Evaluation (LREC'06)*, Gênes, Italie.

Patrick Paroubek, Éric Villemonte de la Clergerie, Sylvain Loiseau, Anne Vilnat, and Gil Francopoulo. 2009. The PASSAGE syntactic representation. In *7th International Workshop on Treebanks and Linguistic Theories (TLT7)*, Groningen, January.

Corentin Ribeyre, Djamé Seddah, and Éric Villemonte De La Clergerie. 2012. A Linguistically-motivated 2-stage Tree to Graph Transformation. In Chung-Hye Han and Giorgio Satta, editors, *TAG+11 - The 11th International Workshop on Tree Adjoining Grammars and Related Formalisms - 2012*, Paris, France. INRIA.

K. Sagae, Y. Miyao, T. Matsuzaki, and J. Tsujii. 2008. Challenges in mapping of syntactic representations for framework-independent parser evaluation. In *Proceedings of the Workshop on Automated Syntatic Annotations for Interoperable Language Resources at the First International Conference on Global Interoperability for Language Resources (ICGL'08)*, Hong-Kong, January.

Benoît Sagot and Pierre Boullier. 2008. SxPipe 2 : architecture pour le traitement présyntaxique de corpus bruts. *Traitement Automatique des Langues (T.A.L.)*, 49(2):155–188.

On Different Approaches to Syntactic Analysis Into Bi-Lexical Dependencies An Empirical Comparison of Direct, PCFG-Based, and HPSG-Based Parsers

Angelina Ivanova♠, Stephan Oepen♠♡, Rebecca Dridan♠, Dan Flickinger♣, and Lilja Øvrelid♠

♠ University of Oslo, Department of Informatics

♡ Potsdam University, Department of Linguistics

♣ Stanford University, Center for the Study of Language and Information

{ angelii | oe | rdridan | liljao }@ifi.uio.no, danf@stanford.edu

Abstract

We compare three different approaches to parsing into syntactic, bi-lexical dependencies for English: a 'direct' data-driven dependency parser, a statistical phrase structure parser, and a hybrid, 'deep' grammar-driven parser. The analyses from the latter two are post-converted to bi-lexical dependencies. Through this 'reduction' of all three approaches to syntactic dependency parsers, we determine empirically what performance can be obtained for a common set of dependency types for English, across a broad variety of domains. In doing so, we observe what trade-offs apply along three dimensions, accuracy, efficiency, and resilience to domain variation. Our results suggest that the hand-built grammar in one of our parsers helps in both accuracy and cross-domain performance.

1 Motivation

Bi-lexical dependencies, i.e. binary head–argument relations holding exclusively between lexical units, are widely considered an attractive target representation for syntactic analysis. At the same time, Cer et al. (2010) and Foster et al. (2011), inter alios, have demonstrated that higher dependency accuracies can be obtained by parsing into a phrase structure representation first, and then reducing parse trees into bi-lexical dependencies.[1] Thus, if one is willing to accept pure syntactic dependencies as a viable interface (and evaluation) representation, an experimental setup like the one of Cer et al. (2010) allows the exact experimental comparison of quite different parsing approaches.[2] Existing such studies to date are limited to purely data-driven (or statistical) parsers, i.e. systems where linguistic knowledge is exclusively acquired through supervised machine learning from annotated training data. For English, the venerable Wall Street Journal (WSJ) portion of the Penn Treebank (PTB; Marcus et al., 1993) has been the predominant source of training data, for phrase structure and dependency parsers alike.

Two recent developments make it possible to broaden the range of parsing approaches that can be assessed empirically on the task of deriving bi-lexical syntactic dependencies. Flickinger et al. (2012) make available another annotation layer over the same WSJ text, 'deep' syntacto-semantic analyses in the linguistic framework of Head-Driven Phrase Structure Grammar (HPSG; Pollard & Sag, 1994; Flickinger, 2000). This resource, dubbed DeepBank, is available since late 2012. For the type of HPSG analyses recorded in DeepBank, Zhang and Wang (2009) and Ivanova et al. (2012) define a reduction into bi-lexical syntactic dependencies, which they call Derivation Tree-Derived Dependencies (DT). Through application of the converter of Ivanova et al. (2012) to DeepBank, we can thus obtain a DT-annotated version of the standard WSJ text, to train and test a data-driven dependency and phrase structure parser, respectively, and to compare parsing results to a hybrid, grammar-driven HPSG parser. Furthermore, we can draw on a set of additional corpora annotated in the same HPSG format (and thus amenable to conversion for both phrase structure and dependency parsing), instantiating a comparatively diverse range of domains and genres (Oepen et al., 2004). Adding this data to our setup for additional cross-domain testing, we seek to document not only what trade-offs apply in terms of dependency accuracy vs. parser efficiency, but also how these trade-offs are affected by domain and genre variation, and, more generally, how resilient the different approaches are to variation in parser inputs.

[1] This conversion from one representation of syntax to another is lossy, in the sense of discarding constituency information, hence we consider it a reduction in linguistic detail.

[2] In contrast, much earlier work on cross-framework comparison involved post-processing parser outputs in form *and* content, into a target representation for which gold-standard annotations were available. In § 2 below, we argue that such conversion inevitably introduces blur into the comparison.

2 Related Work

Comparing between parsers from different frameworks has long been an area of active interest, ranging from the original PARSEVAL design (Black et al., 1991), to evaluation against 'formalism-independent' dependency banks (King et al., 2003; Briscoe & Carroll, 2006), to dedicated workshops (Bos et al., 2008). Grammatical Relations (GRs; Briscoe & Carroll, 2006) have been the target of a number of benchmarks, but they require a heuristic mapping from 'native' parser outputs to the target representations for evaluation, which makes results hard to interpret. Clark and Curran (2007) established an upper bound by running the mapping process on gold-standard data, to put into perspective the mapped results from their CCG parser proper. When Miyao et al. (2007) carried out the same experiment for a number of different parsers, they showed that the loss of accuracy due to the mapping process can swamp any actual parser differences. As long as heuristic conversion is required before evaluation, cross-framework comparison inevitably includes a level of fuzziness. An alternative approach is possible when there is enough data available in a particular representation, and conversion (if any) is deterministic. Cer et al. (2010) used Stanford Dependencies (de Marneffe & Manning, 2008) to evaluate a range of statistical parsers. Pre- or post-converting from PTB phrase structure trees to the Stanford dependency scheme, they were able to evaluate a large number of different parsers.

Fowler and Penn (2010) formally proved that a range of Combinatory Categorial Grammars (CCGs) are context-free. They trained the PCFG Berkeley parser on CCGBank, the CCG annotation of the PTB WSJ text (Hockenmaier & Steedman, 2007), advancing the state of the art in terms of supertagging accuracy, PARSEVAL measures, and CCG dependency accuracy. In other words, a specialized CCG parser is not necessarily more accurate than the general-purpose Berkeley parser; this study, however, fails to also take parser efficiency into account.

In related work for Dutch, Plank and van Noord (2010) suggest that, intuitively, one should expected that a grammar-driven system can be more resiliant to domain shifts than a purely data-driven parser. In a contrastive study on parsing into Dutch syntactic dependencies, they substantiated this expectation by showing that their HPSG-based Alpino system performed better and was more resilient to domain variation than data-driven direct dependency parsers.

3 Background: Experimental Setup

In the following, we summarize data and software resources used in our experiments. We also give a brief introduction to the DT syntactic dependency scheme and a comparison to 'mainstream' representations.

DeepBank HPSG analyses in DeepBank are manually selected from the set of parses licensed by the English Resource Grammar (ERG; Flickinger, 2000). Figure 1 shows an example ERG derivation tree, where labels of internal nodes name HPSG constructions (e.g. subject–head or head–complement: sb-hd_mc_c and hd-cmp_u_c, respectively; see below for more details on unary rules). Preterminals are labeled with fine-grained lexical categories, dubbed ERG lexical types, that augment common parts of speech with additional information, for example argument structure or the distinction between count, mass, and proper nouns. In total, the ERG distinguishes about 250 construction types and 1000 lexical types.

DeepBank annotations were created by combining the native ERG parser, dubbed PET (Callmeier, 2002), with a discriminant-based tree selection tool (Carter, 1997; Oepen et al., 2004), thus making it possible for annotators to navigate the large space of possible analyses efficiently, identify and validate the intended reading, and record its full HPSG analysis in the treebank. Owing to this setup, DeepBank in its current version 1.0 lacks analyses for some 15 percent of the WSJ sentences, for which either the ERG parser failed to suggest a set of candidates (within certain bounds on time and memory usage), or the annotators found none of the available parses acceptable.[3] Furthermore, DeepBank annotations to date only comprise the first 21 sections of the PTB WSJ corpus. Following the splits suggested by the DeepBank developers, we train on Sections 0–19, use Section 20 for tuning, and test against Section 21 (abbreviated as WSJ below).[4]

[3]Thus, limitations in the current ERG and PET effectively lead to the exclusion of a tangible percentage of sentences from our training and testing corpora. We discuss methodological ramifications of this setup to our study in § 9 below.

[4]To 'protect' Section 21 as unseen test data, also for the ERG parser, this final section in Version 1.0 of DeepBank was not ex-

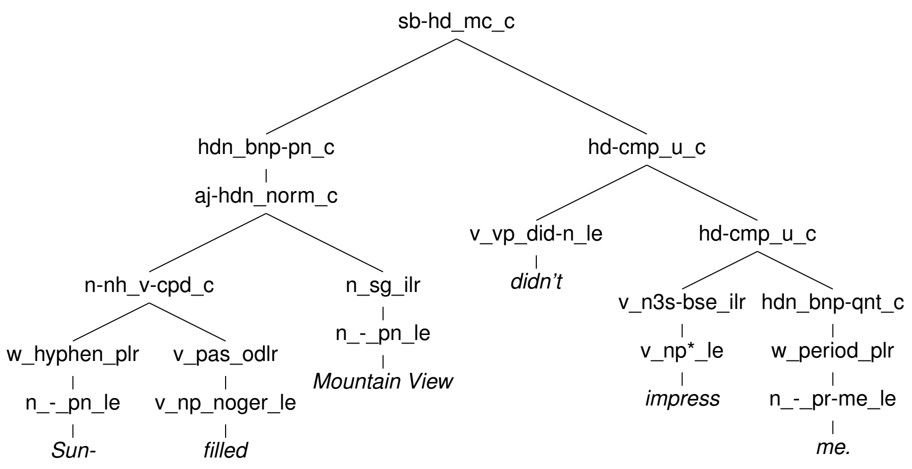


Figure 1: Sample HPSG derivation: construction identifiers label internal nodes, lexical types the preterminals.

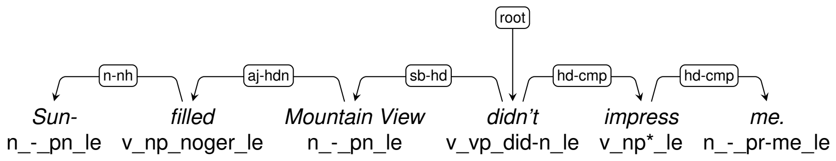


Figure 2: Sample DT bi-lexical dependencies: construction identifiers are generalized at the first underscore.

DT Dependencies As ERG derivations are grounded in a formal theory of grammar that explicitly marks heads, mapping these trees onto bi-lexical dependencies is straightforward (Zhang & Wang, 2009). Ivanova et al. (2012) coin the term DT for ERG Derivation Tree-Derived Dependencies, where they reduce the inventory of some 250 ERG syntactic rules to 48 broad HPSG constructions. The DT syntactic dependency tree for our running example is shown in Figure 2.

To better understand the nature of the DT scheme, Ivanova et al. (2012) offer a quantitative, structural comparison against two pre-existing dependency standards for English, viz. those from the CoNLL dependency parsing competitions (Nivre et al., 2007) and the 'basic' variant of Stanford Dependencies. They observe that the three dependency representations are broadly comparable in granularity and that there are substantial structural correspondences between the schemes. Measured as average Jaccard similarity over unlabeled dependencies, they observe the strongest correspondence between DT and CoNLL (at a Jaccard index of 0.49, compared to 0.32 for DT and Stanford, and 0.43 between CoNLL and Stanford).

posed to its developers until the grammar and disambiguation model were finalized and frozen for this release.

Ivanova et al. (2013) complement this comparison of dependency schemes through an empirical assessment in terms of 'parsability', i.e. accuracy levels available for the different target representations when training and testing a range of state-of-the-art parsers on the same data sets. In their study, the dependency parser of Bohnet and Nivre (2012), henceforth B&N, consistently performs best for all schemes and output configurations. Furthermore, parsability differences between the representations are generally very small.

Based on these observations, we conjecture that DT is as suitable a target representation for parser comparison as any of the others. Furthermore, two linguistic factors add to the attractiveness of DT for our study: it is defined in terms of a formal (and implemented) theory of grammar; and it makes available more fine-grained lexical categories, ERG lexical types, than is common in PTB-derived dependency banks.

Cross-Domain Test Data Another benefit of the DT target representation is the availability of comparatively large and diverse samples of additional test data. The ERG Redwoods Treebank (Oepen et al., 2004) is similar in genealogy and format to DeepBank, comprising corpora from various domains and genres. Although Redwoods counts a total of some 400,000 annotated tokens, we only draw on it for addi-

	Name	Sentences	Tokens	Types
DeepBank	**Train**	33,783	661,451	56,582
	Tune	1,721	34,063	8,964
	WSJ	1,414	27, 515	7,668
Redwoods	CB	608	11,653	3,588
	SC	864	13,696	4,925
	VM	993	7,281	1,007
	WS	520	8,701	2,974

Table 1: Sentence, token, and type counts for data sets.

tional *testing* data. In other words, we do not attempt parser re-training or adaptation against this additional data, but rather test our WSJ-trained parsers on out-of-domain samples from Redwoods. We report on four such test corpora, viz. (a) a software advocacy essay, *The Cathedral and the Bazaar* (CB); (b) a subset of the SemCor portion of the Brown Corpus (SC; Francis & Kucera, 1982); (c) a collection of transcribed, task-oriented spoken dialogues (VM; Wahlster, 2000); and (d) part of the Wikipedia-derived WeScience Corpus (WS; Ytrestøl et al., 2009). Table 1 provides exact sentence, token, and type counts for these data sets.

Tokenization Conventions A relevant peculiarity of the DeepBank and Redwoods annotations in this context is the ERG approach to tokenization. Three aspects in Figure 1 deviate from the widely used PTB conventions: (a) hyphens (and slashes) introduce token boundaries; (b) whitespace in multi-word lexical units (like *ad hoc*, *of course*, or *Mountain View*) does not force token boundaries; and (c) punctuation marks are attached as 'pseudo-affixes' to adjacent words, reflecting the rules of standard orthography. Adolphs et al. (2008) offer some linguistic arguments for this approach to tokenization, but for our purposes it suffices to note that these differences to PTB tokenization may in part counter-balance each other, but do increase the types-per-tokens ratio somewhat. This property of the DeepBank annotations, arguably, makes English look somewhat similar to languages with moderate inflectional morphology. To take advantage of the fine-grained ERG lexical categories, most of our experiments assume ERG tokenization. In two calibration experiments, however, we also investigate the effects of tokenization differences on our parser comparison.

PET: Native HPSG Parsing The parser most commonly used with the ERG is called PET (Callmeier, 2002), a highly engineered chart parser for unification grammars. PET constructs a complete parse forest, using subsumption-based ambiguity factoring (Oepen & Carroll, 2000), and then extracts from the forest n-best lists of complete analyses according to a discriminative parse ranking model (Zhang et al., 2007). For our experiments, we trained the parse ranker on Sections 00–19 of DeepBank and otherwise used the default configuration (which corresponds to the environment used by the DeepBank and Redwoods developers), which is optimized for accuracy. This parser, performing exact inference, we will call ERG_a.

In recent work, Dridan (2013) augments ERG parsing with lattice-based sequence labeling over lexical types and lexical rules. Pruning the parse chart prior to forest construction yields greatly improved efficiency at a moderate accuracy loss. Her lexical pruning model is trained on DeepBank 00–19 too, hence compatible with our setup. We include the best-performing configuration of Dridan (2013) in our experiments, a variant henceforth referred to as ERG_e. Unlike the other parsers in our study, PET internally operates over an ambiguous token lattice, and there is no easy interface to feed the parser pre-tokenized inputs. We approximate the effects of gold-standard tokenization by requesting from the parser a 2000-best list, which we filter for the top-ranked analysis whose leaves match the treebank tokenization. This approach is imperfect, as in some cases no token-compatible analysis may be on the n-best list, especially so in the ERG_e setup (where lexical items may have been pruned by the sequence-labeling model). When this happens, we fall back to the top-ranked analysis and adjust our evaluation metrics to robustly deal with tokenization mismatches (see below).

B&N: Direct Dependency Parsing The parser of Bohnet and Nivre (2012), henceforth B&N, is a transition-based *dependency parser* with joint tagger that implements global learning and a beam search for non-projective labeled dependency parsing. This parser consistently outperforms pipeline systems (such as the Malt and MST parsers) both in terms of tagging and parsing accuracy for typologically diverse languages such as Chinese, English, and German. We apply B&N mostly 'out-of-the-box', training on the DT conversion of DeepBank Sections 00–19, and running the parser with an increased beam size of 80.

Berkeley: PCFG Parsing The Berkeley parser (Petrov et al., 2006; henceforth just Berkeley) is a gen-

	Unary Rules Preserved						**Unary Rules Removed**			
Labels	**Long**		**Short**		**Mixed**		**Long**		**Short**	
Cycles	**5**	**6**	**5**	**6**	**5**	**6**	**5**	**6**	**5**	**6**
Gaps	2	5	*0*	*0*	11	19	3	3	*0*	*0*
TA	90.96	90.62	91.11	*91.62*	90.93	90.94	88.46	87.65	89.16	88.46
F_1	76.39	75.66	79.81	*80.33*	76.70	76.74	74.53	73.72	75.15	73.56
LAS	86.26	85.90	82.50	83.15	*86.72*	86.16	83.96	83.20	80.49	79.56
UAS	89.34	88.92	89.80	*90.34*	89.42	88.84	87.12	86.54	87.95	87.15

Table 2: Tagging accuracy, PARSEVAL F_1, and dependency accuracy for Berkeley on WSJ development data.

erative, unlexicalized *phrase structure* parser that automatically derives a smoothed latent-variable PCFG from the treebank and refines the grammar by a split–merge procedure. The parser achieves state-of-the-art performance on various standard benchmarks. In § 4 below, we explain how we adapt ERG derivations for training and testing with Berkeley; for comparison to the other parsers in terms of DT dependency accuracy, we apply the converter of Ivanova et al. (2012) to Berkeley outputs. For technical reasons, however, the optional mapping from ERG to PTB tokenization is not applicable in this setup, and hence our experiments involving Berkeley are limited to ERG tokens and fine-grained lexical categories.

Evaluation Standard evaluation metrics in dependency parsing are labeled and unlabeled attachment scores (LAS, UAS; implemented by the CoNLL eval.pl scorer). These measure the percentage of tokens which are correctly attached to their head token and, for LAS, have the right dependency label. As assignment of lexical categories is a core part of syntactic analysis, we complement LAS and UAS with tagging accuracy scores (TA), where appropriate. However, in our work there are two complications to consider when using eval.pl. First, some of our parsers occasionally fail to return any analysis, notably Berkeley and ERG_e. For these inputs, our evaluation re-inserts the missing tokens in the parser output, padding with dummy 'placeholder' heads and dependency labels.

Second, a more difficult issue is caused by occassional tokenization mismatches in ERG parses, as discussed above. Since eval.pl identifies tokens by their position in the sentence, any difference of tokenization will lead to invalid results. One option would be to treat all system outputs with token mismatches as parse failures, but this over-penalizes, as potentially correct dependencies among corresponding tokens are also removed from the parser output. For this reason, we modify the evaluation of dependency accuracy to use sub-string character ranges, instead of consecutive identifiers, to encode token identities. This way, tokenization mismatches local to some sub-segment of the input will not 'throw off' token correspondences in other parts of the string.[5] We will refer to this character-based variant of the standard CoNLL metrics as LAS_c and UAS_c.

4 PCFG Parsing of HPSG Derivations

Formally, the HPSG analyses in the DeepBank and Redwoods treebanks transcend the class of context-free grammars, of course. Nevertheless, one can pragmatically look at an ERG derivation as if it were a context-free phrase structure tree. On this view, standard, off-the-shelf PCFG parsing techniques are applicable to the ERG treebanks. Zhang and Krieger (2011) explore this space experimentally, combining the ERG, Redwoods (but not DeepBank), and massive collections of automatically parsed text. Their study, however, does not consider parser efficiency.[6].

In contrast, our goal is to reflect on practical trade-offs along multiple dimensions. We therefore focus on Berkeley, as one of the currently best-performing (and relatively efficient) PCFG engines. Due to its ability to internally rewrite node labels, this parser should be expected to adapt well also to ERG derivations. Compared to the phrase structure annotations in the PTB, there are two structural differences evident in Figure 1. First, the inventories of phrasal and lexical labels are larger, at around 250 and 1000, respectively, compared to only about two dozen phrasal categories and 45 parts of speech in the PTB. Second, ERG derivations contain more unary (non-branching)

[5]Where tokenization is identical for the gold and system outputs, the score given by this generalized metric is exactly the same as that of eval.pl. Unless indicated otherwise, punctuation marks are included in scoring.

[6]Their best PCFG results are only a few points F_1 below the full HPSG parser, using massive PCFGs and exact inference; parsing times in fact exceed those of the native HPSG parser

	Gaps	Time	TA$_c$	LAS$_c$	UAS$_c$
Berkeley	1+0	*1.0*	92.9	86.65	89.86
B&N	0+0	1.7	92.9	86.76	89.65
ERG$_a$	0+0	10	*97.8*	*92.87*	*93.95*
ERG$_e$	13+44	1.8	96.4	91.60	92.72

Table 3: Parse failures and token mismatches ('gaps'), efficiency, and tagging and dependency accuracy on WSJ.

rules, recording for example morphological variation or syntacto-semantic category changes.[7]

Table 2 summarizes a first series of experiments, seeking to tune the Berkeley parser for maximum accuracy on our development set, DeepBank Section 20. We experimented with preserving unary rules in ERG derivations or removing them (as they make no difference to the final DT analysis); we further ran experiments using the native ('long') ERG construction identifiers, their generalizations to 'short' labels as used in DT, and a variant with long labels for unary and short ones for branching rules ('mixed'). We report results for training with five or six split–merge cycles, where fewer iterations generally showed inferior accuracy, and larger values led to more parse failures ('gaps' in Table 2). There are some noticeable trade-offs across tagging accuracy, dependency accuracy, and coverage, without a single best performer along all three dimensions. As our primary interest across parsers is dependency accuracy, we select the configuration with unary rules and long labels, trained with five split–merge cycles, which seems to afford near-premium LAS at near-perfect coverage.[8]

5 In-Domain Results

Our first cross-paradigm comparison of the three parsers is against the WSJ in-domain test data, as summarized in Table 3. There are substantive differences between parsers both in terms of coverage, speed, and accuracy. Berkeley fails to return an analysis for one input, whereas ERG$_e$ cannot parse 13 sentences (close to one percent of the test set); just as the 44 inputs where parser output deviates in tokenization from the treebank, this is likely an effect of the lexical pruning applied in this setup. At an average of one second per input, Berkeley is the fastest of our parsers; ERG$_a$ is exactly one order of magnitude slower. However, the lexical pruning of Dridan (2013) in ERG$_e$ leads to a speed-up of almost a factor of six, making this variant of PET perform comparable to B&N. Maybe the strongest differences, however, we observe in tagging and dependency accuracies: The two data-driven parsers perform very similarly (at close to 93% TA and around 86.7% LAS); the two ERG parsers are comparable too, but at accuracy levels that are four to six points higher in both TA and LAS. Compared to ERG$_a$, the faster ERG$_e$ variant performs very slightly worse—which likely reflects penalization for missing coverage and token mismatches—but it nevertheless delivers much higher accuracy than the data-driven parsers. In subsequent experiments, we will thus focus only on ERG$_e$.

6 Error Analysis

The ERG parsers outperform the two data-driven parsers on the WSJ data. Through in-depth error analysis, we seek to identify parser-specific properties that can explain the observed differences. In the following, we look at (a) the accuracy of individual dependency types, (b) dependency accuracy relative to (predicted and gold) dependency length, and (c) the distribution of LAS over different lexical categories.

Among the different dependency types, we observe that the notion of an adjunct is difficult for all three parsers. One of the hardest dependency labels is hdn-aj (post-adjunction to a nominal head), the relation employed for relative clauses and prepositional phrases attaching to a nominal head. The most common error for this relation is verbal attachment.

It has been noted that dependency parsers may exhibit systematic performance differences with respect to dependency length (i.e. the distance between a head and its argument; McDonald & Nivre, 2007). In our experiments, we find that the parsers perform comparably on longer dependency arcs (upwards of fifteen words), with ERG$_a$ constantly showing the highest accuracy, and Berkeley holding a slight edge over B&N as dependency length increases.

In Figure 3, one can eyeball accuracy levels per lexical category, where conjunctions (c) and various types of prepositions (p and pp) are the most difficult for all three parsers. That the DT analysis of coordination is challenging is unsurprising. Schwartz et al.

[7]Examples of morphological rules in Figure 1 include v_pas_odlr and v_n3s-bse_ilr, for past-participle and non-third person singular or base inflection, respectively. Also, there are two instances of bare noun phrase formation: hdn_bnp-pn_c and hdn_bnp-qnt_c.

[8]A welcome side-effect of this choice is that we end up using native ERG derivations without modifications.

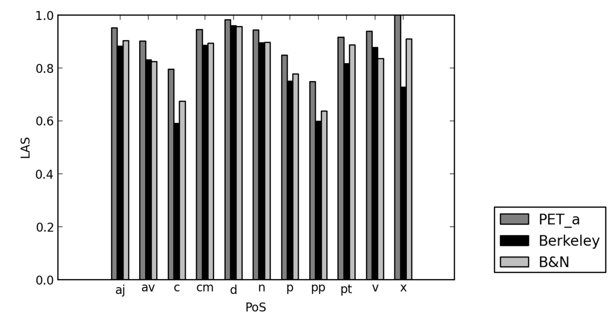


Figure 3: WSJ per-category dependency accuracies on coarse lexical head categories: adjective, adverb, conjunction, complementizer, determiner, noun, preposition, lexical prepositional phrase, punctuation, verb, and others.

(2012) show that choosing conjunctions as heads in coordinate structures is harder to parse for direct dependency parsers (while this analysis also is linguistically more expressive). Our results confirm this effect also for the PCFG and (though to a lesser degree) for ERG$_a$. At the same time, conjunctions are among the lexical categories for which ERG$_a$ most clearly outperforms the other parsers. Berkeley and B&N exhibit LAS error rates of around 35–41% for conjunctions, whereas the ERG$_a$ error rate is below 20%. For many of the coordinate structures parsed correctly by ERG$_a$ but not the other two, we found that attachment to root constitutes the most frequent error type—indicating that clausal coordination is particularly difficult for the data-driven parsers.

The attachment of prepositions constitutes a notorious difficulty in syntactic analysis. Unlike 'standard' PoS tag sets, ERG lexical types provide a more fine-grained analysis of prepositions, for example recognizing a lexicalized PP like *in full*, or making explicit the distinction between semantically contenful vs. vacuous prepositions. In our error analysis, we find that parser performance across the various prepositional sub-types varies a lot. For some prepositions, all parsers perform comparatively well; e.g. p_np_ptcl-of_le, for semantically vacuous *of*, ranks among the twenty most accurate lexical categories across the board. Other types of prepositions are among the categories exhibiting the highest error rates, e.g. p_np_i_le for 'common' prepositions, taking an NP argument and projecting intersective modifier semantics. Even so, Figure 3 shows that the attachment of prepositions (p and pp) is an area where ERG$_a$ excels most markedly. Three frequent prepositional lexical types that show the largest ERG$_a$ advantages are p_np_ptcl-of_le (*history <u>of</u> Linux*), p_np_ptcl_le (*look <u>for</u> peace*), and p_np_i_le (*talk <u>about</u> friends*). Looking more closely at inputs where the parsers disagree, they largely involve (usages of) prepositions which are lexically selected for by their head. In other words, most prepositions in isolation are ambiguous lexical items. However, it appears that lexical information about the argument structure of heads encoded in the grammar allows ERG$_a$ to analyse these prepositions (in context) much more accurately.

		Gaps	TA$_c$	LAS$_c$	UAS$_c$
CB	Berkeley	1+0	87.1	78.13	83.14
	B&N	0+0	87.7	77.70	82.96
	ERG$_e$	8+8	95.3	90.02	91.58
SC	Berkeley	1+0	87.2	79.81	85.10
	B&N	0+0	85.9	78.08	83.21
	ERG$_e$	11+7	94.9	89.94	91.26
VM	Berkeley	7+0	84.0	74.40	83.38
	B&N	0+0	83.1	75.28	82.86
	ERG$_e$	11+42	94.4	90.18	91.75
WS	Berkeley	7+0	87.7	80.31	85.09
	B&N	0+0	88.4	80.63	85.24
	ERG$_e$	4+12	96.9	90.64	91.76

Table 4: Cross-domain coverage (parse failures and token mismatches) and tagging and dependency accuracies.

7 Cross-Domain Results

To gauge the resilience of the different systems to domain and genre variation, we applied the same set of parsers—without re-training or other adaptation—to the additional Redwoods test data. Table 4 summarizes coverage and accuracy results across the four diverse samples. Again, Berkeley and B&N pattern alike, with Berkeley maybe slightly ahead in terms of dependency accuracy, but penalized on two of the test sets for parse failures. LAS for the two data-driven parsers ranges between 74% and 81%, up to 12 points below their WSJ performance. Though large, accuracy drops on a similar scale have been observed repeatedly for purely statistical systems when moving out of the WSJ domain without adaptation (Gildea, 2001; Nivre et al., 2007). In contrast, ERG$_e$ performance is more similar to WSJ results, with a maximum LAS drop of less than two points.[9] For

[9]It must be noted that, unlike the WSJ test data, some of these cross-domain data sets have been used in ERG development throughout the years, notably VM and CB, and thus the grammar is likely to have particularly good linguistic coverage of this data.

		Gaps	Lexical Types LAS$_c$	Lexical Types UAS$_c$	PTB PoS Tags LAS$_c$	PTB PoS Tags UAS$_c$
WSJ	B&N	0+0	88.78	91.52	91.56	93.63
	ERG$_e$	13+9	92.38	93.53	92.38	93.53
CB	B&N	0+0	81.56	86.18	84.54	88.53
	ERG$_e$	8+4	90.77	92.21	90.77	92.21
SC	B&N	0+0	81.69	86.11	85.17	88.85
	ERG$_e$	11+0	90.13	91.86	90.13	91.86
VM	B&N	0+0	77.00	83.73	82.76	88.11
	ERG$_e$	10+0	91.55	93.08	91.55	93.08
WS	B&N	0+0	82.09	86.17	84.59	88.41
	ERG$_e$	4+0	91.61	92.62	91.61	92.62

Table 5: Coverage and dependency accuracies with PTB tokenization and either detailed or coarse lexical categories.

Wikipedia text (WS; previously unseen data for the ERG, just as for the other two), for example, both tagging and dependency accuracies are around ten points higher, an error reduction of more than 50%. From these results, it is evident that the general linguistic knowledge available in ERG parsing makes it far more resilient to variation in domain and text type.

8 Sanity: PTB Tokenization and PoS Tags

Up to this point, we have applied the two data-driven parsers in a setup that one might consider somewhat 'off-road'; although our experiments are on English, they involve unusual tokenization and lexical categories. For example, the ERG treatment of punctuation as 'pseudo-affixes' increases vocabulary size, which PET may be better equipped to handle due to its integrated treatment of morphological variation. In two concluding experiments, we seek to isolate the effects of tokenization conventions and granularity of lexical categories, taking advantage of optional output flexibility in the DT converter of Ivanova et al. (2012).[10] Table 5 confirms that tokenization does make a difference. In combination with fine-grained lexical categories still, B&N obtains LAS gains of two to three points, compared to smaller gains (around or below one point) for ERG$_e$.[11] However, in this setup our two earlier observations still hold true: ERG$_e$ is substantially more accurate within the WSJ domain and far more resilient to domain and genre variation. When we simplify the syntactic analysis task and train and test B&N on coarse-grained PTB PoS tags only, in-domain differences between the two parsers are further reduced (to 0.8 points), but ERG$_e$ still delivers an error reduction of ten percent compared to B&N. The picture in the cross-domain comparison is not qualitatively different, also in this simpler parsing task, with ERG$_e$ maintaining accuracy levels comparable to WSJ, while B&N accuracies degrade markedly.

Conversely, SC has hardly had a role in grammar engineering so far, and WS is genuinely unseen (for the current ERG and Redwoods release), i.e. treebankers were first exposed to it once the grammar and parser were frozen.

[10]As mapping from ERG derivations into PTB-style tokens and PoS tags is applied when converting to bi-lexical dependencies, we cannot easily include Berkeley in these final experiments.

[11]When converting to PTB-style tokenization, punctuation marks are always attached low in the DT scheme, to the immediately preceding or following token, effectively adding a large group of 'easy' dependencies.

9 Discussion and Conclusion

Our experiments sought to contrast state-of-the-art representatives from three parsing paradigms on the task of producing bi-lexical syntactic dependencies for English. For the HPSG-derived DT scheme, we find that hybrid, grammar-driven parsing yields superior accuracy, both in- and in particular cross-domain, at processing times comparable to the currently best direct dependency parser. These results corroborate the Dutch findings of Plank and van Noord (2010) for English, where more training data is available and in comparison to more advanced data-driven parsers. In most of this work, we have focussed exclusively on parser inputs represented in the DeepBank and Redwoods treebanks, ignoring 15 percent of the original running text, for which the ERG and PET do not make available a gold-standard analysis. While a parser with partial coverage can be useful in some contexts, obviously the data-driven parsers must be credited for providing a syntactic analysis of (almost) all inputs. However, the ERG coverage gap can be straighforwardly addressed by falling back to another parser when necessary. Such a system combination would undoubtedly yield better tagging and dependency accuracies than the data-driven parsers by themselves, especially so in an open-domain setup. A secondary finding from our experiments is that PCFG parsing with Berkeley and conversion to DT dependencies yields equivalent or mildly more accurate analyses, at much greater efficiency. In future work, it would be interesting to include in this comparison other PCFG parsers and linear-time, transition-based dependency parsers, but a tentative generalization over our findings to date is that linguistically richer representations enable more accurate parsing.

Acknowledgments

We are grateful to our colleagues Emily M. Bender, Francis Bond, Rui Wang, and Yi Zhang for many helpful discussions and suggestions, as well as to our three anonymous reviewers for insightful comments. This work is in part funded by the Norwegian Research Council through its WeSearch project. Large-scale experimentation is made possible through access to the ABEL high-performance computing facilities at the University of Oslo, and we are grateful to the Scientific Computing staff at UiO, as well as to the Norwegian Metacenter for Computational Science, and the Norwegian tax payer.

References

Adolphs, P., Oepen, S., Callmeier, U., Crysmann, B., Flickinger, D., & Kiefer, B. (2008). Some fine points of hybrid natural language parsing. In *Proceedings of the 6th International Conference on Language Resources and Evaluation.* Marrakech, Morocco.

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., ... Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the workshop on speech and natural language* (p. 306 – 311). Pacific Grove, USA.

Bohnet, B., & Nivre, J. (2012). A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning* (p. 1455 – 1465). Jeju Island, Korea.

Bos, J., et al. (Eds.). (2008). *Workshop on cross-framework and cross-domain parser evaluation.* Manchester, UK.

Briscoe, T., & Carroll, J. (2006). Evaluating the accuracy of an unlexicalised statistical parser on the PARC DepBank. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics* (p. 41 – 48). Sydney, Australia.

Callmeier, U. (2002). Preprocessing and encoding techniques in PET. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. A case study in efficient grammar-based processing* (p. 127 – 140). Stanford, CA: CSLI Publications.

Carter, D. (1997). The TreeBanker. A tool for supervised training of parsed corpora. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering* (p. 9 – 15). Madrid, Spain.

Cer, D., de Marneffe, M.-C., Jurafsky, D., & Manning, C. (2010). Parsing to Stanford Dependencies. Trade-offs between speed and accuracy. In *Proceedings of the 7th International Conference on Language Resources and Evaluation* (p. 1628 – 1632). Valletta, Malta.

Clark, S., & Curran, J. R. (2007). Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Meeting of the Association for Computational Linguistics* (p. 248 – 255). Prague, Czech Republic.

de Marneffe, M.-C., & Manning, C. D. (2008). The Stanford typed dependencies representation. In *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation* (p. 1 – 8). Manchester, UK.

Dridan, R. (2013). Ubertagging. Joint segmentation and supertagging for English. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (p. 1 – 10). Seattle, WA, USA.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering, 6 (1)*, 15 – 28.

Flickinger, D., Zhang, Y., & Kordoni, V. (2012). DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories* (p. 85 – 96). Lisbon, Portugal: Edições Colibri.

Foster, J., Cetinoglu, O., Wagner, J., Le Roux, J., Nivre, J., Hogan, D., & van Genabith, J. (2011). From news to comment. Resources and benchmarks for parsing the language of Web 2.0. In *Proceedings of the 2011 International Joint Conference on Natural Language Processing* (p. 893 – 901).

Fowler, T. A. D., & Penn, G. (2010). Accurate context-free parsing with Combinatory Categorial Grammar. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics* (p. 335 – 344). Uppsala, Sweden.

Francis, W. N., & Kucera, H. (1982). *Frequency analysis of english usage*. New York: Houghton Mifflin Co.

Gildea, D. (2001). Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing* (p. 167 – 202). Pittsburgh, USA.

Hockenmaier, J., & Steedman, M. (2007). CCGbank. A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, *33*, 355 – 396.

Ivanova, A., Oepen, S., & Øvrelid, L. (2013). Survey on parsing three dependency representations for English. In *Proceedings of the 51th Meeting of the Association for Computational Linguistics* (p. 31 – 37). Sofia, Bulgaria.

Ivanova, A., Oepen, S., Øvrelid, L., & Flickinger, D. (2012). Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proceedings of the sixth linguistic annotation workshop* (p. 2 – 11). Jeju, Republic of Korea.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M., & Kaplan, R. M. (2003). The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora* (p. 1 – 8). Budapest, Hungary.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpora of English: The Penn Treebank. *Computational Linguistics*, *19*, 313 – 330.

McDonald, R. T., & Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning* (p. 122 – 131). Prague, Czech Republic.

Miyao, Y., Sagae, K., & Tsujii, J. (2007). Towards framework-independent evaluation of deep linguistic parsers. In *Proceedings of the 2007 Workshop on Grammar Engineering across Frameworks* (p. 238 – 258). Palo Alto, California.

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., & Yuret, D. (2007). The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Natural Language Learning* (p. 915 – 932). Prague, Czech Republic.

Oepen, S., & Carroll, J. (2000). Ambiguity packing in constraint-based parsing. Practical results. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics* (p. 162 – 169). Seattle, WA, USA.

Oepen, S., Flickinger, D., Toutanova, K., & Manning, C. D. (2004). LinGO Redwoods. A rich and dynamic treebank for HPSG. *Research on Language and Computation*, *2*(4), 575 – 596.

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics* (p. 433 – 440). Sydney, Australia.

Plank, B., & van Noord, G. (2010). Grammar-driven versus data-driven. Which parsing system is more affected by domain shifts? In *Proceedings of the 2010 Workshop on NLP and Linguistics: Finding the common ground* (p. 25 – 33). Uppsala, Sweden: Association for Computational Linguistics.

Pollard, C., & Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar.* Chicago, USA: The University of Chicago Press.

Schwartz, R., Abend, O., & Rappoport, A. (2012). Learnability-based syntactic annotation design. In *Proceedings of the 24th International Conference on Computational Linguistics.* Mumbai, India.

Wahlster, W. (Ed.). (2000). *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed.). Berlin, Germany: Springer.

Ytrestøl, G., Oepen, S., & Flickinger, D. (2009). Extracting and annotating Wikipedia sub-domains. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories* (p. 185 – 197). Groningen, The Netherlands.

Zhang, Y., & Krieger, H.-U. (2011). Large-scale corpus-driven PCFG approximation of an HPSG. In *Proceedings of the 12th International Conference on Parsing Technologies* (p. 198 – 208). Dublin, Ireland.

Zhang, Y., Oepen, S., & Carroll, J. (2007). Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies* (p. 48 – 59). Prague, Czech Republic.

Zhang, Y., & Wang, R. (2009). Cross-domain dependency parsing using a deep linguistic grammar. In *Proceedings of the 47th Meeting of the Association for Computational Linguistics* (p. 378 – 386). Suntec, Singapore.

Generalization of Words for Chinese Dependency Parsing

Xianchao Wu, Jie Zhou, Yu Sun, Zhanyi Liu, Dianhai Yu, Hua Wu, Haifeng Wang
Baidu Inc.
{wuxianchao,zhoujie01,sunyu02,liuzhanyi,yudianhai,wu_hua,wanghaifeng}@baidu.com

Abstract

In this paper, we investigate the influence of generalization of words to the accuracies of Chinese dependency parsing. Specially, in our shift-reduce parser, we use a neural language model based word embedding (NLMWE) method (Bengio et al., 2003) to generate *distributed word feature vectors* and then perform K-means based word clustering to generate word classes. We designed feature templates by making use of words, part-of-speech (POS) tags, coarse-grained POS (CPOS) tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags, especially when POS-tags are automatically generated. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLMWE-based word classes yield the best unlabelled attachment scores (UASs). Our final UAS$-p$ (excluding punctuations) of 86.79% on the CTB5 test set is comparable to state-of-the-art results. Our final UAS$-p$ of 86.80% and 87.05% on the CTB7 Stanford dependency test set and original test set is significantly better than three well known open-source dependency parsers.

1 Introduction

Current dependency parsing framework is facing the following challenge, training the model using manually annotated treebanks and then apply the model to the whole Web. In terms of words, it is not possible for the treebank to cover all the words in the Web. Given a test sentence, how can we expect the parser to output a correct tree if there are out-of-vocabulary (OOV) words (compared to the training data of the treebank) and/or the POS-tags are wrongly annotated?

Words need to be *generalized* to solve this problem in a sense. Indeed, POS-tag itself is a way to generalize words into word classes. This is because POS-taggers can be trained on larger-scale data compared with treebanks. Annotating trees is far more difficult than annotating POS-tags. Considering that unsupervised word clustering methods can make use of TB/PB-level Web data, these approaches have been shown to be helpful for dependency parsing (Koo et al., 2008).

In this paper, we investigate the influence of generalization of words to the accuracies of Chinese dependency parsing. Specially, in our shift-reduce parser, we use a neural language model based word embedding method (Bengio et al., 2003) to generate *distributed word feature vectors* and then perform K-means based word clustering (Yu et al., 2013) to generate word classes. Our usage of word embedding is in line with Turian et al. (2010) and Yu et al. (2013), who study the effects of different clustering algorithms for POS tagging and named entity recognition (NER). We designed feature templates by making use of words, POS tags, CPOS tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLMWE-based word classes yield the best UASs.

1.1 Shift-reduce parsing

We use a transition-based shift-reduce parser (Kudo and Matsumoto, 2002; Nivre, 2003; Nivre et al., 2006; Huang and Sagae, 2010) to perform all the experiments in this paper. In a typical transition-based parsing process, the input words are stored in a queue and partially built dependency structures (e.g., sub-trees) are organized by a configuration (or state). A parser configuration (or, state) can be represented by a tuple $< S, N, A >$, where S is the stack, N is the queue of incoming words, and A is the set of dependency arcs that have been built. A set of shift-reduce actions are defined, which are used to construct new dependency arcs by connecting the top word of the queue and the top word of the stack. We adopt the arc-standard system (Nivre, 2008), whose actions include:

- *shift*, which removes the top word in the queue and pushes it onto the top of the stack;
- *left-arc*, which pops the top item off the stack, and adds it as a modifier to the front of the queue;
- *right-arc*, which removes the front of the queue, and adds it as a modifier to the top of the stack. In addition, the top of the stack is popped and added to the front of the queue.

We follow Kudo and Matsumoto (2002) and use the Support Vector Machines (SVMs) for action classification training and beam search (Zhang and Clark, 2008) for decoding.

2 Neural Language Model Based Word Embedding

Following (Bengio et al., 2003), we use a neural network with two hidden layers to learn *distributed word feature vectors* from large-scale training data. Recall that, the goal of statistical language modelling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the *curse of dimensionality*: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training (so called OOV words and/or sequences). N-gram based approach obtains generalization by concatenating very short overlapping sequences seen in the training set. Bengio et al. (2003) propose to fight the curse of dimensionality by learning a *distributed* representation (of feature vectors) for words which allows each training sentence to inform the model about an exponential number of semantically neighbouring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations.

The general process of neural language model based word embedding is as follows:

- associate with each word in the vocabulary a distributed word feature vector (a real valued vector in R^m);
- express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence; and,
- learn simultaneously the word feature vectors and the parameters of that probability function.

The training set of the NLMWE model is a sequence $w_1, ..., w_T$ of words $w_t \in V$, where the vocabulary V is a large yet finite set. The objective is to learn a model $f(w_t, ..., w_{t-n+1}) = P(w_t|w_1^{t-1})$, so that that the model gives high out-of-sample likelihood. The only constraint on the model is that for any choice of w_1^{t-1}, $\sum_{i=1}^{|V|} f(i, w_{t-1}, ..., w_{t-n+1}) = 1$ with $f > 0$. By the product of these conditional probabilities, one obtains a model of the joint probability of sequences of words.

Function $f(w_t, ..., w_{t-n+1}) = P(w_t|w_1^{t-1})$ is decomposed into two parts:

- A mapping C from any element i of V to a real vector $C(i) \in R^m$. It represents the *distributed feature vectors* associated with each word in the vocabulary. In practice, C is represented by a $|V| \times m$ matrix of free parameters.
- The probability function over words, expressed with C: a function g maps an input sequence of feature vectors for words in context, $(C(w_{t-n+1}), ..., C(w_{t-1}))$, to a conditional probability distribution over words in V for the next word w_t . The output of g is a vector whose i-th element estimates the probability $P(w_t = i|w_1^{t-1})$ as in Figure 1 in (Bengio et al., 2003).

Following (Bengio et al., 2003), in the real implementation, we speed up with both parallel data and parallel parameter estimation. We finally use the well-known K-means clustering algorithm based on the distributed word feature vectors for word clustering. We separately set the number of word classes to be 100, 500, and 1,000 in our experiments.

3 Feature Templates

At each step during shift-reducing, a parser configuration (or, state) can be represented by a tuple $< S, N, A >$. We denote the top of stack with S_0, the front items from the queue with N_0, N_1, N_2, and N_3, the leftmost and rightmost modifiers of S_0 (if any) with S_{0l} and S_{0r}, respectively, and the leftmost modifier of N_0 (if any) with N_{0l} (refer to Figure 1). The baseline feature templates without any word class level information (such as POS-tags) are shown in Table 1. These features are mostly taken from Zhang and Clark (2008), Huang and Sagae (2010), and Zhang and Nivre (2011). In this table, w, l and d represents the word, dependency label, and the distance between S_0 and N_0, respectively. For example, S_0wN_0w represents the feature template that takes the word of S_0, and combines it with the word of N_0.

In Table 1, $(S/N)_{0l2}$, $(S/N)_{0r2}$, and $(S/N)_{0rn}$ refer to the second leftmost modifier, the second rightmost modifier, and the right nearest modifier of $(S/N)_0$, respectively. It should be mentioned that, for the arc-standard algorithm used in this paper, S_0 or N_0 never contain a head word. The reason is that, once the head is found for a node, that node will be hidden under the head and being removed from the stack or the queue[1].

Table 2 lists the features templates that are related to POS-tags and their combination with words and dependency labels. In the table, $(S/N)_{0ll}$ and $(S/N)_{0rr}$ stand for the left(ll)/right(rr)-hand-side neighbour word of S_0/N_0 in the input sentence for training or decoding. For example, features such as POS-tags of *bi-siblings* of N_0 (e.g., $N_{0l2}pN_{0r2}p$) and S_0 (e.g., $S_{0r2}pS_{0r}p$) are included in the combination of two feature templates. These *bi-sibling* features together with $(S/N)_0$ (e.g., $N_0N_{0l2}pN_{0r2}p$ and $S_0S_{0r2}pS_{0r}p$) are included in the combination of three feature templates. N_0 (similar with S_0) and its surrounding words, such as neighbour words and child words are shown in Figure 1 for intuitive understanding. For comparison with other levels of word classes, we will directly replace POS-tags (p) by other kind of word classes, such as CPOS-tags and NLMWE-based K-means word clusterings. For example, instead of returning the POS-tag of S_0, S_0p will return CPOS-tag of the word of S_0 in CPOS-tag related feature templates, and return NLMWE-based word class index of the word of S_0 in NLMWE clustering related feature templates.

single feature templates (14)
$N_{0l}w$, N_0w, N_1w, N_2w, $S_{0l2}l$, $S_{0l2}w$, $S_{0l}l$, $S_{0l}w$, $S_{0r2}l$, $S_{0r2}w$, $S_{0rn}w$, $S_{0r}l$, $S_{0r}w$, S_0w
combination of two feature templates (19)
$N_{0l}wS_0w$, $N_{0rn}wS_0w$, $N_{0r}wS_0w$, $N_0wN_{0l}l$, N_0wN_1w, N_0wN_2w, N_0wN_3w, N_0wd, N_1wN_2w, N_1wN_3w, $N_1wS_{0r}l$, N_2wN_3w, $S_{0l}wN_0w$, $S_{0rn}wN_0w$, $S_{0r}wN_0w$, S_0wN_0w, $S_0wS_{0l}l$, $S_0wS_{0r}l$, S_0wd
combination of three feature templates (8)
$N_0wN_{0l}lN_{0l2}l$, $N_0wN_1wN_2w$, $N_0wN_1wN_3w$, $N_0wN_2wN_3w$, $N_1wN_2wN_3w$, S_0wN_0wd, $S_0wS_{0l}lS_{0l2}l$, $S_0wS_{0r}lS_{0r2}l$
combination of four feature templates (1)
$N_0wN_1wN_2wN_3w$

Table 1: Feature templates related to words (w), dependency labels (l) and the distance between S_0 and N_0 (d).

combination of two feature templates (3)
S_1wS_1p, S_0wS_1w, S_0pS_1p
combination of three feature templates (13)
$S_0wS_0pS_1p$, $S_0pS_1wS_1p$, $S_0wS_1wS_1p$, $S_0wS_0pS_1w$, $S_1pS_0pN_0p$, $S_1pS_0wN_0p$, $S_1pS_{1l}pS_0p$, $S_1pS_{1r}pS_0p$, $S_1pS_0pS_{0r}p$, $S_1pS_{1l}pS_0p$, $S_1pS_{1r}pS_0w$, $S_1pS_0wS_{0l}p$, $S_2pS_1pS_0p$
combination of four feature templates (1)
$S_0wS_0pS_1wS_1p$

Table 3: Feature templates related to S_1 and S_2.

Besides Table 1 and 2, we further include S_0 and S_1 related features[2] following (Huang and Sagae, 2010). These features are listed in Table 3. As will be shown in Table 16, UASs are improved around 0.2% after appending these feature templates to Table 1 and 2.

4 Experiments

4.1 Setup

We use three Chinese treebanks in our experiments. The first one is an in-house Chinese Query treebank with 12,028 sentences (averagely 4.04 words per sen-

[1]Thanks one reviewer for pointing this out.

[2]Thanks one reviewer for pointing this out.

single feature templates (13)
$N_{0l2}p$, $N_{0l}p$, N_0p, N_1p, N_2p, N_4p, $S_{0l2}p$, $S_{0l}p$, $S_{0r2}p$, $S_{0r}p$, S_0p, S_1p, S_2p
combination of two feature templates (54)
$N_{0l2}pN_{0r2}p$, $N_{0l2}pN_{0rn}p$, $N_{0l2}pN_{0r}p$, $N_{0l}pN_{0l2}p$, $N_{0l}pN_{0r2}p$, $N_{0l}pN_{0rn}p$, $N_{0l}pN_{0r}p$, $N_{0l}pS_0p$, $N_{0l}pS_0w$, $N_{0l}wS_0p$, $N_{0r2}pN_{0r}p$, $N_{0rn}pN_{0r2}p$, $N_{0rn}pN_{0r}p$, $N_{0rn}pS_0p$, $N_{0rn}pS_0w$, $N_{0rn}wS_0p$, $N_{0r}pS_0p$, $N_{0r}pS_0w$, $N_{0r}wS_0p$, $N_0pN_{0l}l$, N_0pN_1p, N_0pN_2p, N_0pN_3p, N_0pd, N_0wN_0p, N_1pN_2p, N_1pN_3p, N_1pS_0w, N_1wN_1p, N_2pN_3p, $S_{0l2}pS_{0r2}p$, $S_{0l2}pS_{0rn}p$, $S_{0l2}pS_{0r}p$, $S_{0l}pN_0p$, $S_{0l}pN_0w$, $S_{0l}pS_{0l2}p$, $S_{0l}pS_{0r2}p$, $S_{0l}pS_{0rn}p$, $S_{0l}pS_{0r}p$, $S_{0l}wN_0p$, $S_{0r2}pS_{0r}p$, $S_{0rn}pN_0p$, $S_{0rn}pN_0w$, $S_{0rn}pS_{0r2}p$, $S_{0rn}pS_{0r}p$, $S_{0rn}wN_0p$, $S_{0r}pN_0p$, $S_{0r}pN_0w$, $S_{0r}wN_0p$, S_0pN_0p, $S_0pS_{0l}l$, $S_0pS_{0r}l$, S_0pd, S_0wS_0p
combination of three feature templates (43)
$N_{0ll}pN_0pS_0p$, $N_{0ll}pS_0pS_{0rr}p$, $N_0pN_{0l2}pN_{0r2}p$, $N_0pN_{0l2}pN_{0rn}p$, $N_0pN_{0l2}pN_{0r}p$, $N_0pN_{0l}lN_{0l2}l$, $N_0pN_{0l}pN_{0l2}p$, $N_0pN_{0l}pN_{0r2}p$, $N_0pN_{0l}pN_{0rn}p$, $N_0pN_{0l}pN_{0r}p$, $N_0pN_{0r2}pN_{0r}p$, $N_0pN_{0rn}pN_{0r2}p$, $N_0pN_{0rn}pN_{0r}p$, $N_0pN_{0rn}pS_0p$, $N_0pN_{0rr}pS_0p$, $N_0pN_{0r}pS_0p$, $N_0pN_1pN_2p$, $N_0pN_1pN_3p$, $N_0pN_2pN_3p$, $N_0pS_{0ll}pS_0p$, $N_0pS_0pS_{0rr}p$, $N_1pN_2pN_3p$, $S_0pN_0pN_{0l}p$, $S_0pN_0pN_1p$, S_0pN_0pd, $S_0pN_0wN_0p$, $S_0pS_{0l2}pS_{0r2}p$, $S_0pS_{0l2}pS_{0rn}p$, $S_0pS_{0l2}pS_{0r}p$, $S_0pS_{0l}lS_{0l2}l$, $S_0pS_{0l}pN_0p$, $S_0pS_{0l}pS_{0l2}p$, $S_0pS_{0l}pS_{0r2}p$, $S_0pS_{0l}pS_{0rn}p$, $S_0pS_{0l}pS_{0r}p$, $S_0pS_{0rn}pN_0p$, $S_0pS_{0rn}pS_{0r2}p$, $S_0pS_{0rn}pS_{0r}p$, $S_0pS_{0r}lS_{0r2}l$, $S_0pS_{0r}pS_{0r2}p$, $S_0wN_0wN_0p$, $S_0wS_0pN_0p$, $S_0wS_0pN_0w$
combination of four feature templates (6)
$N_{0ll}pN_0pS_{0ll}pS_0p$, $N_{0ll}pN_0pS_0pS_{0rr}p$, $N_0pN_{0rr}pS_{0ll}pS_0p$, $N_0pN_{0rr}pS_0pS_{0rr}p$, $N_0pN_1pN_2pN_3p$, $S_0wS_0pN_0wN_0p$

Table 2: Feature templates related to POS-tags (p) and their combination with words (w) and dependency labels (l).

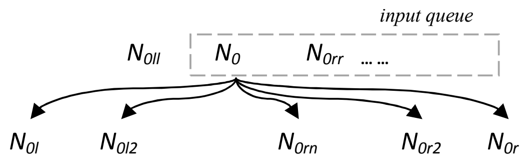


Figure 1: N_0, its neighbour words (N_{0ll} and N_{0rr}), and child words (N_{0l}, N_{0l2}, N_{0rn}, N_{0r2}, and N_{0r}).

tence) for training and 500 sentences (averagely 3.95 words per sentence) for testing.

	Sections	Sentences	Tokens
Train	001-815;1001-1136	16,118	437,859
Dev	886-931;1148-1151	804	20,453
Test	816-885;1137-1147	1,915	50,319

Table 4: Standard split of CTB5 data.

The second one is the Chinese Treebank 5.0[3] (CTB5) (Xue et al., 2007). We follow the standard split of this treebank, as shown in Table 4. The development set is used to tune the hyper-parameter in the SVM classification model for predicting the next sift-reduce actions. We report the unlabelled attachment scores (UASs) of the test set with and without punctuations. We use Penn2Malt toolkit[4] with Chinese head rules[5] to convert the PCFG trees into CoNLL-style[6] dependency trees. Besides the head rules originally used in Penn2Malt, we also independently use the head rules expressed in (Zhang and Clark, 2008) for direct comparison with related works (refer to Table 16). In our experiments, it reveals that using of these two kinds of head rules does not bring a significant differences of UASs.

[3]http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId =LDC2005T01

[4]http://stp.lingfil.uu.se/ñivre/research/Penn2Malt.html

[5]http://stp.lingfil.uu.se/ñivre/research/chn_headrules.txt

	Files	Sentences	Tokens
Train	2,083	46,572	1,039,942
Dev	160	2,079	59,955
Test	205	2,796	81,578

Table 5: Statistics of CTB7 data.

The third one is the Chinese Treebank 7.0[7] (CTB7). The statistics is shown in Table 5 by following the standard split of this treebank. Again, the development set is used to tune the hyper-parameter in the SVM classification model. The PCFG tree to CoNLL format conversion is similar to CTB5.

Table 6 shows the coverage rates of training/testing words in our NLMWE word clustering dictionary (with 1,000 word classes and 4,999,890 unique words) and OOV rates in the testing sets of the Query treebank, CTB5 and CTB7.

As shown in Table 7, we manually construct a mapping from POS-tags to CPOS-tags and then apply this

[6]http://ilk.uvt.nl/conll/#dataformat

[7]http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId =LDC2010T07

	Train	Test	
	NLMWE	NLMWE	OOV (in NLMWE)
Query	99.94	99.90	24.08 (99.58)
CTB5	91.60	91.60	6.25 (65.46)
CTB7	91.39	92.11	4.78 (57.03)

Table 6: Coverage rates (%) of training/testing words in NLMWE word clustering dictionary and OOV rates in the testing sets of the Query treebank, CTB5 and CTB7.

POS	CPOS
VA VC VE VV	1
NR NT NN	2
LC	3
PN	4
DT CD OD	5
M	6
AD	7
P	8
CC CS	9
DEC DEG DER DEV AS SP ETC MSP	10
IJ ON LB SB BA JJ FW	11
PU	12
X	13

Table 7: Mapping from POS-tags to CPOS-tags in CTB5 and CTB7.

mapping to CTB5 and CTB7 to obtain CPOS-tags for each word. For the Query treebank with POS-tags of the Peking University Corpus standard, we simply choose the first letter of the POS-tag to be its CPOS-tag.

4.2 Comparison of number of word classes

	wc.1000	wc.500	wc.100
nlmwe	87.13 (84.00)	86.21 (82.32)	85.86 (81.89)
+pos	88.70 (85.89)	87.84 (85.47)	87.28 (83.58)
+cpos	88.60 (85.68)	87.08 (83.79)	88.19 (84.42)
+pos+cpos	89.56 (86.74)	88.24 (85.05)	88.09 (84.84)

Table 8: The influence of the number of word classes to UAS$-p$ (%) of words and OOV words (numbers in the brackets) in the Query treebank.

Table 8 shows the influence of the number of word classes (of 100, 500, and 1,000) to UAS$-p$ (without punctuations kept and a beam size of 10) in the Query treebank. We observe that, for NLMWE, POS+NLMWE and POS+CPOS+NLMWE, UAS increases as the number of word classes increases. Here, 1,000 word classes performs the best accuracies.

In particular, in order to investigate the generalization of unknown words, we individually show the UAS of OOV words in the test set as influenced by the number of words classes (UASs in the brackets in Table 8). The absolute value of OOV words' UAS is lower than UAS of all words. Again, the combination of CPOS-tags, POS-tags, and NLMWE clustering with 1,000 classes yields the best result.

4.3 Comparison of POS, CPOS, NLMWE and their combinations

	UAS	UASoov	UASin
words	84.03	80.42	85.18
nlmwe	87.13	84.00	88.12
pos	87.18	85.05	87.85
pos+nlmwe	88.70	85.89	89.59
cpos	86.67	84.21	87.45
cpos+nlmwe	88.60	85.68	89.52
pos+cpos	87.63	85.05	88.45
pos+cpos+nlmwe	**89.56**	**86.74**	**90.45**

Table 9: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in the Query treebank.

Table 9 intuitively shows the UAS scores of using feature templates that are related to POS, CPOS, NLMWE (of 1,000 word classes) and their combinations in the Query treebank. We use "words" to represent the feature templates listed in Table 1, i.e., no additional word generalized feature templates are used. For "pos", "cpos", and "nlmwe", they separately use not only the feature templates listed in Table 2 but also the word-related feature templates listed in Table 1. We observe that:

- when only using surface word related feature templates, the UAS (80.42%) of OOV words is the worst. This tells us that word classes are important for generalization. Referring to Table 1, we argue n-gram features such as $N_0wN_1wN_2w$ and their combinations with dependency labels such as $N_0wN_{0l}l$ contribute for the 80.42% accuracy of UAS;
- respectively appending POS/CPOS/NLMWE related features (Table 2) to words yields significant improvements of UAS;
- the combination of either two of POS, CPOS,

and NLMWE yields a better UAS than the individuals;

- finally, the UAS scores are the best for both OOV words (UASoov) and non-OOV words (UASin) when we combine POS, CPOS and NLMWE together.

	UAS	UASoov	UASin
words	71.93	66.98	72.32
nlmwe	76.98	71.88	77.39
pos	86.49	84.87	86.62
pos+nlmwe	86.41	84.91	86.53
cpos	84.43	82.46	84.59
cpos+nlmwe	84.82	83.64	84.92
pos+cpos	86.56	84.78	86.70
pos+cpos+nlmwe	**86.58**	**84.87**	**86.72**

Table 10: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB5.

	UAS	UASoov	UASin
words	72.71	68.80	72.94
nlmwe	78.03	75.17	78.19
pos	86.47	86.25	86.49
pos+nlmwe	87.01	86.69	87.03
cpos	84.21	83.04	84.28
cpos+nlmwe	85.03	83.81	85.10
pos+cpos	86.81	86.45	86.83
pos+cpos+nlmwe	**87.05**	**86.48**	**87.09**

Table 11: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB7.

	UAS	UASoov	UASin
words	69.76	64.16	70.09
nlmwe	74.56	69.86	74.83
pos	86.18	83.89	86.31
pos+nlmwe	86.36	84.38	86.48
cpos	83.66	81.94	83.76
cpos+nlmwe	84.55	82.35	84.68
pos+cpos	86.53	84.71	86.64
pos+cpos+nlmwe	**86.80**	**84.99**	**86.91**

Table 12: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB7 with Stanford dependency.

Table 10, 11 and 12 intuitively shows the UAS scores of using feature templates that are related to POS, CPOS, NLMWE (of 1,000 word classes) and their combinations in CTB5, CTB7 and CTB7 with Stanford dependency, respectively. Different with the 24.08% OOV rate of the test of the Query treebank, 6.25%/4.78% OOV rates of the test sets of CTB5/CTB7 are smaller. This makes UAS and UASin in these three tables quite close with each other. For NLMWE, we have the following observations:

- in CTB5, NLMWE does not bring a significant improvements after being appended to POS and/or CPOS related feature templates;

- in CTB7, appending NLMWE to POS yields an UAS$-p$ improvement of 0.54% (87.01% - 86.47%), which is significant; appending NLMWE to POS+CPOS further yields an UAS$-p$ improvement of 0.24% (87.05% - 86.81%);

- in CTB7 with Stanford dependency, appending NLMWE to POS yields an UAS$-p$ improvement of 0.18% (86.36% - 86.18%); appending NLMWE to POS+CPOS further yields an UAS$-p$ improvement of 0.27% (86.80% - 86.53%), which is significant;

- in CTB5, CTB7 and CTB7 with Stanford dependency, POS+CPOS+NLMWE yields the best UAS$-p$s.

Recall that we introduce CPOS to generalize POS in a sense. For example, all verb related POS tags of VA, VC, VE, and VV are taken as one CPOS tag. We hope that CPOS tags can capture the dependency grammar in a more generalized level than POS tags. For CPOS, we have the following observations:

- in CTB5, appending CPOS to POS yields a slight UAS$-p$ improvement of 0.07% (86.56% - 86.49%); appending CPOS to POS+NLMWE further yields an UAS$-p$ improvement of 0.17% (86.58% - 86.41%);

- in CTB7, appending CPOS to POS yields a slightly UAS$-p$ improvement of 0.34% (86.81% - 86.47%), which is significant; appending CPOS to POS+NLMWE further yields a slight UAS$-p$ improvement of 0.04% (87.05% - 87.01%);

- in CTB7 with Stanford dependency, appending CPOS to POS yields an UAS$-p$ improvement of 0.35% (86.53% - 86.18%), which is significant; appending CPOS to POS+NLMWE further yields an UAS$-p$ improvement of 0.44% (86.80% - 86.36%), which is also significant.

4.4 Comparison of golden/non-golden POS tags

	Not-Golden	Diff	Same	Golden
words	84.03	89.19	83.94	84.03
nlmwe	87.13	**91.89**	87.04	87.13
pos	87.13	89.19	87.09	87.18
pos+nlmwe	88.80	89.19	88.79	88.70
cpos	86.67	**91.89**	86.57	86.67
cpos+nlmwe	88.65	**91.89**	88.58	88.60
pos+cpos	87.53	89.19	87.50	87.63
pos+cpos+nlmwe	**89.56**	89.19	**89.57**	**89.56**

Table 13: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the Query test set.

	Not-Golden	Diff	Same	Golden
words	71.91	63.75	72.14	71.93
nlmwe	76.96	**68.81**	77.19	76.98
pos	83.78	61.61	84.40	86.49
pos+nlmwe	83.94	62.55	84.54	86.41
cpos	82.80	61.78	83.39	84.43
cpos+nlmwe	83.17	62.90	83.74	84.82
pos+cpos	83.83	60.93	84.48	86.56
pos+cpos+nlmwe	**83.96**	61.95	**84.58**	**86.58**

Table 14: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the CTB5 test set.

	Not-Golden	Diff	Same	Golden
words	72.71	65.46	72.87	72.71
nlmwe	78.03	**69.71**	78.22	78.03
pos	83.82	59.25	84.39	86.47
pos+nlmwe	84.41	59.82	84.97	87.01
cpos	82.61	60.71	83.11	84.21
cpos+nlmwe	83.42	62.23	83.91	85.03
pos+cpos	84.12	57.79	84.72	86.81
pos+cpos+nlmwe	**84.49**	59.13	**85.07**	**87.05**

Table 15: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the CTB7 test set.

We finally replace the golden POS-tags in the test sets by POS-tags generated by a CRF-based POS-tagger. Table 13 (POS precision = 98.1%), 14 (POS precision = 97.7%), and 15 (POS precision = 98.1%) respectively show the UAS$-p$ differences of the Query, CTB5, and CTB7 test sets. In this three tables, "Not-Golden" and "Golden" (denoted as UASnot-golden and UASgolden, hereafter) stand for the UAS$-p$s of not using or using the golden POS-tags; "Diff" and "Same" (denoted as UASdiff and UASsame, hereafter) are the UAS$-p$s of those words whose POS-tags are different or similar with the golden POS-tags.

Specially, it is important for us to check whether NLMWE performs better when POS-tags are wrongly annotated. In all these three test sets, NLMWE's UASdiff performs better than that of POS or CPOS. Also, by combining wrong POS-tags (and/or CPOS-tags) and NLMWE together, UASdiff drops. Most importantly, when we look at UASdiff and compare POS with POS+NLMWE, NLMWE does supply POS (and CPOS). For example, in the CTB5 test set, UASdiff significantly increases from 61.61% to 62.55%; in the CTB7 test set, UASdiff significantly increases from 59.25% to 59.82%. On the other hand, by combining correct POS-tags (and/or CPOS-tags) and NLMWE together, UASsame increases, showing that NLMWE even being a good supplement for correct POS-tags.

Finally, when we check the results of appending NLMWE to POS/CPOS in the combination of UASsame and UASdiff, i.e., UASnot-golden, the increasing of UAS is significant in the Query test set (e.g., from 87.53% of POS+CPOS to 89.56% of POS+CPOS+NLMWE). In CTB5 and CTB7 test sets, we observe that NLMWE does bring increasings yet the increasings are not that significant. Referring back to Table 6, we argue that the low coverage (91-92% which is much lower than 99.9% in the Query treebank) of NLMWE clustering dictionary in the CTB5 and CTB7 response for this tendency.

4.5 Comparison to state-of-the-art

Table 16 shows the comparison of our system with state-of-the-art results on CTB5 and CTB7 test sets. We set the beam size of all our system variants to be 10. Note that most related researches only report their results on the CTB5 test set which makes direct comparison in the CTB7 test set unavailable. In the CTB5 test set, we observe that our final result of using POS+CPOS+NLMWE is comparable to most of related researches.

Besides the original data of CTB5 and CTB7, fol-

System	UAS$_{+p}$	UAS$_{-p}$
	CTB5	
Ours (words)	69.85	71.93
Ours (nlmwe)	75.05	76.98
Ours (pos)	84.65	86.49
Ours (pos+nlmwe)	84.52	86.41
Ours (cpos)	82.50	84.43
Ours (cpos+nlmwe)	82.91	84.82
Ours (pos+cpos)	84.72	86.56
Ours (pos+cpos+nlmwe)	84.70	86.58
Ours (pos+cpos+nlmwe)*	84.62	86.59
Ours (pos+cpos+nlmwe)$^{*,+s1s2}$	**84.88**	**86.79**
(Huang and Sagae, 2010)*	NA	85.2
(Zhang and Nivre, 2011)*	NA	86.0
(Zhang and McDonald, 2012)*	NA	**86.87**
(Li et al., 2012)*	NA	86.55
(Sun and Wan, 2013)	84.65	NA
(Hayashi et al., 2013)*	NA	85.9
(Ma et al., 2013)*	NA	86.33
	CTB7	
Ours (words)	70.57	72.71
Ours (nlmwe)	75.90	78.03
Ours (pos)	84.35	86.47
Ours (pos+nlmwe)	84.85	87.01
Ours (cpos)	82.08	84.21
Ours (cpos+nlmwe)	82.91	85.03
Ours (pos+cpos)	84.66	86.81
Ours (pos+cpos+nlmwe)	**84.89**	**87.05**
Ours (pos+cpos+nlmwe)*	84.84	86.99
	CTB7, Stanford	
Ours (words)$^{+}$	68.14	69.76
Ours (nlmwe)$^{+}$	73.00	74.56
Ours (pos)$^{+}$	84.80	86.18
Ours (pos+nlmwe)$^{+}$	84.97	86.36
Ours (cpos)$^{+}$	82.27	83.66
Ours (cpos+nlmwe)$^{+}$	83.19	84.55
Ours (pos+cpos)$^{+}$	85.11	86.53
Ours (pos+cpos+nlmwe)$^{+}$	**85.43**	**86.80**
MaltParser (libsvm)$^{+}$	NA	78.0
MSTParser (1st-order)$^{+}$	NA	78.9
Mate (2nd-order)$^{+}$	NA	**83.1**

Table 16: Comparison of our system with state-of-the-art results on CTB5/CTB7 test sets. Here, UAS$+p$/$-p$ stands for UAS (%) with/without punctuations taken into computing. * stands for using (Zhang and Clark, 2008)'s head rules. $^{+}$ stands for using (Chang et al., 2009)'s head rules. $^{+s1s2}$ stands for appending S_1 and S_2 related feature templates.

lowing (Che et al., 2012), we further apply the Stanford Chinese Dependency (Chang et al., 2009) to the CTB7's train/dev/test sets and compare the UASs with well known open-source systems. Different from the head rules defined in (Zhang and Clark, 2008) which yield 12 dependency labels, the Stanford Chinese Dependency has its own head rules as described in (Chang et al., 2009) which yields 46 fine-grained dependency labels. We directly used the UASs of these systems reported in (Che et al., 2012). In this test set, we observe that our system is significantly better than open-source systems of MSTParser of version 0.5[8] (McDonald and Pereira, 2006), Mate parser of version 2.0[9] (Bohnet, 2010) (2nd-order MSTParser) and MaltParser of version 1.6.1 with the Arc-Eager algorithm[10] (Nivre et al., 2006).

5 Conclusion

We have investigated the influence of generalization of words to the final accuracies of Chinese shift-reduce dependency parsing. We designed feature templates by making use of words, POS-tags, CPOS-tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags, especially for the automatically generated POS-tags. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLP-WE-based word classes yield the best UASs. Our final UAS$-p$ of 86.79% on the CTB5 test set is comparable to state-of-the-art results. Our final UAS$-p$ of 86.80% and 87.05% on the CTB7 Stanford dependency test set and original test set is significantly better than three well known open-source dependency parsers.

Acknowledgments

The authors thank Wanxiang Che for processing the CTB7 data into Stanford dependency format, and the anonymous reviewers for their feedback that substantially improved this paper.

References

Yushua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bernd Bohnet. 2010. Top accuracy and fast dependency

[8]http://sourceforge.net/projects/mstparser
[9]http://code.google.com/p/mate-tools
[10]http://www.maltparser.org/

parsing is not a contradiction. In *Proceedings of COLING*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Pi-Chuan Chang, Huihsin Tseng, Dan Jurafsky, and Christopher D. Manning. 2009. Discriminative reordering with Chinese grammatical relations features. In *Proceedings of SSST-3*, pages 51–59, Boulder, Colorado, June. Association for Computational Linguistics.

Wanxiang Che, Valentin Spitkovsky, and Ting Liu. 2012. A comparison of chinese parsers for stanford dependencies. In *Proceedings of ACL (Volume 2: Short Papers)*, pages 11–16, Jeju Island, Korea, July. Association for Computational Linguistics.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1(1):139–150.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of Co-NLL*, pages 63–69.

Zhenghua Li, Min Zhang, Wanxiang Che, and Ting Liu. 2012. A separately passive-aggressive training algorithm for joint POS tagging and dependency parsing. In *Proceedings of COLING 2012*, pages 1681–1698, Mumbai, India, December. The COLING 2012 Organizing Committee.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first pos tagging and dependency parsing with beam search. In *Proceedings of ACL (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

Joakim Nivre, Johan Hall, , and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, pages 2216–2219.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Weiwei Sun and Xiaojun Wan. 2013. Data-driven, pcfg-based and pseudo-pcfg-based models for chinese dependency parsing. *Transactions of the Association for Computational Linguistics*, 1(1):301–314.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394, Uppsala, Sweden, July. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2007. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Mo Yu, Tiejun Zhao, Daxiang Dong, Hao Tian, and Dianhai Yu. 2013. Compound embedding features for semi-supervised learning. In *Proceedings of NAACL-HLT*, pages 563–568, Atlanta, Georgia, June. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of EMNLP-CoNLL*, pages 320–331, Jeju Island, Korea, July. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL:HLT*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Dynamic-oracle Transition-based Parsing with Calibrated Probabilistic Output

Yoav Goldberg
Computer Science Department
Bar Ilan University
Ramat Gan, Israel
yoav.goldberg@gmail.com

Abstract

We adapt the dynamic-oracle training method of Goldberg and Nivre (2012; 2013) to train classifiers that produce probabilistic output. Evaluation of an Arc-Eager parser on 6 languages shows that the AdaGrad-RDA based training procedure results in models that provide the same high level of accuracy as the averaged-perceptron trained models, while being sparser and providing well-calibrated probabilistic output.

1 Introduction

For dependency parsing, it is well established that greedy transition-based parsers (Nivre, 2008) are very fast (both empirically and theoretically) while still providing relatively high parsing accuracies (Nivre et al., 2007; Kübler et al., 2009).

Recently, it has been shown that by moving from static to dynamic oracles during training, together with a training method based on the averaged-perceptron, greedy parsers can become even more accurate. The accuracy gain comes without any speed penalty at parsing time, as the inference procedure remains greedy (Goldberg and Nivre, 2012).

In transition-based parsing, the parsing task is viewed as performing a series of actions, which result in an incremental construction of a parse-tree. At each step of the parsing process, a classification model is used to assign a score to each of the possible actions, and the highest-scoring action is chosen and applied. When using perceptron based training, the action scores are in the range $(-\infty, \infty)$, and the only guarantee is that the highest-scoring action should be considered "best". Nothing can be inferred from the scale of the highest-scoring action, as well as from the scores assigned to the other actions.

In contrast, we may be interested in a classification model which outputs a proper probability distribution over the possible actions at each step of the process. Such output will allow us to identify uncertain actions, as well as to reason about the various alternatives. Probabilistic output can also be used in situations such as best-first parsing, in which a probabilistic score can be used to satisfy the required "superiority" property of the scoring function (Sagae and Lavie, 2006; Zhao et al., 2013).

Classifiers that output probabilities are well established, and are known as maximum-entropy or multinomial logistic regression models. However, their applications in the context of the dynamic-oracle training is not immediate. The two main obstacles are (a) the dynamic oracle may provide more than one correct label at each state while the standard models expect a single correct label, and (b) the exploration procedure used by Goldberg and Nivre (2012; 2013) assumes an online-learning setup, does not take into account the probabilistic nature of the classifier scores, and does not work well in practice.

This work is concerned with enabling training of classifiers with probabilistic outputs in the dynamic-oracle framework. Concretely, we propose a loss function capable of handling multiple correct labels, show how it can be optimized in the AdaGrad framework (Duchi et al., 2010), and adapt the exploration procedure used in dynamic-oracle training to the probabilistic setup. We use a variant of AdaGrad that performs RDA-based (Xiao, 2010) L_1 regularization, achieving sparse model at inference time.

We implemented our method and applied it to training an Arc-Eager dependency parser on treebanks in

6 languages. On all languages we achieve 1-best parsing results which are on-par with the averaged-perceptron trained models, while also providing well calibrated probability estimates at each step. The probabilistic models have 3-4 times fewer parameters than the perceptron-trained ones. Our code is available for download at the author's web page.

2 Background

2.1 Transition Based Parsing

We begin with a quick review of transition-based dependency parsing (Nivre, 2008), establishing notation. Transition-based parsing assumes a *transition system*, an abstract machine that processes sentences and produces parse trees. The transition system has a set of *states* (also called *configurations*) and a set of *actions* (also called *transitions*) which are applied to states, producing new states. In what follows we denote a state as $x \in \mathcal{X}$, an action as $y \in \mathcal{Y}$, and an application of an action to a state as $y(x)$. When parsing, the system is initialized to an *initial state* based on the input sentence S, to which actions are applied repeatedly. After a finite (in our case linear) number of action applications, the system arrives at a *terminal state*, and a parse tree is read off the terminal configuration. In a greedy parser, a classifier is used to choose the action to take in each state, based on features extracted from the state itself. Transition systems differ by the way they define states, and by the particular set of transitions available. One such system is the Arc-Eager system (Nivre, 2003), which has 4 action types, SHIFT, REDUCE, LEFT$_{lb}$, RIGHT$_{lb}$, where the last two are parameterized by a dependency label lb, resulting in 2+2L actions for a treebank with L distinct arc labels. The system parses a sentence with n words in $2n$ actions. The reader is referred to (Nivre, 2003; Nivre, 2008; Goldberg and Nivre, 2013) for further details on this system.

2.2 Greedy parsing algorithm

Assuming we have a function score$(x, y; w)$ parameterized by a vector w and assigning scores to pairs of states x and actions y, greedy transition-based parsing is simple and efficient using Algorithm 1. Starting with the initial state for a given sentence, we repeatedly choose the highest-scoring action according to our model parameters w and apply it, until we reach a terminal state, at which point we stop and return the parse tree accumulated in the configuration.

Algorithm 1 Greedy transition-based parsing

```
1: Input: sentence $S$, parameter-vector $w$
2: $x \leftarrow$ INITIAL$(S)$
3: while not TERMINAL$(x)$ do
4:     $y \leftarrow \arg\max_{y \in \text{LEGAL}(x)} \text{score}(x, y; w)$
5:     $x \leftarrow y(x)$
6: return tree$(x)$
```

In practice, the scoring function takes a linear (or log-linear) form:

$$\text{score}(x, y; w) \propto w \cdot \phi(x, y)$$

where ϕ is a feature extractor returning a high-dimensional sparse vector, and $\cdot$ is the dot-product operation. The role of training a model is to a set good weights to the parameter vector w, based on a training corpus of $\langle x, y\rangle$ pairs. The corpus is provided in the form of $\langle \text{sentence}, \text{tree}\rangle$ pairs, from which states and actions are extracted.

2.3 Static vs. Dynamic Oracles

Until recently, the training corpus of $\langle x, y\rangle$ pairs was extracted by use of a static-oracle – a function mapping a $\langle \text{sentence}, \text{tree}\rangle$ pair to a sequence of $\langle x, y\rangle$ pairs.

Recently, Goldberg and Nivre (2012; 2013) proposed the notion of a *dynamic* parsing oracle. Dynamic parsing oracles are functions oracle$(x; T)$ from a state x to set of actions Y (given a reference tree T). Note that unlike the static oracles which provide only $\langle x, y\rangle$ pairs that are part of a single action sequence leading to a gold tree (and associate a single action y with each state x on this path), the dynamic oracles are defined for every state x (even states that cannot lead to the gold tree), and may associate more than a single action y with each state. The semantics of the dynamic oracle is that the set Y associated with state x contains all and only actions that can lead to an optimal tree (in terms of hamming distance from the reference tree T) which is reachable from state x.

2.4 A Dynamic Oracle for the Arc-Eager System

Goldberg and Nivre (2013) provide a concrete dynamic oracle for the Arc-Eager system, which we use in this work and repeat here for completeness.

We use a notation in which dependency arcs are of the form (h, m) where h is a head and m is a modifier, and a tree T is represented as a set of dependency arcs. Each state x is of the form $x = (\sigma|s, b|\beta, A)$[1] where $\sigma|s$ is a stack with body σ and top s, $b|\beta$ is a buffer (queue) with body β and top b, and A is a set of dependency arcs.

The dynamic oracle for the Arc-Eager system works by calculating the cost of each action in a given state, and returning the set of actions with a cost of zero (the set is guaranteed to be non-empty):

$$\text{oracle}(x, T) = \{a \mid \text{cost}(a; x, T) = 0\}$$

The cost of an action at a state is the number of gold arcs which are mutually-reachable from the state, but will not be reachable after taking the action. The cost function $\text{cost}(\textsc{Action}; x, T)$ of taking an action at state x with respect to a gold set T of dependency arcs is calculated as follows (for further details, see (Goldberg and Nivre, 2013)):

- $\text{cost}(\textsc{Left}; x, T)$: Adding the arc (b, s) and popping s from the stack means that s will not be able to acquire any head or dependents in β. The cost is therefore the number of arcs in T of the form (k, s) or (s, k) such that $k \in \beta$. Note that the cost is 0 for the trivial case where $(b, s) \in T$, but also for the case where b is not the gold head of s but the real head is not in β (due to an erroneous previous transition) and there are no gold dependents of s in β.

- $\text{cost}(\textsc{Right}; x, T)$: Adding the arc (s, b) and pushing b onto the stack means that b will not be able to acquire any head in σ or β, nor any dependents in σ. The cost is therefore the number of arcs in T of the form (k, b), such that $k \in \sigma \cup \beta$, or of the form (b, k) such that $k \in \sigma$ and there is no arc (u, k) in A_c. Note again that the cost is 0 for the trivial case where $(s, b) \in T$, but also for the case where s is not the gold head of b but the real head is not in σ or β (due to an erroneous previous transition) and there are no gold dependents of b in σ.

- $\text{cost}(\textsc{Reduce}; x, T)$: Popping s from the stack means that s will not be able to acquire any dependents in $B = b|\beta$. The cost is therefore the number of arcs in T of the form (s, k) such that $k \in B$. While it may seem that a gold arc of the form (k, s) should be accounted for as well, note that a gold arc of that form, if it exists, is already accounted for by a previous (erroneous) $\textsc{Right}$ transition when s acquired its head.

- $\text{cost}(\textsc{Shift}; x, T)$: Pushing b onto the stack means that b will not be able to acquire any head or dependents in $S = s|\sigma$. The cost is therefore the number of arcs in T of the form (k, b) or (b, k) such that $k \in S$ and (for the second case) there is no arc (u, k) in A_c.

2.5 Training with Exploration

An important assumption underlying the training of greedy transition-based parsing models is that an action taken in a given state is independent of previous or future actions given the feature representation of the state. This assumption allows treating the training data as a bag of independent $\langle \text{state}, \text{action} \rangle$ pairs, ignoring the fact that the states and actions are part of a sequence leading to a tree, and not considering the interactions between different actions. If the data (both at train and test time) was separable and we could achieve perfect classification at parsing time, this assumption would hold. In reality, however, perfect classification is not possible, and different actions do influence each other. In particular, once a mistake is made, the parser may either reach a state it has not seen in training, or reach a state it has seen before, but needs to react differently to (previous erroneous decisions caused the state to be associated with different optimal actions). The effect of this is *error-propagation*: once a parser erred, it is more likely to err again, as it reaches states it was not trained on, and don't know how to react to them.

As demonstrated by (Goldberg and Nivre, 2012), error propagation can be mitigated to some extent by training the parser on states resulting from common parser errors. This is referred to as "training with exploration" and is enabled by the dynamic oracle. In (Goldberg and Nivre, 2012), training with exploration is performed by sometimes following incorrect classifier predictions during training.

Training with exploration still assumes that the $\langle x, y \rangle$ pairs are independent from each other given the

[1]This is a slight abuse of notation, since for the $\textsc{Shift}$ action s may not exist, and for the $\textsc{Reduce}$ action b may not exist.

feature representation, but instead of working with a fixed corpus D of $\langle x, y\rangle$ pairs, the set D is generated dynamically based on states x the parser is likely to reach, and the optimal actions $Y = oracle(x; T)$ proposed for these states by the dynamic oracle.

In practice, training with exploration using the dynamic oracle yields substantial improvements in parsing accuracies across treebanks.

3 Training of Sparse Probabilistic Classifiers

As discussed in the introduction, our aim is to replace the averaged-perceptron learner and adapt the training with exploration method of (Goldberg and Nivre, 2012) to produce classifiers that provide probabilistic output.

3.1 Probabilistic Objective Function and Loss

Our first step is to replace the perceptron hinge-loss objective with an objective based on log-likelihood. As discussed in section 2.5 the training corpus is viewed as a bag of states and their associated actions, and our objective would be to maximize the (log) likelihood of the training data under a probability model.

Static-oracle objective In static-oracle training each state x is associated with a single action y.

Denoting the label by $y \in \mathcal{Y}$ and the states by $x \in \mathcal{X}$, we would like to find a parameter vector w to maximize the data log-likelihood L of our training data D under parameter values w:

$$L(D; w) = \sum_{\langle x,y\rangle \in D} \log P(y|x; w)$$

where $P(y|x; w)$ takes the familiar log-linear form:

$$P(y|x; w) = \frac{\exp w \cdot \phi(x, y)}{\sum_{y' \in \mathcal{Y}} \exp w \cdot \phi(x, y')}$$

in which ϕ is a feature extraction function and $\cdot$ is the dot-product operation. This is the well known maximum-entropy classification formulation, also known as multinomial logistic regression.

Dynamic-oracle objective When moving to the dynamic oracle setting, each state x is now associated with a *set* of correct actions $Y \subseteq \mathcal{Y}$, and we would like at least one of these actions $y \in Y$ to get a high probability under the model. To accommodate this, we change the numerator to sum over the elements $y \in Y$, resulting in the following model form (the same approach was taken by Riezler et al. (2002) for dealing with latent LFG derivations in LFG parser training, and by Charniak and Johnson (2005) in the context of discriminative reranking):

$$L(D; w) = \sum_{\langle x,Y\rangle \in D} \log P(Y|x; w)$$

$$P(Y|x; w) = \frac{\sum_{y \in Y} \exp w \cdot \phi(x, y)}{\sum_{y' \in \mathcal{Y}} \exp w \cdot \phi(x, y')}$$

Note the change from y to Y, and the difference between the Y in the numerator (denoting the set of correct outcomes) and $\mathcal{Y}$ in the denominator (denoting the set of all possible outcomes). This subsumes the definition of $P(y|x; w)$ given above as a special case by setting $Y = \{y\}$. We note that the sum ensures that at least one $y \in Y$ receives a high probability score, but also allows other elements of Y to receive low scores.

The (convex) loss for a given $\langle x, Y\rangle$ pair under this model is then:

$$\text{loss}(Y, x; w) = \log \sum_{y \in Y} e^{w \cdot \phi(x,y)} - \log \sum_{y \in \mathcal{Y}} e^{w \cdot \phi(x,y)}$$

and the gradient of this loss with respect to w is:

$$\frac{\partial \text{loss}}{\partial w_i} = \sum_{y \in Y} \frac{e^{w \cdot \phi(x,y)}}{Z_Y} \phi_i(x, y) - \sum_{y \in \mathcal{Y}} \frac{e^{w \cdot \phi(x,y)}}{Z_{\mathcal{Y}}} \phi_i(x, y)$$

where:

$$Z_Y = \sum_{y' \in Y} e^{w \cdot \phi(x,y')} \qquad Z_{\mathcal{Y}} = \sum_{y' \in \mathcal{Y}} e^{w \cdot \phi(x,y')}$$

3.2 L$_1$ Regularized Training with AdaGrad-RDA

The generation of the training set used in the training-with-exploration procedure calls for an online optimization algorithm. Given the objective function and its gradient, we could have used a stochastic gradient based method to optimize the objective. However, recent work in NLP (Green et al., 2013; Choi and McCallum, 2013) demonstrated that the adaptive-gradient (AdaGrad) optimization framework of Duchi et al. (2010) converges quicker and produces superior results in settings which have a large number of training instances, each with a very high-dimensional but sparse feature representation, as common in NLP and in dependency-parsing in particular.

Moreover, a variant of the AdaGrad algorithm called AdaGrad-RDA incorporates an L_1 regularization, and produces sparse, regularized models. Regularization is important in our setting for two reasons: first, we would prefer our model to not overfit accidental features of the training data. Second, smaller models require less memory to store, and are faster to parse with as more of the parameters can fit in the CPU cache.[2]

For these reasons, we chose to fit our model's parameters using the regularized-dual-averaging (RDA) variant of the AdaGrad algorithm. The AdaGrad framework works by maintaining a per-feature learning rate which is based on the cumulative gradient values for this feature. In the RDA variant, the regularization depends on the average vector of all the gradients seen so far.

Formally, the weight after the $J+1$th AdaGrad-RDA update with a regularization parameter λ is:

$$w_i^{J+1} \leftarrow \alpha \frac{1}{\sqrt{G_i + \rho}} \text{shrink}(g_i, J\lambda)$$

where w_i^{J+1} is the value of the ith coordinate of w at time $J+1$, α is the learning rate, and ρ is a ridge parameter used for numerical stability (we fix $\rho = 0.01$ in all our experiments). G is the sum of the squared gradients seen so far, and g is the sum of the gradients seen so far.

$$G_i = \sum_{j=0}^{J} (\partial_i^j)^2 \qquad g_i = \sum_{j=0}^{J} \partial_i^j$$

$\text{shrink}(g_i, J\lambda)$ is the regularizer, defined as:

$$\text{shrink}(g_i, J\lambda) = \begin{cases} g_i - J\lambda & \text{if } g_i > 0, |g_i - J\lambda| > 0 \\ g_i + J\lambda & \text{if } g_i < 0, |g_i - J\lambda| > 0 \\ 0 & \text{otherwise} \end{cases}$$

[2]Note that in contrast to the perceptron loss that considers only the highest-scoring and the correct class for each instance, the multilabel log-likelihood loss considers all of the classes. When the number of classes is large, such as in the case of labeled parsing, this will result in very many non-zero scores, unless strong regularization is employed.

For efficiency reasons, the implementation of the AdaGrad-RDA learning algorithm keeps track of the two vectors G and g, and calculates the needed coordinates of w based on them as needed. When training concludes, the final w is calculated and returned. We note that while the resulting w is sparse, the G and g vectors are quite dense, requiring a lot of memory at training time.[3]

For completeness, the pseudo-code for an AdaGrad-RDA update with our likelihood objective is given in algorithm 2.

Algorithm 2 Adagrad-RDA with multilabel logistic loss update.

Globals The global variables G, g and j are initialized to 0. The vectors g and G track the sum and the squared sum of the gradients. The scalar j tracks the number of updates.
Parameters α: learning rate, ρ: ridge, λ: L_1 penalty.
Arguments w: current weight vector, ϕ feature extraction function, x: state, Y: set of good labels (actions) for x.
Others Z_Y, $Z_\mathcal{Y}$ and shrink(·,·) are as defined above.
Returns: An updated weight vector w.

```
1: function AdagradUpdate($w, \phi, x, Y$)
2:     $\forall y \in \mathcal{Y} \quad f_y = \begin{cases} \frac{e^{w\cdot\phi(x,y)}}{Z_Y} & \text{if } y \in Y \\ 0 & \text{otherwise} \end{cases}$
3:     $\forall y \in \mathcal{Y} \quad p_y = \frac{e^{w\cdot\phi(x,y)}}{Z_\mathcal{Y}}$
4:     for $i$ s.t. $\exists y, \phi_i(x,y) \neq 0$ do
5:         $\partial_i = \sum_{y\in\mathcal{Y}} \phi_i(x,y)(f_y - p_y)$
6:         $g_i \leftarrow g_i + \partial_i$
7:         $G_i \leftarrow G_i + \partial_i^2$
8:         $w_i \leftarrow \alpha \frac{1}{\sqrt{G_i+\rho}} \text{shrink}(g_i, j\lambda)$
9:         $j \leftarrow j + 1$
       return $w$
```

3.3 Probabilistic Data Exploration

A key component of dynamic-oracle training is that the training set D is not fixed in advance but changes according to the training progression. As we cannot explore the state set $\mathcal{X}$ in its entirety due to its exponential size (and because the optimal actions Y at a state x depend on the underlying sentence), we would like to explore regions of the state space that we are likely to encounter when parsing using the parameter

[3]For example, in our implementation, training on the English treebank with 950k tokens and 42 dependency labels requires almost 12GB of RAM for AdaGrad-RDA vs. less than 1.8GB for the averaged-perceptron.

vector w, together with their optimal actions Y according to the dynamic oracle.

That is, our set D is constructed by sampling values from $\mathcal{X}$ in accordance to our current belief w, and using the oracle $oracle(x; T)$ to associate Y values with each x. In the averaged-perceptron setup, this sampling is achieved by following the highest-scoring action rather than a correct one according to the oracle with some (high) probability p. This approach does not fit well with our probabilistic framework, for two reasons. (a) Practically, the efficiency of the AdaGrad optimizer results in the model achieving a good fit of the training data very quickly, and the highest scoring action is all too often a correct one. While great from an optimization perspective, this behavior limits our chances of exploring states resulting from incorrect decisions. (b) Conceptually, focusing on the highest scoring action ignores the richer structure that our probabilistic model offers, namely the probabilistic interpretation of the scores and the relations between them.

Instead, we propose a natural sampling procedure. Given a state x we use our model to obtain a multinomial distribution $P(y|x; w)$ over possible next actions y, sample an action from this distribution, and move to the state resulting from the sampled action.[4] This procedure focuses on states that the model has a high probability of landing on, while still allowing exploration of less likely states.

The training procedure is given in algorithm 3. In the first iteration, we focus on states that are on the path to the gold tree by following actions $\hat{y}$ in accordance to the oracle set Y (line 6), while on subsequent iteration we explore states which are off of the gold path by sampling the next action $\hat{y}$ in accordance to the model belief $P(y|x; w)$ (line 8).

4 Evaluation and Results

Data and Experimental Setup We implemented the above training procedure in an Arc-Eager transition based parser, and tested it on the 6 languages comprising the freely available Google Universal Dependency Treebank (McDonald et al., 2013). In all cases, we trained on the training set and evaluated the models on the dev-set, using *gold POS-tags* in both test and train time. Non-projective sentences were removed from the training set. In all scenarios, we used the feature set of (Zhang and Nivre, 2011). We compared different training scenarios: training perceptron based models (PERCEP) and probabilistic models (ME) with static (ST) and dynamic (DYN) oracles. For the dynamic oracles, we varied the parameter k (the number of initial iterations without error exploration). For the probabilistic dynamic-oracle models further compare the sampling-based exploration described in Algorithm 3 with the error-based exploration used for training the perceptron models in Goldberg and Nivre (2012, 2013). All models were trained for 15 iterations. The PERCEP+DYN models are the same as the models in (Goldberg and Nivre, 2013). For the ME models, we fixed the values of $\rho = 0.01$, $\alpha = 1$ and $\lambda = 1/20|D|$ where we take $|D|$ to be the number of tokens in the training set.[5]

Parsing Accuracies are listed in Table 1. Two trends are emergent: Dynamic Oracles with Error Ex-

Algorithm 3 Online training with exploration for probabilistic greedy transition-based parsers (ith iteration)

```
1:  for sentence $S$ with gold tree $T$ in corpus do
2:      $x \leftarrow$ INITIAL($S$)
3:      while not TERMINAL($x$) do
4:          $Y \leftarrow$ oracle$(x, T)$
5:          $P(y|x; w) \leftarrow \frac{\exp w \cdot \phi(x,y)}{\sum_{y' \in \mathcal{Y}} \exp w \cdot \phi(x,y')}$  $\forall y \in \mathcal{Y}$
6:          if $i \leq k$ then
7:              $\hat{y} \leftarrow \arg\max_{y \in Y} P(y|x; w)$
8:          else
9:              Sample $\hat{y}$ according to $P(y|x; w)$
10:         $w \leftarrow$ ADAGRADUPDATE$(w, \phi, x, Y)$
11:         $x \leftarrow \hat{y}(x)$
    return $w$
```

[4]Things are a bit more complicated in practice: as not all actions are valid at each state due to preconditions in the transition system, we restrict $P(y|x; w)$ to only the set of valid actions at x, and renormalize. In case x is a terminal state (and thus having no valid actions) we move on to the initial state of the next sentence. The sentences are sampled uniformly without replacement at each training round.

[5]We set the ρ and α values based on initial experiments on an unrelated dataset. The formula for the L_1 penalty λ is based on an advice from Alexandre Passos (personal communication) which proved to be very effective. We note that we could have probably gotten a somewhat higher scores in all the settings by further optimizing the ρ, α and λ parameters, as well as the number of training iterations, on held-out data.

SETUP	DE UAS / LAS	EN UAS / LAS	ES UAS / LAS	FR UAS / LAS	KO UAS / LAS	SV UAS / LAS
PERCEP+ST	84.95 / 80.32	91.06 / 89.48	85.93 / 82.82	85.75 / 82.44	79.96 / 71.90	83.21 / 79.40
ME+ST	84.71 / 80.06	90.83 / 89.32	85.72 / 82.59	85.42 / 82.19	80.47 / 72.15	83.12 / 79.36
PERCEP+DYN(K=1)	86.30 / 81.67	92.22 / 90.72	86.68 / 83.64	86.95 / 83.93	80.59 / 72.66	84.16 / 80.48
PERCEP+DYN(K=0)	86.50 / 81.88	92.28 / 90.82	86.18 / 83.19	86.87 / 83.70	80.59 / 73.06	84.79 / 81.00
ME+DYN(K=1,SAMPLE)	86.34 / 82.04	92.16 / 90.73	86.38 / 83.57	86.59 / 83.46	80.92 / 73.06	84.56 / 80.97
ME+DYN(K=0,SAMPLE)	86.51 / 82.19	92.30 / 90.83	86.66 /83.77	86.69 / 83.61	81.17 / 73.19	84.17 / 80.54

Table 1: Labeled (LAS) and Unlabeled (UAS) parsing accuracies of the different models on various datasets. All scores are excluding punctuations an using gold POS-tags. Dynamic-oracle training with error exploration clearly outperforms static-oracle training. The perceptron and ME results are equivalent.

SETUP	DE UAS / LAS	EN UAS / LAS	ES UAS / LAS	FR UAS / LAS	KO UAS / LAS	SV UAS / LAS
ME+DYN(K=1,ERR)	85.26 / 80.94	91.62 / 90.10	86.08 / 82.92	86.13 / 83.06	80.42 / 71.97	83.73 / 80.03
ME+DYN(K=0,ERR)	85.78 / 81.63	91.77 / 90.30	86.37 / 83.33	86.53 / 83.23	80.94 / 72.57	83.73 / 80.05
ME+DYN(K=1,SAMPLE)	86.34 / 82.04	92.16 / 90.73	86.38 / 83.57	86.59 / 83.46	80.92 / 73.06	84.56 / 80.97
ME+DYN(K=0,SAMPLE)	86.51 / 82.19	92.30 / 90.83	86.66 /83.77	86.69 / 83.61	81.17 / 73.19	84.17 / 80.54

Table 2: Comparing the sampling-based exploration in Algorithm 3 with the error-based exploration of Goldberg and Nivre (2012, 2013). Labeled (LAS) and Unlabeled (UAS) parsing accuracies of the different models on various datasets. All scores are excluding punctuations an using gold POS-tags. The sampling based algorithm outperforms the error-based one.

ploration in training (DYN) models clearly outperform the models trained with the traditional static oracles (ST), and the probabilistic models (ME) perform on par with their averaged-perceptron (PERCEP) counterparts.

Sampling vs. Error-Driven Exploration Table 2 verifies that the sampling-based exploration proposed in this work is indeed superior to the error-based exploration which was used in Goldberg and Nivre (2012, 2013), when training multinomial logistic-regression models with the AdaGrad-RDA algorithm.

Model Sizes Table 3 lists the number of parameters in the different models. RDA regularization is effective: the regularization ME models are much smaller. In the accurate dynamic oracle setting, the regularized ME models are 3-4 times smaller than their averaged-perceptron counterparts, while achieving roughly the same accuracies.

Calibration To asses the quality of the probabilistic output of the ME+DYN models, we binned the probability estimates of the highest-scored actions into 10 equally-sized bins, and for each bin calculated the percentage of time an action falling in the bin was correct. Table 4 lists the results, together with the bin sizes.

SETUP	DE	EN	ES	FR	KO	SV
PERCEP+ST	438k	5.4M	1.2M	849k	1.9M	912k
ME+ST	150k	1.9M	448k	294k	725k	304k
PERCEP+DYN	525k	8.5M	1.7M	1.1M	2.9M	1.2M
ME+DYN	160k	2.4M	516k	336k	755k	357k

Table 3: Model sizes (number of non-0 parameters).

First, it is clear that the vast majority of parser actions fall in the 0.9-1.0 bin, indicating that the parser is confident, and indeed the parser is mostly correct in these cases. Second, the models seem to be well calibrated from the 0.5-0.6 bin and above. The lower bins are under-estimating the confidence, but they also contain very few items. Overall, the probability output of the ME+DYN model is calibrated and trustworthy.[6]

5 Conclusions

We proposed an adaptation of the dynamic-oracle training with exploration framework of Goldberg and Nivre (2012; 2013) to train classifiers with probabilistic output, and demonstrated that the method works:

[6]Note, however, that with 69k predictions in bin 0.9-1.0 for English, an accuracy of 98% means that almost 1400 predictions with a probability score above 0.9 are, in fact, wrong.

BIN	DE	EN	ES	FR	KO	SV
0.1	(7) 71%	(1) 0%	(3) 66%	(2) 1%	(0) 0%	(2) 50%
0.2	(51) 51%	(38) 55%	(26) 57%	(17) 64%	(2) 100%	(21) 57%
0.3	(121) 54%	(139) 54%	(83) 65%	(58) 72%	(29) 55%	(100) 61%
0.4	(292) 54%	(323) 62%	(206) 65%	(146) 57%	(178) 63%	(193) 63%
0.5	(666) 66%	(1.2k) 64%	(642) 68%	(453) 66%	(464) 55%	(578) 62%
0.6	(787) 66%	(1.4k) 69%	(694) 73%	(469) 70%	(616) 60%	(636) 70%
0.7	(840) 73%	(1.7k) 74%	(853) 77%	(546) 73%	(739) 65%	(747) 75%
0.8	(1.5k) 78%	(2.9k) 82%	(1.2k) 80%	(810) 78%	(1.1k) 72%	(1.2k) 80%
0.9	(18.5k) 97%	(69k) 98%	(16k) 97%	(13k) 97%	(9k) 96%	(14k) 96%
1.0	(800) 100%	(1.7k) 100%	(370) 100%	(366) 100%	(588) 100%	(493) 100%

Table 4: Calibration of the ME+DYN(K=0,SAMPLE) scores. (num) denotes the number of items in the bin, and num% the percent of correct items in the bin. The numbers for ME+DYN(K=1,SAMPLE) are very similar.

the trained classifiers produce well calibrated probability estimates, provide accuracies on par with the averaged-perceptron trained models, and, thanks to regularization, are 3-4 times smaller. However, the training procedure is slower than for the averaged-perceptron models, requires considerably more memory, and has more hyperparameters. If probabilistic output or sparse models are required, this method is recommended. If one is interested only in 1-best parsing accuracies and can tolerate the larger model sizes, training with the averaged-perceptron may be preferable.

References

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria, August. Association for Computational Linguistics.

John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1.

Spence Green, Sida Wang, Daniel Cer, and Christopher D. Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 311–321, Sofia, Bulgaria, August. Association for Computational Linguistics.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Stephan Riezler, Margaret H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–278.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698.

Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 9:2543–2596.

Yue Zhang and Joakim 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.

Kai Zhao, James Cross, and Liang Huang. 2013. Dynamic programming for optimal best-first shift-reduce parsing. In *EMNLP*.

Dependency Structure for Incremental Parsing of Japanese and Its Application

Tomohiro Ohno
Information Technology Center,
Nagoya University
Nagoya, Japan
ohno@nagoya-u.jp

Shigeki Matsubara
Graduate Schoool of Information Science,
Nagoya University
Nagoya, Japan
matubara@nagoya-u.jp

Abstract

Previous researches on incremental dependency parsing have not discussed how a parser should output the information about dependency relation where the modified word has not been inputted yet. This paper proposes a dependency structure which a Japanese incremental dependency parser should produce in the middle of an input sentence. The dependency structure also expresses the fact that a word depends on a word which the parser would receive later. In addition, we present a method for incremental dependency parsing to produce our proposed dependency structure. As the result of an experiment on incremental parsing, we confirmed the feasibility of our incremental dependency parsing. Furthermore, this paper shows its application in the simultaneous linefeed insertion for real-time generation of more readable captions, and then describes the effectiveness of our proposed dependency structure.

1 Introduction

In applications such as simultaneous machine interpretation, spoken dialogue and real-time captioning, it is necessary to execute processing simultaneously with speech input, and thus, the questions of what syntactic information is acquirable and when the information can be acquired become issues. However, previous researches on incremental dependency parsing (Kato et al., 2005; Johansson and Nugues, 2007; Nivre, 2008) have not discussed how a parser should output the information about dependency relation where the modified word has not been inputted.

This paper proposes a dependency structure which a Japanese incremental dependency parser should produce in the middle of an input sentence. Our proposed dependency structure makes a parser clarify the fact that the modified *bunsetsu*[1] of a bunsetsu will be inputted later. This enables a higher layer application to use the information that the bunsetsu does not depend on any bunsetsus which have been already inputted. Especially, in the case of Japanese, since the modified bunsetsu is always inputted after the modifier bunsetsu, our proposal is effective for the applications which execute processing simultaneously with speech input.

In addition, this paper describes a method for incremental dependency parsing to produce our proposed dependency structure whenever receiving a bunsetsu. We conducted an experiment on incremental parsing and confirmed the feasibility of our method. Furthermore, this paper shows its application to the simultaneous linefeed insertion for real-time generation of more readable captions. As the result of an experiment on linefeed insertion, we confirmed the effectiveness of our proposed dependency structure.

2 Dependency structure for incremental parsing

Since applications such as simultaneous machine interpretation and real-time captioning need to execute the process simultaneously with speech input, it is desirable for them to be able to acquire the dependency information at any time. In our research, incremental dependency parsing shall output the parsing result whenever receiving a bunsetsu.

[1]*Bunsetsu* is a linguistic unit in Japanese that roughly corresponds to a basic phrase in English. A bunsetsu consists of one independent word and zero or more ancillary words. A dependency relation in Japanese is a modification relation in which a modifier bunsetsu depends on a modified bunsetsu. That is, the modifier bunsetsu and the modified bunsetsu work as modifier and modifyee, respectively.

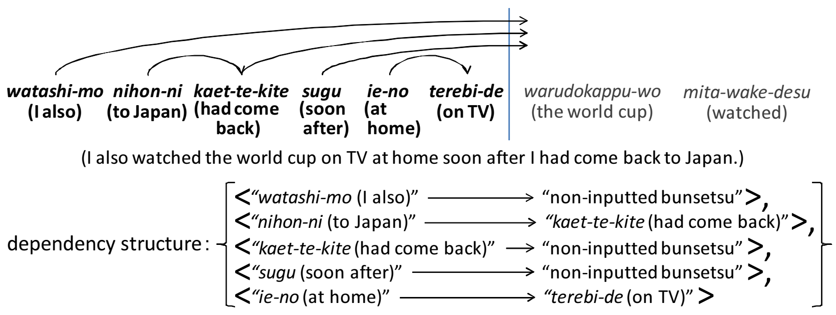

watashi-mo (I also)
nihon-ni (to Japan)
kaet-te-kite (had come back)
sugu (soon after)
ie-no (at home)
terebi-de (on TV)
warudokappu-wo (the world cup)
mita-wake-desu (watched)
(I also watched the world cup on TV at home soon after I had come back to Japan.)
dependency structure :
<"watashi-mo (I also)" → "non-inputted bunsetsu">,
<"nihon-ni (to Japan)" → "kaet-te-kite (had come back)">,
<"kaet-te-kite (had come back)" → "non-inputted bunsetsu">,
<"sugu (soon after)" → "non-inputted bunsetsu">,
<"ie-no (at home)" → "terebi-de (on TV)">

Figure 1: Example of a dependency structure outputted in the middle of an input sentence

Next, we discuss the output contents of incremental dependency parsing. When applications use incremental dependency parsing, the information about whether or not the dependency structure of a sequence of busnetsus is closed, that is, the information about a syntactically sufficient unit is also useful. In fact, when identifying the timing of the start of interpreting in the simultaneous machine interpretation (Ryu et al., 2006; Fügen et al., 2007; Paulik and Waibel, 2010), the timing of inserting back-channel feedbacks (Fujie et al., 2004; Kamiya et al., 2010), and the appropriate position of a linefeed in captioning (Ohno et al., 2009), the information about a syntactically sufficient unit is used as an important clue.

In our research, an incremental dependency parsing shall clearly output the fact that a bunsetsu depends on a bunsetsu which will be later inputted. If it becomes clear that the modified bunsetsu of a bunsetsu has not been inputted yet, the higher layer applications can identify the syntactically sufficient units in the inputted sequence of bunsetsus and use the information effectively.

In what follows, we describe the dependency structure which incremental dependency parsing in our research outputs in the middle of an input sentence. When a sentence S composed of a sequence of bunsetsus $b_1 \cdots b_n$ is parsed[2], we define the dependency structure D_x which is outputted at the time that a bunsetsu $b_x (1 \leq x \leq n)$ is inputted as follows:

$$D_x = \left\{ \langle b_1 \rightarrow dep(b_1) \rangle, \cdots, \langle b_{x-1} \rightarrow dep(b_{x-1}) \rangle \right\}$$

$$\left(\text{Here, } dep(b_y) \in \begin{Bmatrix} \{b_{y+1}, \cdots, b_x, b_{over}\} (\text{if } b_x \neq b_n) \\ \{b_{y+1}, \cdots, b_x\} \qquad (\text{otherwise}) \end{Bmatrix} \atop (1 \leq y \leq x-1) \right)$$

[2]In this research, we assume that sentence boundaries are detected as its preprocessing.

where $\langle b_y \rightarrow dep(b_y) \rangle$ means the dependency relation where the modifier bunsetsu is b_y and where the modified bunsetsu is $dep(b_y)$. $dep(b_y)$ is a member of a set which contains not only $b_{y+1}, \cdots, b_x$, which are inputted after the bunsetsu b_y, but also b_{over}. $dep(b_y) = b_{over}$ means that the modified bunsetsu of b_y is not any of $b_{y+1}, \cdots, b_x$ but a bunsetsu which will be inputted later. In addition, we assume the following three Japanese syntactic constraints:

- No dependency is directed from right to left.
- Dependencies don't cross each other.
- Each bunsetsu, except the final one in a sentence, depends on only one bunsetsu.

Figure 1 shows an example of a dependency structure outputted in the middle of an input sentence. This example shows the dependency structure which is outputted right after the bunsetsu "*terebi-de* (on TV)" was inputted when a parser parses the sentence "*watashi-mo nihon-ni kaet-te-kite sugu ie-no terebi-de warudokappu-wo mita-wake-desu* (I also watched the world cup on TV at home soon after I had come back to Japan.)"

3 Incremental dependency parsing

This section describes a method to produce the dependency structure proposed in Section 2 whenever receiving a bunsetsu. This method is implemented by improving the dependency parsing method proposed by Uchimoto et al. (2000).

Generally, stochastic dependency parsing methods identify the dependency structure which maximizes the conditional probability $P(D|S)$, which means the probability that the dependency structure for a sentence S is D. In usual methods, a relationship between two bunsetsus is tagged with a "1" or "0" to indicate whether or not there is a dependency relation between the bunsetsus, respectively. On the other hand, in the Uchimoto et al.'s method, a relationship between two bunsetsus is tagged with a "between," "depend," or "beyond" to indicate the following three cases, respectively. The anterior bunsetsu can depend on (1) a bunsetsu between the two, (2) the posterior bunsetsu, or (3) a bunsetsu beyond the posterior one. Then, the dependency probability of two bunsetsus is estimated by using the product of the probabilities of the relationship between the left bunsetsu and those to its right in a sentence.

In our research, we thought that by adapting the basic idea in the Uchimoto et al.'s method, we might be able to calculate the probability that a bunsetsu does not depend on any bunsetsus which have been already inputted, that is, the probability that a bunsetsu depends on a non-inputted bunsetsu (b_{over}). As a difference from the Uchimoto et al.'s method, our method identifies the dependency structure D_x which should be outputted when the sequence of bunsetsus $B_x = b_1, \cdots, b_x (1 \leq x \leq n)$, which is a subsequence of a sentence $S = b_1 \cdots b_n$, is inputted. Our method outputs the dependency structure which maximizes the probability $P(D_x|B_x)$ as the most appropriate dependency structure for the sequence of bunsetsus B_x.

Here, D_x is described as an ordered set $\{d_1, \cdots, d_{x-1}\}$ of dependency relations d_i $(1 \leq i \leq x-1)$, where the modifier bunsetsu is a bunsetsu b_i. In addition, d_i is described as $d_i = \{d_{i,i+1}, \cdots, d_{i,x}, d_{i,x+1}\}$ $(1 \leq i \leq x-1)$. $d_{i,i+j}$ is a flag which indicates the relationship between bunsetsus b_i and b_{i+j}, and takes the following value:

$$d_{i,i+j} = \left\{ \begin{array}{ll} 0 & (1 \leq j < dist(i)) \\ 1 & (j = dist(i)) \\ 2 & (dist(i) < j \leq n-i) \end{array} \right\}$$

where $dist(i)$ is defined as $dist(i) = l - i$ when a bunsetsu b_i depends on a bunsetsu $b_l (i < l \leq x+1 \leq n)$. Note that our method adds a flag $d_{i,x+1}$, which indicates the relationship between b_i and a non-inputted bunsetsu b_{x+1}, to the elements of d_i as a dummy.

Using the above-mentioned notation, our method calculates $P(D_x|B_x)$ as follows[3]:

$$P(D_x|B_x)^2 = \prod_{i=1}^{x-1} P(d_i|B_x)$$

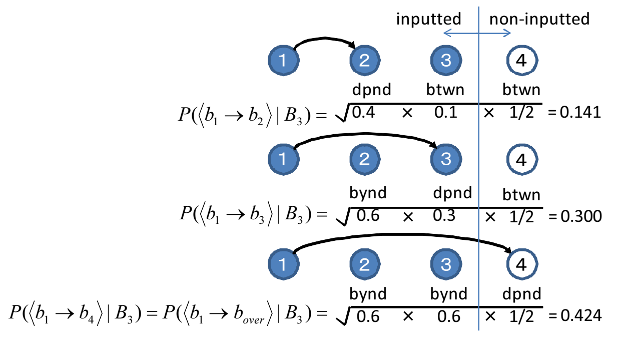

inputted
non-inputted
1
2
3
4
dpnd
btwn
btwn
$P(\langle b_1 \rightarrow b_2 \rangle | B_3) = \sqrt{0.4 \times 0.1 \times 1/2} = 0.141$
1
2
3
4
bynd
dpnd
btwn
$P(\langle b_1 \rightarrow b_3 \rangle | B_3) = \sqrt{0.6 \times 0.3 \times 1/2} = 0.300$
1
2
3
4
bynd
bynd
dpnd
$P(\langle b_1 \rightarrow b_4 \rangle | B_3) = P(\langle b_1 \rightarrow b_{over} \rangle | B_3) = \sqrt{0.6 \times 0.6 \times 1/2} = 0.424$

Figure 2: Calculation of dependency probability

$$= \prod_{i=1}^{x-1} P(d_{i,i+1}, \cdots, d_{i,x+1}|B_x)$$

$$= \prod_{i=1}^{x-1} \left(\prod_{j=1}^{dist(i)-1} P(d_{i,i+j} = 0|B_x) \times P(d_{i,i+dist(i)} = 1|B_x) \times \prod_{j=dist(i)+1}^{x-i+1} P(d_{i,i+j} = 2|B_x) \right)$$

where the value of $dist(i)$ is decided based on the D_x. Here, we use the following value as $P(d_{i,i+j}|B_x)$, which means the probability of the relationship ("beyond," "depend," or "between") between two bunsetsus.

- When $1 \leq j \leq x - i$: $P(d_{i,i+j}|B_x) =$ the value which is estimated using the maximum entropy method in the same way[4] as the Uchimoto et al.'s method.
- When $j = x - i + 1$: $P(d_{i,i+j} = 0|B_x) = 0$, $P(d_{i,i+j} = 1|B_x) = P(d_{i,i+j} = 2|B_x) = 1/2$

The reasons that we decided the second itemization are as follows: The relationship $d_{i,x+1}$ between a bunsetsu b_i and a non-inputted bunsetsu b_{x+1} does not become the relationship "beyond," but becomes either "depend" or "between." Moreover, we assume that the probability that the relationship becomes "depend" is equal to that of "between" because the probabilities are unknown. This assumption enables $argmax_{D_x} P(D_x|B_x)$ to be calculated with no relation to the probabilities between b_i and b_{x+1}.

Finally, if there exists a $\langle b_i \rightarrow b_{x+1} \rangle$ in the dependency structure D_x which maximizes $P(D_x|B_x)$,

[3] Uchimoto et al. (2000a) discussed the rationale and assumptions behind the form of this model.

[4] However, the features which we used in the maximum entropy method are the same as those used by Ohno et al. (2007).

our method outputs the dependency structure which is made by replacing every $\langle b_i \rightarrow b_{x+1}\rangle$ in the D_x with $\langle b_i \rightarrow b_{over}\rangle$.

Figure 2 shows how to calculate the dependency probability $P(d_1|B_3)$ when a bunsetsu b_3 in a sentence composed of $b_1, \cdots, b_5$ is inputted. Note that we consider $P(\langle b1 \rightarrow b_4\rangle|B_3) = P(\langle b1 \rightarrow b_{over}\rangle|B_3)$.

4 Experiment

To evaluate the feasibility of incrementally producing the dependency structure proposed in Section 2, we conducted an experiment on dependency parsing.

4.1 Outline of experiment

As the experimental data, we used the transcribed data of Japanese discourse speech in the Simultaneous Interpretation Database (Matsubara et al., 2002). All the data had been manually annotated with information on morphological analysis, bunsetsu boundary detection, clause boundary detection and dependency analysis (Ohno et al., 2009).

We performed a cross-validation experiment by using 16 discourses. That is, we repeated the experiment, in which we used one discourse from among 16 discourses as the test data and the others as the learning data, 16 times. However, since we used 2 discourses among 16 discourses as the preliminary analysis data, we evaluated the experimental results for the remaining 14 discourses (1,714 sentences, 20,707 bunsetsus). Here, as the morphological information, bunsetsu boundary information, and clause boundary information in the input of dependency parsing, we used one which had been manually annotated. In addition, we used the maximum entropy method tool (Zhang, 2013) with the default options except "-i (iteration) 2000."

4.2 Evaluation index

Our method is designed to output the dependency structure defined in Section 2 whenever a bunsetsu is inputted. Therefore, we introduced the following new evaluation index to evaluate the accuracy of incremental dependency parsing:

$$acc. = \frac{\sum_i^N \sum_{j=1}^{n_i} DepNum(Match(D_j^i, G_j^i))}{\sum_i^N \sum_{j=1}^{n_i} DepNum(D_j^i)}$$

where D_j^i and G_j^i indicate the dependency structure which a parser outputs and the correct one which

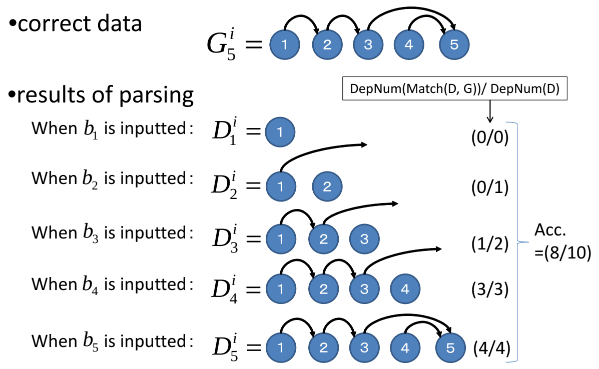

•correct data
$G_5^i =$
1 2 3 4 5
•results of parsing
DepNum(Match(D, G))/ DepNum(D)
When b_1 is inputted: $D_1^i =$
(0/0)
When b_2 is inputted: $D_2^i =$
(0/1)
When b_3 is inputted: $D_3^i =$
(1/2)
Acc. =(8/10)
When b_4 is inputted: $D_4^i =$
(3/3)
When b_5 is inputted: $D_5^i =$
(4/4)

Figure 3: Calculation of accuracy

	recall	precision	f-measure
non-inputted	70.4% (22,811/ 32,389)	75.0% (22,811/ 30,413)	72.6%
inputted	74.8% (102,778/ 137,376)	73.8% (102,778/ 139,352)	74.3%

Table 1: Recalls and precisions of our method

a parser should output respectively when a bunsetsu $b_j (1 \leq j \leq n_i)$ in a sentence $S_i (1 \leq i \leq N)$ is inputted. In addition, $DepNum()$ is a function which returns the number of elements of the input set of dependency relations, and $Match()$ is a function which returns an intersection between the two input sets of dependency relations.

Figure 3 shows an example of the calculation of our defined accuracy in case of parsing a sentence S_i composed of 5 bunsetsus. Here, we explain how to calculate the $DepNum(Match(D_3^i, G_3^i))$ when the 3rd bunsetsu is inputted. From the correct data of the dependency structure of a whole sentence S_i, $G_3^i = \{\langle b_1 \rightarrow b_2\rangle, \langle b_2 \rightarrow b_3\rangle\}$ is derived. On the other hand, the result of parsing is $D_3^i = \{\langle b_1 \rightarrow b_2\rangle, \langle b_2 \rightarrow b_{over}\rangle\}$. From the G_3^i and D_3^i, $DepNum(Match(D_3^i, G_3^i)) = 1$ is derived.

Furthermore, we obtained the recalls and precisions, separately for the case that the modified bunsetsu has not been inputted and the case that the modified bunsetsu has been inputted. The recall and precision for the former case are respectively defined as follows:

$$rec. = \frac{\sum_i^N \sum_{j=1}^{n_i} DepNum(Over(Match(D_j^i, G_j^i)))}{\sum_i^N \sum_{j=1}^{n_i} DepNum(Over(G_j^i))}$$

$$prec. = \frac{\sum_i^N \sum_{j=1}^{n_i} DepNum(Over(Match(D_j^i, G_j^i)))}{\sum_i^N \sum_{j=1}^{n_i} DepNum(Over(D_j^i))}$$

where Over() is a function which returns a set of dependency relations where the modified bunsetsu has not been inputted yet from among the input set of dependency relations. The recall and precision for the latter case are defined by replacing the Over() in the above-mentioned definition with the NonOver(). NonOver() is a function which returns a set of dependency relations where the modified bunsetsu has been inputted from among the input set of dependency relations.

4.3 Experimental results

The accuracy obtained using our method was 74.0% (125,589/169,765). Only as a guide, we conducted a parsing experiment by using the Uchimoto et al.'s method[5] and CaboCha(Kudo and Matsumoto, 2002), which identify the dependency structure after the whole sentence is inputted. As the result of measuring the conventional dependency accuracy ($= \sum_i^N DepNum(Match(D_{n_i}^i, G_{n_i}^i)) / \sum_i^N DepNum(D_{n_i}^i)$), the dependency accuracy of the Uchimoto et al.'s method and CaboCha were 75.8% (14.391/18,993) and 78.0% (14,814/18,993) respectively.

Table 1 shows the recalls and precisions of our method, separately for the case that the modified bunsetsu has not been inputted and the case that the modified bunsetsu has been inputted. The f-measure for the case that the modified bunsetsu has not been inputted was not as high as that for the non-inputted case. However, the precision for the inputted case was higher than that for the non-inputted case. We confirmed the feasibility of outputting our proposed dependency structure whenever a bunsetsu is inputted.

5 Application to sequential linefeed insertion

In this section, we describe application of our incremental dependency parsing described in Section 3 to sequential linefeed insertion.

5.1 Sequential linefeed insertion

Whenever a bunsetu is inputted, our method decisively judges whether or not a linefeed should be inserted between the inputted bunsetsu and the adjacent

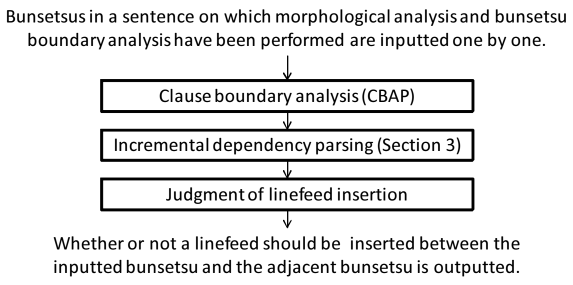

Bunsetsus in a sentence on which morphological analysis and bunsetsu boundary analysis have been performed are inputted one by one.
Clause boundary analysis (CBAP)
Incremental dependency parsing (Section 3)
Judgment of linefeed insertion
Whether or not a linefeed should be inserted between the inputted bunsetsu and the adjacent bunsetsu is outputted.

Figure 4: Flow of sequential linefeed insertion

bunsetsu. Here, morphological analysis and bunsetsu boundary analysis have been already performed for the input sequence of bunsetsus in a sentence.

Figure 4 shows the flow of our sequential linefeed insertion. First, our method decides whether there exists a clause boundary between the inputted bunsetsu b_x and the adjacent bunsetsu b_{x-1} or not by using Clause Boundary Annotation Program (CBAP) (Kashioka and Maruyama, 2004). Next, our method parses the dependency structure D_x proposed in Section 2 by using the incremental dependency parsing method proposed in Section 3.

Finally, in judgment of linefeed insertion, our method decisively judges whether a linefeed should be inserted between the inputted bunsetsu b_x and the adjacent bunsetsu b_{x-1} by using the maximum entropy method. The maximum entropy method estimates the probability that a linefeed should be inserted at a point by considering such information as morphological analysis results, bunsetsu boundaries, clause boundaries, and dependency structures. When the probability is larger than 0.5, our method decides to insert a linefeed at the point.

The features used in the ME method, which estimates the probability that a linefeed is inserted between a bunsetsu b_{x-1} and b_x, are as roughly the same as those for the conventional sentence-based method (Ohno et al., 2009)[6]. The difference is that our method does not use the following three features among those for the sentence-based method:

- whether or not b_{x-1} depends on the final bunsetsu of a clause
- whether or not b_{x-1} depends on a bunsetsu to

[5]We used the same features in Section 3 when estimating the probability of the relationship between two bunsetsus.

[6]The sentence-based method decides the most appropriate linefeed insertion points in the whole of a sentence after the sentence-end bunsetsu is inputted.

	recall	precision	f-measure
our method	79.6% (4,375/5,497)	70.2% (4,375/6,228)	74.6%
conv. method	73.7% (4,052/5,497)	74.4% (4,052/5,447)	74.0%

Table 2: Experimental results of linefeed insertion

which the number of characters from the start of the line is less than or equal to the maximum number of characters (20 characters)

- whether or not there exists a bunsetsu which depends on the modified bunsetsu of b_{x-1}, among bunsetsus which are located after b_{x-1} and to which the number of characters from the start of the line is less than or equal to the maximum number of characters

The values of these features can not be decided until the modified bunsetsu of b_{x-1} is inputted if b_{x-1} does not depend on b_x. Since our sequential linefeed insertion method needs to identify the linefeed insertion points simultaneously with speech input, our method does not use these features.

5.2 Experiment on linefeed insertion

To evaluate the application potency of the dependency structure proposed in Section 2, we conducted an experiment on linefeed insertion using the sequential linefeed insertion method described in the previous section.

5.2.1 Experimental outline

We used the same experimental data and performed a cross-validation experiment in the same way as Section 4.1. However, when inputting the test data, we eliminated the information on clause boundaries, dependency, and linefeeds. In the evaluation, we obtained the recall, precision and f-measure on the linefeed insertion performance.

For comparison, we also performed linefeed insertion using the conventional sentence-based method (Ohno et al., 2009). Here, when performing the conventional method, clause boundary information and dependency information were beforehand provided using Clause Boundary Annotation Program (CBAP) and Uchimoto et al.'s dependency parsing method (Uchimoto et al., 2000b), respectively.

5.2.2 Experimental results on linefeed insertion

Table 2 shows the recalls, precisions and f-measures of our method and conventional method. The f-measure for our method was 74.6%, which was slightly larger than that for the conventional method. We confirmed that our method can output linefeed insertion points simultaneously with speech input, maintaining the approximately-same f-measure of linefeed insertion as that for the conventional method.

However, we measured the sentence accuracy, which is the ratio of sentences of which all linefeed insertion points were correctly detected. The sentence accuracies of our method and the conventional method were 35.8% (614/1,714) and 46.2% (792/1,714) respectively. The conventional method decides the most appropriate linefeed insertion points by considering the whole sentence while our method judges whether a linefeed should be inserted or not whenever a bunsetsu is inputted. This difference is thought to have caused the result. We will confirm that the readability of captions generated by our method does not decrease, by conducting not only objective evaluation based on comparing the linefeed insertion result with the correct data but also subjective evaluation.

6 Conclusion

This paper proposed the dependency structure which an incremental dependency parser outputs for an inputted bunsetsu sequence. Our proposed dependency structure makes a parser clarify the fact that the modified bunsetsu is a bunsetsu which will be inputted later. In addition, we proposed a method for incremental dependency parsing to output our proposed dependency structure. As the result of an experiment on incremental parsing, we confirmed the feasibility of our method. Furthermore, we applied our proposed incremental dependency parsing to the simultaneous linefeed insertion, and then we confirmed the application potency of our proposed dependency structure.

Acknowledgments

This research was partially supported by the Grant-in-Aid for Young Scientists (B) (No.25730134) and Scientific Research (B) (No.22300051) of JSPS.

References

Christian Fügen, Alex Waibel, and Muntsin Kolss. 2007. Simultaneous translation of lectures and speeches. *Machine Translation*, 21:209–252. 10.1007/s10590-008-9047-0.

Shinya Fujie, Kenta Fukushima, and Tetsunori Kobayashi. 2004. A conversation robot with back-channel feedback function based on linguistic and nonlinguistic information. In *Procedings of the 2nd International Conference on Autonomous Robots and Agents (ICARA2004)*, pages 379–384.

Richard Johansson and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL2007)*, pages 1134–1138.

Yuki Kamiya, Tomohiro Ohno, and Shigeki Matsubara. 2010. Coherent back-channel feedback tagging of in-car spoken dialogue corpus. In *Proceedings of the 11th Annual SIGdial Meeting on Discourse and Dialogue (SIGDIAL2010)*, pages 205–208.

Hideki Kashioka and Takehiko Maruyama. 2004. Segmentation of semantic units in Japanese monologues. In *Proceedings of International Conference on Speech and Language Technology and Oriental COCOSDA (ICSLT/O-COCOSDA2004)*, pages 87–92.

Yoshihide Kato, Shigeki Matsubara, Katsuhiko Toyama, and Yasuyoshi Inagaki. 2005. Incremental dependency parsing based on headed context-free grammar. *Systems and Computers in Japan*, 36:63–77.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the 6th Conference on Natural Language Learning 2002 (CoNLL 2002)*, pages 63–69.

Shigeki Matsubara, Akira Takagi, Nobuo Kawaguchi, and Yasuyoshi Inagaki. 2002. Bilingual spoken monologue corpus for simultaneous machine interpretation research. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC2002)*, pages 153–159.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Tomohiro Ohno, Masaki Murata, and Shigeki Matsubara. 2009. Linefeed insertion into Japanese spoken monologue for captioning. In *Proceedings of Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing (ACL-IJCNLP2009)*, pages 531–539.

Matthias Paulik and Alex Waibel. 2010. Spoken language translation from parallel speech audio: simultaneous interpretation as SLT training data. In *Proceedings of the 2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP2010)*, pages 5210–5213.

Koichiro Ryu, Shigeki Matsubara, and Yasuyoshi Inagaki. 2006. Simultaneous English-Japanese spoken language translation based on incremental dependency parsing and transfer. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-2006)*, pages 683–690.

Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, and Hitoshi Isahara. 2000a. Dependency model using posterior context. *Journal of Natural Language Processing*, 7(5):3–18. (In Japanese).

Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, and Hitoshi Isahara. 2000b. Dependency model using posterior context. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT2000)*.

Le Zhang. 2013. Maximum entropy modeling toolkit for Python and C++ (online). `http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html` (accessed 2013-08-21).

Active Learning for Dependency Parsing by A Committee of Parsers

Saeed Majidi
Department of Computer Science
Tufts University
Medford, MA USA
saeed.majidi@tufts.edu

Gregory Crane
Perseus Project
Tufts University
Medford, MA USA
gregory.crane@tufts.edu

Abstract

Data-driven dependency parsers need a large annotated corpus to learn how to generate dependency graph of a given sentence. But annotations on structured corpora are expensive to collect and requires a labor intensive task. Active learning is a machine learning approach that allows only informative examples to be selected for annotation and is usually used when the number of annotated data is abundant and acquisition of more labeled data is expensive. We will provide a novel framework in which a committee of dependency parsers collaborate to improve their efficiency using active learning techniques. Queries are made up only from uncertain tokens, and the annotations of the remaining tokens of selected sentences are voted among committee members.

1 Introduction

Emerging digital libraries must manage not only surrogates for traditional publications (such as PDF and HTML files) but the data-sets associated with, and sometimes derived from, these traditional publications. Linguistic annotations such as part of speech and syntactic function of the words are increasingly important for students of language. These annotations are, however, expensive to collect in the best case when native speakers are readily available. Developing such databases of annotations for historical languages, where no native speakers are alive and where few have even developed advanced language skills; the process becomes even more expensive. This paper describes how such linguistic data can be extracted automatically and lays the foundations for smart digital libraries that can not only offer preservation and access, but also generate fundamental data.

In the occasions that the number of annotated data is abundant and acquisition of more labeled data is expensive, using active learning techniques is one of the promising approaches. Active learning is a machine learning approach that allows only informative examples to be selected for labeling. The main idea of utilizing active learning is that the learner can achieve better performance with fewer training data if it can choose the examples it needs to learn from in an intelligent manner.

Active learning has been successfully applied to many of natural language processing applications such as information extraction (Jones et al., 2003; Culotta et al., 2006), named entity recognition (Kim et al., 2006; Velachos, 2006; Laws and Schütze, 2008), part of speech tagging (Argamon-Engelson and Dagan, 1999; Ringger et al., 2007), text classification (Schohn and Cohn, 2000; Tong and Koller, 2002; Hoi et al., 2006), word segmentation (Sassano, 2002) and word-sense disambiguation (Zhu and Hovy, 2007).

In this paper we investigate active learning techniques that can be used for dependency parsing to help us reach better performance with cheaper annotation cost. The main motivation of this work is, as McDonald and Nivre showed in (McDonald and Nivre, 2007), different parsers generate different models that produce different types of errors. Hence, when two or more parsers agree about annotation of a token it is not worth to spend a budget to know its true annotation. We will provide a novel framework in which a committee of dependency parsers collaborate to improve their efficiency using active learning techniques. In each round of annotation, the committee of parsers select a few tokens they disagree most from uncertain selected sentences. An expert then annotates the chosen tokens, and the committee members vote about the annotations of the rest of the tokens of the selected sentences.

The rest of this paper is structured as follows. We briefly review active learning and pool-based sampling as a commonly used framework of active learning in section 2. We also talk about different querying scenarios for pool-based active learning. In section 3, we bring a literature survey of applications of active learning for the task of dependency parsing. We also introduce an uncertainty sampling framework with a different confidence measure as a baseline to compare and evaluate our methods. We introduce our proposed methods of committee-based active learning for dependency parsing in section 4. The experimental results are presented in section 5 and show that we can gain better results using a committee-based strategy. We conclude this paper at section 6 with a brief summary of the findings and an outline of the future work.

2 Active Learning

Active learning is one of the mostly used applied machine learning techniques. In an active learning framework, the learner is able to ask an oracle, a domain expert, about the annotations of only those unlabeled instances that could help it to improve itself.

Pool-based sampling (Lewis and Gale, 1994) is one of the basic scenarios for active learning. It is usually used when there is a small set of labeled data $\mathcal{L}$ and a large pool of unlabeled data $\mathcal{U}$. In each round, instances are chosen from $\mathcal{U}$ according to a selection strategy, and their labels are asked from an oracle. Then, the labeled instances are added to $\mathcal{L}$. This process is repeated until there are enough annotated data at $\mathcal{L}$.

At the heart of each active learning scenario there is a *query strategy* that formulates how the next instances should be selected to be queried for their labels. There are different query strategies in the literature. Two of commonly used query strategies are *uncertainty sampling* and *query-by-committee*.

The simplest and most commonly used method for selecting samples to be annotated is uncertainty sampling (Lewis and Gale, 1994). In uncertainty sampling, after training a model using existing labeled data in $\mathcal{L}$, it is used to predict the labels of instances in $\mathcal{U}$. Then the learner selects the instances about whose labels it is less confident.

Another approach for sample selection is query-by-committee (Sebastian Seung and Sompolinsky, 1992) (QBC). In a QBC framework, the system contains a committee of c competing models $M = \theta^1, \theta^2, \ldots, \theta^c$ all trained on the current labeled data set $\mathcal{L}$. Then each of the committee members votes on the label of examples in $\mathcal{U}$. The query contains the samples about which committee members most disagree.

We refer the interested reader to look at Settels (Settles, 2009) for a comprehensive survey of active learning in general and to Olsson (Olsson, 2009) for a literature survey of active learning in the context of natural language processing.

3 Active Learning for Dependency Parsing

There are two common approaches for the selection of samples to be queried: *sentence-wise* sample selection and *token-wise* sample selection.

In sentence-wise sample selection, the learner selects full sentences that it is not confident about their parsing to be annotated. Tang et al. (Tang et al., 2002) and Hwa (Hwa, 2004) use sentence entropy, calculated based on the n-best parses of a sentence S, to measure the uncertainty of each sentence. The k uncertain sentences are selected for the annotation. Lynn et al. (Lynn et al., 2012) use QBC-based active learning for bootstrapping a dependency bank at the sentence level.

Another approach is to select only some parts of a sentence for the annotation rather than full sentences. Sassano and Kurohashi (Sassano and Kurohashi, 2010) use the score that parser assigns to dependencies to select uncertain dependencies for the task of Japanese dependency parsing. They utilize syntactic constraints of Japanese to decide about annotations of the rest of dependencies. Mirroshandel and Nasr (Mirroshandel and Nasr, 2011) use a combined active learning strategy to select uncertain tokens for the annotation. They first calculate each sentence entropy based on the n-best scores that the parser generates for each sentence. After selecting k most uncertain sentences, they calculate the attachment entropy for every token of each sentence. Given the n-best parse of a sentence S, they compute the attachment entropy of token w based on the number of its possible governors in the n-best parse. They use parser annotations for the rest of tokens of selected sentences.

We use the same idea of selecting uncertain sentences first and then choosing some parts of those sentences for the annotation. But the contribution of our work is that unlike (Mirroshandel and Nasr, 2011) work, we use a committee of parsers rather than only one parser. The parsers of committee collaborate with each other to choose the next set of tokens to be queried, and decide about the annotation of remaining set of tokens. We also use a different query method and uncertainty measure to select the tokens. In this paper we extend our approach in (Majidi and Crane, 2013). We provide a solid algorithm and investigate the effect of number of selected tokens for query.

3.1 Uncertainty Sampling Word Selection

We set *uncertainty sampling word selection* as a baseline to compare our work with it. The confidence measure that we use for uncertainty sampling is KD-Fix, K-draws with fixed standard deviation, proposed in (Mejer and Crammer, 2011). KD-Fix is a stochastic method to generate K alternatives for the best labeling. Given a model parameter μ learned by the parser, a Gaussian probability distribution with an isotropic covariance matrix, $\Sigma = sI$, is defined over that, $w \sim \mathcal{N}(\mu, \Sigma)$. Then a set of K weight vectors w_i are drawn, and each one outputs a single alternative labeling. If $y_w^{(i)}, i = 1, \ldots, K$ be the K alternative labeling generated for token w and $\hat{y}_w$ be the actual predicted label by the model, the confidence in the label $\hat{y}_w$ is defined as :

$$\nu_w = \frac{\left|\{i : \hat{y}_w = y_w^{(i)}\}\right|}{K}$$

Along parsing the sentences in $\mathcal{U}$ we have the parser to generate the confidence score for each edge that shows the parser confidence on the correctness of that edge (token). We assign the confidence score of each sentence as the average confidence score of its tokens. Then we rank the sentences of $\mathcal{U}$ based on the computed confidence score and select k sentences with least confidence score. For each of these k sentences we choose l tokens with the least confidence score among the tokens of that sentence. We ask the expert to annotate the selected tokens (head and relation), and the trained parser annotates rest of the tokens. Finally these k annotated sentences are added to $\mathcal{L}$.

4 Committee-Based Active Learning for Dependency Parsing

We use a committee of c parsers in the active learning framework. Each parser is first trained, using labeled pool $\mathcal{L}$, and then predicts the head and the relation to the head of every instance in unlabeled pool $\mathcal{U}$. We show the head prediction of parser P_i for token w as $h(P_i, w)$, and its relation prediction as $r(P_i, w)$. Here we assume that the head and relation of token w are predicted independently. The trained parsers parse a separate test set $\mathcal{T}$ and their parsing accuracy on the test set is computed. $UA(P_i)$ shows the unlabeled accuracy of parser P_i and $LA(P_i)$ shows its labeled accuracy. *Unlabeled accuracy* is the percentage of correct head dependencies, and *labeled accuracy* is the percentage of correct relations for the correct dependencies that the parser predicts.

To select the next tokens to be included in the query, two entropy measures, $HE(w)$ and $RE(w)$, are computed as the confidence score of each token w:

$$HE(w) = -\sum \frac{V(h_j, w)}{\sum_{i=1}^{c} UA(P_i)} \cdot \log \frac{V(h_j, w)}{\sum_{i=1}^{c} UA(P_i)}$$

$HE(w)$ is the head vote entropy of token w in which h_j varies over all of the head values assigned to token w and $V(h_j, w)$ is the number of votes assigned to h_j as the head of token w:

$$V(h_j, w) = \sum_{\forall i, h_j = h(P_i, w)} UA(P_i)$$

In a same way, we calculate the relation vote entropy for token w:

$$RE(w) = -\sum \frac{U(r_j, w)}{\sum_{i=1}^{c} LA(P_i)} \cdot \log \frac{U(r_j, w)}{\sum_{i=1}^{c} LA(P_i)}$$

$$U(r_j, w) = \sum_{\forall i, r_j = r(P_i, w)} LA(P_i)$$

$RE(w)$ is the relation vote entropy of token w and r_j varies over all of the relations assigned to token w. $U(r_j, w)$ is the number of votes assigned to r_j as the relation of token w to its head.

The entropy measure of token w is computed as the mean value of the head entropy and relation entropy:

$$WE(w) = \frac{HE(w) + RE(w)}{2}$$

Finally, for a sentence S we assign the average entropy value of all its tokens as the sentence entropy:

$$SE(S) = \sum_{i=1}^{n} \frac{WE(w_i)}{n}$$

4.1 Single Parser Prediction

After computing the confidence score of each token and sentence in $\mathcal{U}$, we select k sentences with most entropy value. For each sentence we ask the expert to annotate l tokens that has the highest entropy. Here, we select one of the parsers as the main parser and use it to annotate the rest of the words of those k selected sentences and add them all to $\mathcal{L}$.

4.2 Weighted Committee Prediction

In this section we use the same querying strategy as the previous one to select l tokens of k sentences with higher entropy value to be annotated by the expert. But for predicting the labels of the rest of the words, we run a majority voting among the parsers. The vote of each parser P_i for predicting the head and relation of each token is weighted by its labeled and unlabeled accuracy, $LA(P_i)$ and $UA(P_i)$, and the label of each token is set to the one that has majority of votes among parsers. Algorithm 1 shows the QBC active learning for dependency parser.

Algorithm 1 QBC Active Learning for Dependency Parsing with Committee Prediction

```
$\mathcal{L} \leftarrow$ Initial labeled training set
$\mathcal{U} \leftarrow$ Unlabeled pool
$\mathcal{T} \leftarrow$ Test set
$C \leftarrow P_1, \ldots, P_c$ // A committee of $c$ parsers
repeat
    $W \leftarrow [\,]$ // Weight vector
    for $\forall P_i \in C$ do
        $P_i \leftarrow$ train($\mathcal{L}$)
        $\mathcal{U}'_i \leftarrow$ parse($\mathcal{U}, P_i$)
        $\mathcal{T}'_i \leftarrow$ parse($\mathcal{T}, P_i$)
        $w_i \leftarrow$ Calculate weight of $P_i$ given $\mathcal{T}$ and $\mathcal{T}'_i$
        $W \leftarrow W \cup w_i$
    Calculate confidence score of each token of $\mathcal{U}'$ given $W$
    $\mathcal{S} \leftarrow k$ least confident sentences of $\mathcal{U}'$
    $\mathcal{I} \leftarrow \{\}$
    for $\forall s \in \mathcal{S}$ do
        Query the expert $l$ least confident tokens of $s$
        Vote among parsers for annotations of the rest of tokens of $s$
        Add new annotated sentence to $\mathcal{I}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{I}$
    $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{S}$
until $\mathcal{U}$ is empty or some stopping criteria is not met
return $C, \mathcal{L}$
```

5 Experiments

To evaluate the proposed method, we set up 5 different experiments. In the first two ones, we select the tokens that should be annotated randomly. In one case we first select sentences randomly from unlabeled pool, and then we choose some random tokens of each selected sentence and ask for their labels. The trained parser annotates the rest of the words. In another case, we have a committee of parsers that vote about the annotations of the rest of the words of sentences that should be added to the labeled pool. In the third experiment, we try the query by uncertainty sampling with the confidence measure that we discussed earlier in 3.1. We set up these three experiments as baselines with which we compare the proposed method. The last two experiments are related to the QBC active learning. In one case we use only one of the parsers of the committee to predict the rest of the tokens of selected sentences that we do not choose for expert annotation. The latter case is an implementation of algorithm 1. All parsers of committee decide together about the annotation of those tokens. The votes of each parser are weighted by its accuracy on the test set.

We use MSTParser (McDonald et al., 2005) as the main parser in each experiment that only needs one parser. To set up the QBC experiments, we make a committee of three parsers including MSTParser, Mate parser (Bohnet, 2010) and DeSR parser (Attardi, 2006).

5.1 Corpora

In our experiments we use data sets from Ancient Greek Dependency Treebank (Bamman et al., 2009) and TUT corpora (Bosco et al., 2000). The Ancient Greek Dependency Treebank (AGDT) is a dependency-based treebank of literary works of the

Data set	Language	# Of Sentences	# Of Tokens	Avg Sent Length
Sophocles	Ancient Greek	3307	39891	12.06
Wiki	Italian	459	14747	32.12

Table 1: Corpora used in the experiments.

Archaic and Classical age published by Perseus. It includes 13 texts of five Greek scholars from which we select Sophocles' works. TUT corpora has been organized in five sections. Here we use the Wiki section that includes 459 sentences, randomly chosen from the Italian version of the Wikipedia. Table 1 shows the number of sentences and tokens for both data sets.

5.2 Experimental Setup

In each experiment we divide whole sentences of a text in two random training and test sets. Training set has 75% of the sentences of the text. We also divide the training set to two pools $\mathcal{L}$ and $\mathcal{U}$. Initially we put 10% of training data in $\mathcal{L}$ as the labeled pool and the rest go to unlabeled pool $\mathcal{U}$. In each iteration we select 10% of unlabeled sentences in initial training set from $\mathcal{U}$ and after having their annotations add them to $\mathcal{L}$. For every text, we replicate each experiment for 10 different random seeds.

5.3 Experimental Results

Figure 1 shows the learning curve for unlabeled dependency parsing of the Wiki data set when 10 tokens per sentence are selected for annotation. x arrow grows with the number of tokens in training set, and y arrow shows the test set accuracy[1] that the main parser, MSTParser, can achieve[2]. We can observe that the methods which use an active query strategy do a better job than those methods which are based on random selection strategy. Among active learning methods, QBC strategy works better than the rest. One explanation could be the way that the queries are selected. Since each parser generates a different model, they can make different types of errors. The selected query by the committee is the one that most of the members have difficulty on that, and hence knowing its label is more informative. We run one-tailed, paired t-tests to test if these differences are statistically significant. The t-tests run for the best performance that each of the methods can achieve after the final loop of active learning. Table 2 shows the p-values for the case that 10 tokens of each sentence is selected.

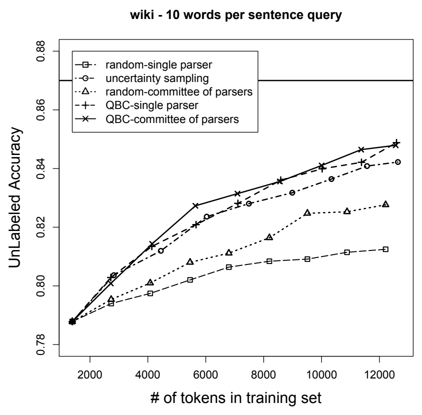


Figure 1: Learning curves for dependency parsing of the Wiki data set. 10 words per sentence are selected. The solid horizontal line is the unlabeled accuracy of parser on the fully annotated training set.

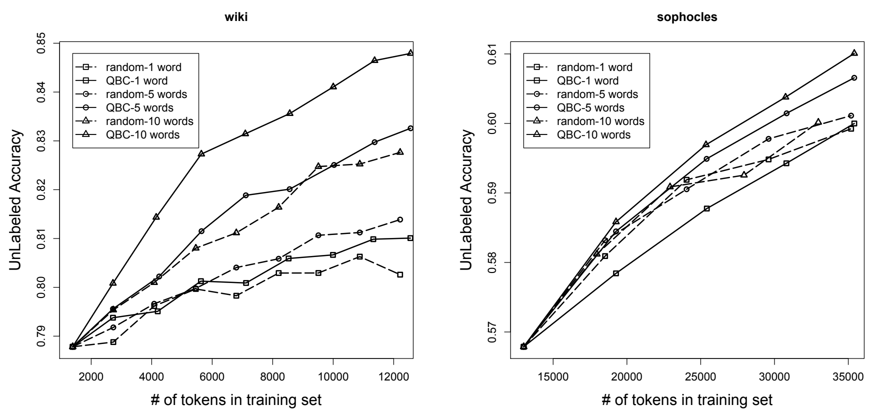
Method	p-value
Random single parser	0.0014
Random committee of parsers	0.03
Uncertainty sampling	0.002
QBC single parser	0.0006

Table 2: p-values of paired t-tests to compare QBC-committee of parsers with other methods.

The number of selected tokens, variable l in algorithm 1, has a direct effect on the performance that we get. To investigate how many tokens we need to select, we plot the learning curves for both the Wiki and Sophocles when different number of tokens have been selected. Figure 2 shows the learning curves of unlabeled dependency parsing for Wiki and Sophocles. It compares QBC and random selection scenarios when 1, 5, and 10 words are selected. Solid lines depict QBC strategy and dashed ones show random selec-

[1]The percentage of correct head dependency predictions

[2]Our experiments on Sophocles show the same behavior. But due to the lack of space we do not report them here.



(a) Wiki (b) Sophocles

Figure 2: Comparing learning curves of unlabeled dependency parsing for QBC active learning (solid lines) and random selection (dashed lines) for different number of selected tokens per sentence (1, 5, and 10).

tion. One can see that when we only select one token per sentence, both active learning and random selection strategies perform almost the same. When we increase the number of selected tokens to 5 and then 10, we observe that for Sophocles active learning approach can achieve better than random selection. For the Wiki, selecting 10 tokens randomly is almost the same as selecting 5 tokens with active learning.

One reason could be the length of the sentences in each data set. As we can see in table 1, the average sentence length in the Wiki is as twice as the average sentence length of Sophocles. Table 3 reports the percent of the expert's annotated tokens that we have in the training set after the final loop of active learning. When only 1 token per sentence is selected in each round, only less than 10% of total tokens in final training set have gold standard label. Therefore one should not expect that the active learning approach performs better than random selection. For the Wiki data set that has long sentences, 32 tokens per sentence in average, when 5 tokens from each uncertain sentence are selected, we finally reach a point that only 15% of tokens have gold standard label and the random selection of 10 tokens is doing better than that. But as Sophocles has smaller number of tokens per sentence, 12 tokens per sentence in average, selecting 5 uncertain tokens from each sentence will lead us to a point that finally more than 40% of tokens in the training set have gold standard label, and hence active learning has better performance even better than the case that 10 tokens per sentence are selected randomly.

Data set	1-token	5-token	10-token
Sophocles	8%	41%	83%
Wiki	3%	15%	31%

Table 3: Percentage of tokens in the final training set annotated by an expert.

6 Conclusions and Future Work

We have set up an active learning framework with a committee of dependency parsers. The experimental results show that using a committee of parsers, we can reach better accuracy with less cost of annotation than the case where there is only one parser with uncertainty sampling.

We are currently working on a model that instead of a single oracle we have a committee of experts with different levels of expertise. We want to build a model to combine the annotation of those experts together and send that feedback for the parser.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS-0910165 and by a Tufts Collaborates 2011 seed grant. Thanks are also due to Bridget Almas and Alison Babeu for their invaluable research assistance.

References

Shlomo Argamon-Engelson and Ido Dagan. 1999. Committee-based sample selection for probabilistic classifiers. *Journal of Artificial Intelligence Research*, 11:335–360.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 166–170. Association for Computational Linguistics.

David Bamman, Francesco Mambrini, and Gregory Crane. 2009. An ownership model of annotation: The ancient greek dependency treebank. In *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories*, pages 5–15.

Bernd 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. Coling 2010 Organizing Committee, August.

Cristina Bosco, Vincenzo Lombardo, Daniela Vassallo, and Leonardo Lesmo. 2000. Building a treebank for italian: a data-driven annotation schema. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 99–105.

Aron Culotta, Trausti Kristjansson, Andrew McCallum, and Paul Viola. 2006. Corrective feedback and persistent learning for information extraction. *Artificial Intelligence*, 170(14):1101–1122, October.

Steven C. H. Hoi, Rong Jin, and Michael R. Lyu. 2006. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th international conference on World Wide Web*, pages 633–642. ACM.

Rebecca Hwa. 2004. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276, September.

Rosie Jones, Rayid Ghani, Tom Mitchell, and Ellen Rilo. 2003. Active learning for information extraction with multiple view feature sets. In *the ECML workshop on Adaptive Text Extraction and Mining*.

Seokhwan Kim, Yu Song, Kyungduk Kim, Jeong won Cha, and Gary Geunbae Lee. 2006. Mmr-based active machine learning for bio named entity recognition. In *Proceedings of the Human Language Technology Conference/North American chapter of the Association for Computational Linguistics annual meeting*, pages 69–72. Association for Computational Linguistics.

Florian Laws and Hinrich Schütze. 2008. Stopping criteria for active learning of named entity recognition. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, pages 465–472. Association for Computational Linguistics.

David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. ACM/Springer.

Teresa Lynn, Jennifer Foster, Elaine Ui Dhonnchadha, and Mark Dras. 2012. Active learning and the irish treebank. *The Annual Meeting of the Australasian Language Technology Association (ALTA 2012)*.

Saeed Majidi and Gregory Crane. 2013. Committee-based active learning for dependency parsing. In *Research and Advanced Technology for Digital Libraries*, pages 442–445. Springer.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Natural Language Learning*. Association for Computational Linguistics, June.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Avihai Mejer and Koby Crammer. 2011. Confidence estimation in structured prediction. *CoRR*, abs/1111.1386.

Seyed Abolghasem Mirroshandel and Alexis Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *the 12th International Conference on Parsing Technologies*, pages 140–149. Association for Computational Linguistics, October.

Fredrik Olsson. 2009. A literature survey of active machine learning in the context of natural language processing. Computer sciences technical report, Swedish Institute of Computer Science.

Eric Ringger, Peter McClanahan, Robbie Haertel, George Busby, Marc Carmen, James Carroll, Kevin Seppi, and Deryle Lonsdale. 2007. Active learning for part-of-speech tagging: Accelerating corpus annotation. In *Proceedings of the Linguistic Annotation Workshop*, pages 101–108. Association for Computational Linguistics, June.

Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for japanese dependency parsing. In *the 48th*

Annual Meeting of the ACL, pages 356–365. Association for Computational Linguistics, July.

Manabu Sassano. 2002. An empirical study of active learning with support vector machines for japanese word segmentation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 505–512. Association for Computational Linguistics.

Greg Schohn and David Cohn. 2000. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 839–846.

Manfred Opper Sebastian Seung and Haim Sompolinsky. 1992. Query by committee. In *the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 287–294. ACM.

Burr Settles. 2009. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Min Tang, Xaoqiang Luo, and Salim Roukos. 2002. Active learning for statistical natural language parsing. In *the 40th Annual Meeting of the Association for Computational Linguistics*, pages 120–127. Association for Computational Linguistics, July.

Simon Tong and Daphne Koller. 2002. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March.

Andreas Velachos. 2006. Active annotations. In *the ECML workshop on Adaptive Text Extraction and Mining*, pages 64–71. Association for Computational Linguistics.

Jingbo Zhu and Eduard H Hovy. 2007. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *EMNLP-CoNLL*, volume 7, pages 783–790.

Development of Amharic Grammar Checker Using Morphological Features of Words and N-Gram Based Probabilistic Methods

Aynadis Temesgen
Department of Computer Science
Addis Ababa University
temesgen.aynadis@gmail.com

Yaregal Assabie
Department of Computer Science
Addis Ababa University
yaregal.assabie@aau.edu.et

Abstract

Amharic is one of the most morphologically complex and under-resourced languages which effectively hinder the development of efficient natural language processing applications. Amharic words, especially verbs, are marked for a combination of several grammatical functions which makes grammar checking complex. This paper describes the design and development of statistical grammar checker for Amharic by treating its morphological features. In a given Amharic sentence, the morphologies of individual words making up the sentence are analyzed and then n-gram based probabilistic methods are used to check grammatical errors in the sentence. The system is tested with a test corpus and experimental results are reported.

1 Introduction

With the rise of electronic documents, the need of natural language processing (NLP) applications that automatically process texts has drastically increased. One of such important NLP applications is grammar checker which automatically checks grammatical errors in texts and also possibly suggests the user to choose among other alternatives. Initially, most of the grammar checkers were based on checking styles, uncommon words and sentence structures, but now they are upgraded to high capacity with the capability of analyzing complex sentence structures, not only as a part of other programs but also as easy software to be installed in many operating system (Richardson, 1997; Liddy, 2001; Mudge, 2010; Mozgovoy, 2011). Various techniques and methods have been proposed so far to build systems that could check the grammars of texts. Among the most widely used approaches to implement grammar checkers are *rule-based*, *statistical* and *hybrid* (Tsuruga and Aizu, 2011; Ehsan and Faili, 2013; Xing *et al*, 2013). Rule-based systems check grammars based on a set of manually developed rules which are used to match against the text. However, it is very difficult to understand and include all grammatical rules of languages, especially for complex sentences. On the other hand, in statistical grammar checking, part-of-speech (POS)-annotated corpus is used to automatically build the grammatical rules by identifying the patterns of POS tag sequences in which case common sequences that occur often can be considered correct and the uncommon ones are reported to be incorrect. This has lead statistical approaches to become popular methods to build efficient grammar checkers. However, it is very difficult to understand error messages suggested by such checking system as there is no specific error message. Hybrid grammar checking is then introduced to benefit from the synergy effect of both approaches (Xing *et al*, 2013). A number of grammar checkers have been developed so far for many languages around the world. Among the most notable grammar checkers are those developed over the past few years for resourceful languages such as English (Richardson, 1997; Naber, 2003), Swedish (Arppe, 2000; Domeij *et al*, 2000), German Schmidt-Wigger, (1998), and Arabic (Shaalan, 2005), etc. However, to our best knowledge, there is no commercial Amharic grammar checker or published article that presents grammar checking for Amharic.

This paper presents statistical-based Amharic grammar checker developed by treating the morphological features of the language. The organization of the remaining part of the paper is as follows. Section 2 discusses an overview of the grammatical structure of Amharic. Section 3 presents the statistical methods applied to develop the system. Experimental results are presented in Section 4. In Section 5, we present our conclusion and recommendation for future works. A list of references is provided at the end.

2 Grammatical Structure of Amharic

2.1 Amharic Morphology

Amharic is the working language of Ethiopia having a population of over 90 million at present. Even though many languages are spoken in Ethiopia, Amharic is the dominant language that is spoken as a mother tongue by a large segment of the population and it is the most commonly learned second language throughout the country (Lewis *et al*, 2013). Amharic is written using its own script which has 33 consonants (base characters) out of which six other characters representing combinations of vowels and consonants are derived for each character.

Amharic is one of the most morphologically complex languages. Amharic nouns and adjectives are marked for any combination of number, definiteness, gender and case. Morover, they are affixed with prepositions. For example, from the noun ተማሪ (*tämari*/student), the following words are generated through inflection and affixation: ተማሪዎች (*tämariwoč*/students), ተማሪው (*tämariw*/ the student {masculine}/his student), ተማሪየ (*tämariyän*/my student), ተማሪየን (*tämariyän*/my student {objective case}), ተማሪሽ (*tämariš*/your {feminine} student), ለተማሪ (*lätämari*/for student), ከተማሪ (*kätämari*/ from student), etc. Similarly, we can generate the following words from the adjective ፈጣን (*fäṭan*/fast): ፈጣኑ (*fäṭanu/*fast, {definite} {masculine} { singular}), ፈጣኖች (*fäṭanoč/fast* {plural}), ፈጣኖቹ (*fäṭanoču/*fast {definite} {plural}), etc.

Amharic verb inflections and derivations are even more complex than those of nouns and adjectives. Several verbs in surface forms are derived from a single verbal stem, and several stems in turn are derived from a single verbal root. For example, from the verbal root ውስድ (*wsd*/to take), we can derive verbal stems such as *wäsd*, *wäsäd*, *wasd*, *wäsasd*, *täwäsasd*, etc. From each of these verbal stems we can derive many verbs in their surface forms. For example, from the stem *wäsäd* the following verbs can be derived:

ወሰደ (*wäsädä*/he took)
ወሰደች (*wäsädäč*/she broke)
ወሰድኩ (*wäsädku*/I broke)
ወሰድኩት (*alwäsädkutĭm*/I took [it/him])
አልወሰድኩም (*alwäsädkum*/I didn’t take)
አልወሰደችም (*alwäsädäčĭm*/she didn’t take)
አልወሰደም (*alwäsädäm*/he didn’t take)
አልወሰደኝም (*alwäsädäňĭm*/he didn’t take me)
አስወሰደ (*aswäsädä*/he let [someone] to take)
ተወሰደ (*täwäsädä*/[it/he] was taken)
ስለተወሰደ (*sĭlätäwäsädä*/as [it/he] was taken)
ከተወሰደ (*kätäwäsädä*/ if [it/he] is taken)
እስኪወሰድ (*ĭskiwäsäd*/until [it/he] is taken)
ሲወሰድ (*sĭwäsäd*/when [it/he] is taken)
:
etc.

Amharic verbs are marked for any combination of person, gender, number, case, tense/aspect, and mood resulting in thousands of words from a single verbal root. As a result, a single word may represent a complete sentence cosutructed with subject, verb and object. For example, ይወስደኛል (*yĭwäsdäňal*/[he/it] will take me) is a sentence where the verbal stem ወስድ (*wäsd*/ will take) is marked for various grammatical functions as shown in Figure 1.

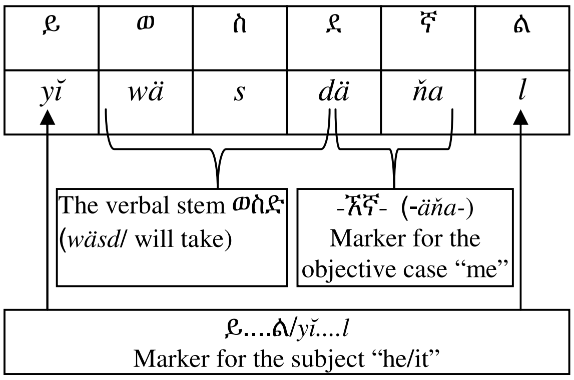

ይ ወ ስ ደ ኛ ል
yĭ wä s dä ňa l
The verbal stem ወስድ (wäsd/ will take)
-ኧኛ- (-äňa-) Marker for the objective case “me”
ይ....ል/yĭ....l
Marker for the subject “he/it”

Figure 1. Morphology of the word ይወስደኛል.

2.2 Grammatical Rules of Amharic

Common for most languages, if not for all, grammar checking starts with checking the validity of the sequence of words in the sentence. This is also true for Amharic. In addition, since Amharic is morphologically complex language where verbs, nouns and adjectives are marked for various grammatical functions, the following agreements are required to be checked: adjective-noun, adjective-verb, subject-verb, object-verb, and adverb-verb (Yimam, 2000; Amare, 2010).

Word Sequence: Amharic language follows subject-object-verb (SOV) grammatical pattern as opposed to, for example, English language which has SVO sequence of words. For instance, the Amharic equivalent of sentence “John ate bread” is written as “ጆን (*jon*/*John*) ዳቦ (*dabo*/bread) በላ (*bäla*/ate)”. Here, the part-of-speech (POS) tags of individual words are used as inputs to check the validity of grammatical patterns.

Adjective-Noun Agreement: Amharic nouns are required to agree for number of modifying adjectives. For example, ረጃጅም ልጆች (*räjajĭm lĭjoč*/ tall {plural} children) is a valid noun phrase whereas ረጃጅም ልጅ (*räjajĭm lĭj*/ tall {plural} child) is an invalid noun phrase construction.

Subject-Verb Agreement: Amharic verbs are marked for number, person and gender of subjects. For example, ልጆቹ መስኮት ሰበሩ (*lĭjoču mäskot säbäru*/the children broke a window) is a valid Amharic sentence. However, ልጅቷ መስኮት ሰበረ (*lĭjt*^{w}*a mäskot säbärä*/the girl broke {masculine} a window) is not a valid Amharic sentence since the subject ልጅቷ (*lĭjt*^{w}*a*/the girl) is feminine and the verb ሰበረ (*säbärä* /broke {masculine}) is marked for masculine.

Object-Verb Agreement: Amharic verbs are also marked for number, person and gender of objective cases. For example, in the sentence ልጆቹ መስኮቶቹን ሰበሯቸው (*lĭjoču mäskotočun säbär*^{w}*a-čäw*/the children broke {plural} the windows), the verb ሰበሯቸው (*säbäru*/broke {plural}) is marked for the plural property of the object መስኮቶቹን (*mäskotočun*/the windows).

Adverb-Verb Agreement: Tenses of verbs are required to agree with time adverbs. For example, ትላንት ሰበሩ (*tĭlant säbäru*/ [they] broke yesterday) is a valid verb phrase construction whereas ትላንት ይሰብራሉ (*tĭlant yĭsäbralu*/ [they] will break yesterday) is an invalid construction.

3 The Proposed Grammar Checker

The proposed grammar checker for Amharic text passes through three phases:

- Checking word sequences;
- Checking adjective-noun-verb agreements;
- Checking adverb-verb agreement.

In the first two phases, we employ the n-gram based statistical method. The n-gram probabilities are computed from the linguistic properties of words in a sentence.

3.1 Representation of the Morphological Properties of Words

To check grammatical errors in an Amharic sentence, the morphological properties of words is required. The morphological property of Amharic words contains linguistic information such as number, gender, person, etc. Such linguistic information is used to check whether the linguistic properties of one word agree with that of the other words in the sentence. For this task, we used an Amharic morphological analyzer known as HornMorpho developed by Gasser (2011). After performing morphological analysis for a given word, the morphological property of the word is stored along with its POS tag using a structure with four slots as shown in Figure 2.

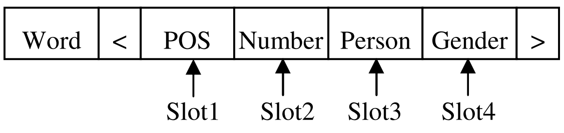


Figure 2: A structure for representing the linguistic properties of words

Slot1: This slot contains information about the POS tag of the word. The corpus we used in this work contains 31 types of POS tags, and the value for this slot is retrieved from the corpus. In addition to checking the correct POS tag sequence in a sentence, this slot is required to check agreements in number, person and gender as well.

Slot2: This slot holds number information about the word, i.e. whether the word is plural (P), singular (S), or unknown (U). In the case of nouns and adjectives, it has three values: P, S, or U. Since Amharic verbs are marked for numbers of subject and object, the value for this slot are combinations of the aforementioned values for subject and objective cases. We use the symbol “^” to represent such combinations. For example, a verb marked for plural subject and singular object is represented as SP^OS; a verb marked for singular subject and singular object is represented as SS^OS; etc.

Slot3: This slot stores person information about the word, i.e. first person (P1), second person (P2), third person (P3), or unknown (PU). The slot has four different possible values for the nouns and adjectives: P1, P2, P3 and PU. However, verbs can have a combination of these four values for subject and object grammatical functions. Examples of slot values for verbs are the following.

SP1^OP1:	verb marked for first person subject and first person object
SP2^OP1:	verb marked for second person subject and first person object
SP3^OP1:	verb marked for third person subject and first person object
SP2^OP3:	verb marked for second person subject and third person object
⋮	
etc.	

Slot4: This slot holds gender information about the word, i.e. whether the word is masculine (M), feminine (F), or unknown (U). In the case of nouns and adjectives, it has three values: M, F, or U. The values of this slot for verbs are are combinations of the aforementioned values for subject and objective cases. Accordingly, a verb marked for masculine subject and feminine object is represented as SM^OF; a verb marked for feminine subject and masculine object is represented as SF^OM; etc.

For example, the linguistic information built for the noun ፕሬዚዳንቱ (*prezidantu*/the president {masculine}) is: ፕሬዚዳንቱ <N|S|P3|M>. Likewise, the linguistic information for the verb ደረሰችበት (*däräsäčïbät*/she reached at him) is: ደረሰችበት <V|SS^OS|SP3^OP3|SF^OM>. Accordingly, the linguistic information about each word in the entire corpus is automatically constructed so as to use it for training and testing.

3.2 Word Sequences

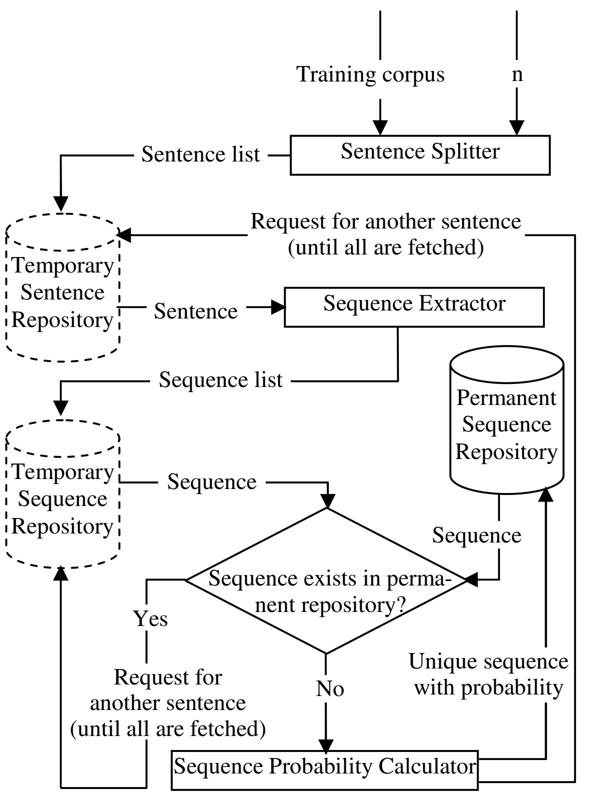
To check the validity of POS tag sequence for a given sentence, we use n-gram probability p_t computed as:

$$p_t(w_n \mid w_1 w_2 ... w_{n-1}) = \frac{count(w_1 w_2 ... w_{n-1} w_n)}{count(w_1 w_2 ... w_{n-1})} \qquad (1)$$

where n is the number of words in a sequence and w is POS tags of words. We have calculated n-gram values for n=2 (bigram) and n=3 (trigram) where they are saved in repository and used in grammar checking process. The probabilities of sequence occurrences are determined from the corpus, which is used to train the system. The training process starts by accepting the training corpus and the n-value as inputs. For each sentence in the corpus, the sequences of POS tags of words are extracted. For each unique sequence of POS tags, the probability of the occurrence of the sequence is computed using n-gram models. The n-gram probabilities of POS tag sequences stored in the permanent repository are accessed to check grammatical errors in a given sentence. The probability p_{st} of the correctness of the POS tag sequence of words in a given sentence construction is computed as:

$$p_{st} = \prod_{i=1}^{n} p_{t_i} \qquad (2)$$

where n is the number of POS tags extracted in the sentence. Sentence with higher values of p_{st} are considered to be having a valid sequence of words whereas those with low values are regarded as having unlikely sequence of words. Finally, the decision is made based on a threshold value set by empirical method. The training process is illustrated in Figure 3.



Training corpus
n
Sentence list
Sentence Splitter
Request for another sentence (until all are fetched)
Temporary Sentence Repository
Sentence
Sequence Extractor
Sequence list
Permanent Sequence Repository
Temporary Sequence Repository
Sequence
Sequence
Sequence exists in permanent repository?
Yes
No
Request for another sentence (until all are fetched)
Unique sequence with probability
Sequence Probability Calculator

Figure 3: A flowchart of the training process for checking sequences of words.

3.3 Adjective-Noun-Verb Agreements

The agreements between words serving various grammatical functions in Amharic sentence are also checked using n-gram approach. Number, person and gender agreements are checked at this phase. We perform this task by analyzing the four slots representing linguistic information about words as discussed in Section 3.1. Since the values for number, person, and gender depends on the word class, the POS tag information is required. Thus, for each word in the corpus, we extract such information as <slot1,slot2>, <slot1|slot3> and <slot1|slot4> where slot1, slot2, slot3 and slot4 represent POS tag, number, person and gender information, respectively. Given the POS tag w of a word, the sequence probability p_a of adjective-noun-verb agreement for a slot is computed as:

$$p_a(w_n v_n \mid w_1 v_1 \ldots w_{n-1} v_{n-1}) = \frac{count(w_1 v_1 \ldots w_n v_n)}{count(w_1 v_1 \ldots w_{n-1} v_{n-1})} \quad (3)$$

where v is the value of the slot. The n-gram probability values for each unique pattern was computed and stored in a permanent repository which would be later accessed to adjective-noun-verb agreements in a given sentence. The probability p_{sa} of the correctness of the adjective-noun-verb agreements in a given sentence is then computed as:

$$p_{sa} = \prod_{i=1}^{n} p_{a_i} \quad (4)$$

3.4 Adverb-Verb Agreement

Amharic adverbs usually come before the verb they modify. When adverb appears in the sentence it usually modifies the next verb that comes after it. There could be a number of other words in between the adverb and the verb, but the modified verb appears next to the modifier before any other verb in the sentence. As Amharic adverbs are few in number, adverb-verb agreement was not checked in the previous phases. To check time agreement between the adverb and the verb, the tense for the verb that the adverb modifies should be identified. In this work, we considered four different types of tenses: perfective, imperfective, jussive/ imperative and gerundive. The pattern of time adverbs associated with each tense type was extracted from the corpus and stored in repository. Whenever these time adverbs are found in the sentence to be checked, the tense type of the next verb is extracted by using morphological analysis. If the tense type extracted from the given sentence matches with an adverb-tense pattern in the repository, the adverb and the verb are considered to have correct agreement. Otherwise, it is reported as grammatically incorrect sentence.

4 Experiment

4.1 The Corpus

We used Walta Information Center (WIC) news corpus which contains 8067 sentences where words are annotated with POS tags. We used 7964 sentences for training and the remaining for testing. In addition, to test the performance of the system with grammatically wrong sentences, we also used manually prepared sentences which are grammatically incorrect.

4.2 Test Results

In order to test the performance of the grammar checker, we are required to compute the number of actual errors in the test set, number of errors reported by the system and the number of false positives generated by the system. These numbers were then used to calculate the precision and recall of the system as follows.

$$precision = \frac{number\ of\ correctly\ flagged\ errors}{total\ number\ of\ flagged\ errors} * 100\% \quad (5)$$

$$recall = \frac{number\ of\ correctly flagged\ errors}{total\ number\ of\ grammatical\ errors} * 100\% \quad (6)$$

Accordingly, we tested the system with simple and complex sentences where we obtained experimental results as shown in Table1.

Type of Sentence	n-gram model	Precision	Recall
Simple	Bigram	59.72%	82.69%
	Trigram	67.14%	90.38%
Complex	Bigram	57.82%	65.38%
	Trigram	63.76%	67.69%

Table 1: Experimental results.

Experimental results were also analyzed to evaluate the performance of the system with regard to identifying various types of grammatical errors. The detection rate of the various grammatical error types is shown in Table 2.

Error type	Detection rate (%)
Incorrect word order	73
Number disagreement	80
Person disagreement	52
Gender disagreement	60
Adjective-noun disagreement	55
Adverb-verb disagreement	90

Table 2: Detection rate by error types.

4.3 Discussion

A complete system that checks Amharic grammatical errors is developed. To train and test our system, we used WIC corpus which is manually annotated with POS tags. However, we have observed that a number of words are tagged with wrong POS and many of them are also misspelled. Since Amharic is one of the less-resourced languages, to our best knowledge, there is no tool that checks and corrects the spelling of Amharic words. Although attempts have been made to correct some of the erroneously tagged words in the corpus, were were unable to manually correct all wrongly tagged words. POS tag errors cause the wrong tag patterns to be interpreted as correct ones during the training process which would ultimately affect the performance of the system. Thus, the performance of the system can be maximized if the system is trained with error-free corpus. Moreover, since the corpus is collected from news items, most of the sentences contain words which refer to third person. For this reason, occurrence of first and second person in the corpus is very small. This has affected the system while checking person disagreement. This is evidenced by the low accuracy obtained while the system detects number disagreement (see Table 2).

To our best knowledge, HornMorpho is the only tool at present publicly available to morphologically analyze Amharic words. However, the tool analyses only some specific types of verbs and nouns. Adjectives analyzed as nouns and adverbs are not analyzed at all. Since Amharic is morphologically very complex language where combinations of various linguistic information are encoded in a single word, the effectiveness of grammar checking is hugely compromised if words are not properly analyzed. Thus, the performance of the system can be greatly enhanced by using a more effective Amharic morphological analyzer.

Test results have shown that trigram models perform better than bigram models. In Amharic, head words in verb phrases, noun phrases, adjective phrases are located at the end of the phrases (Yimam, 2000). This means that, for verb phrases, the nouns and adjectives for which verbs are marked come immediately before the head word (which is a verb). Likewise, sequences of adjectives modifying nouns in noun phrases come immediately before the head word (which is a noun). Thus, sequences of multiple words in phrases are better captured by trigrams than bigrams. We have also seen that grammatical errors in simple sentences are detected more accurately than in complex sentences. The reason is that complex sentences have complex phrasal structures which could not be directly treated by trigram and bigram models. However, the performance of the system can be improved by using a parser that generates phrasal structures hierarchically at different levels. We can then systematically check grammatical errors at various levels in line with the parse result.

5 Conclusion and Future Works

Amharic is one of the most morphologically complex languages. Furthermore, it is considered to be less-resourced language. Despite its importance, these circumstances lead to unavailability of efficient NLP tools that automatically process Amharic texts at present. This work is aimed at contributing to the ever-increasing need of developing Amharic NLP tools. Accordingly, the development of Amharic grammar checker using morphological features and n-gram probabilities is presented. In this work, we have systematically treated the morphological features of the language where we represented grammar dependency rules extracted from the morphological structures of words.

However, lack of error-free corpus and effective morphological analyzer are observed to be affecting the performance of the developed grammar checker. Thus, future works are recommended to be directed at improving linguistic resources and developing effective NLP tools such morphological analyzer, parser, spell checker, etc. for Amharic. The efficiency of these components is crucial not only for Amharic grammar checking but also for many Amharic NLP applications.

References

Amare, G. (2010). ዘመናዊ የአማርኛ ሰዋስው በቀላል አቀራረብ (Modern Amharic Grammar in a Simple Approach). Addis Ababa, Ethiopia.

Arppe, A. (2000). “Developing a Grammar Checker for Swedish”; In *Proceeding of the 12th Nordic conference on computational linguistic*. pp 9 – 27.

Domeij, R., Knutsson, O., Carlberger, J. and Kann, V. (2000). “Granska: An efficient hybrid system for Swedish grammar checking”; In *Proceedings of the 12th Nordic conference on computational linguistics*, Nodalida- 99.

Ehsan, N. and Faili, H. (2013), “Grammatical and context-sensitive error correction using a statistical machine translation framework”; *Softw: Pract. Exper.*, 43: 187–206. doi: 10.1002/spe.2110.

Gasser, M. (2011). “HornMorpho: a system for morphological analysis and generation of Amharic, Oromo, and Tigrinya words”' In *Proceedings of the Conference on Human Language Technology for Development*.

Lewis, M. Paul, Gary F. Simons, and Charles D. Fennig (2013); *Ethnologue: Languages of the World, Seventeenth edition.* Dallas, Texas: SIL International.

Liddy, E. D. (2001) “Natural language processing”, In *Encyclopedia of Library and Information Science*, 2nd Ed. Marcel Decker, Inc.

Mozgovoy, M. (2011). “Dependency-based rules for grammar checking with LanguageTool”, *Federated Conference on Computer Science and Information Systems (FedCSIS)* Sept. 18-21, 2011, pp. 209-212, Szczecin, Poland.

Mudge, R. (2010). “The design of a proofreading software service”; In *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids*. pp 24-32, Stroudsburg, PA, USA.

Naber, D. (2003). “A Rule-Based Style and Grammar Checker”; PhD Thesis, Bielefeld University, Germany.

Richardson, S. (1997). “Microsoft Natuaral language Understanding System and Grammar checker”; *Microsoft*, USA,.

Schmidt-Wigger, A. (1998). “Grammar And Style Checking in German”; In *Proceedings of CLAW*. Vol 98.

Shaalan, K. (2005). “Arabic Gramcheck: A Grammar Checker for Arabic”; The British University in Dubai, United Arab Emirates.

Tsuruga, I. and Aizu W. (2011). “Dependency-Based Rules for Grammar Checking with LanguageTool”. Maxim Mozgovoy. University of Aizu. IEEE. Japan, 2011.

Xing, J., Wang, L., Wong, D. F., Chao, S., and Zeng, X. (2013). “UM-Checker: A Hybrid System for English Grammatical Error Correction”; In *Proceedings of CoNLL-2013*, vol. 34.

Yimam, B. (2000). የአማርኛ ሰዋስው (Amharic Grammar). Addis Ababa, Ethiopia.

LCFRS binarization and debinarization for directional parsing

Wolfgang Maier
Universität Düsseldorf
Institut für Sprache und Information
Universitätsstr. 1, 40225 Düsseldorf, Germany
maierw@hhu.de

Abstract

In data-driven parsing with Linear Context-Free Rewriting System (LCFRS), markovized grammars are obtained through the annotation of binarization non-terminals during grammar binarization, as in the corresponding work on PCFG parsing. Since there is indication that directional parsing with a non-binary LCFRS can be faster than parsing with a binary LCFRS, we present a debinarization procedure with which we can obtain a non-binary LCFRS from a previously binarized one. The resulting grammar retains the markovization information. The algorithm has been implemented and successfully applied to the German NeGra treebank.

1 Introduction

Linear Context-Free Rewriting System (LCFRS), an extension of CFG in which non-terminals can cover more than a single span, can be used to model discontinuous constituents as well as non-projective dependencies (Maier and Lichte, 2011). It has therefore recently been exploited for direct data-driven parsing of such structures. See, e.g., Maier and Søgaard (2008), Maier and Kallmeyer (2010), van Cranenburgh (2011), van Cranenburgh et al. (2012), and Kallmeyer and Maier (2013).

A major problem with data-driven probabilistic LCFRS (PLCFRS) parsing is the high parsing complexity. Given a binary grammar, CYK parsing can be done in $\mathcal{O}(n^{3k})$ where k is the *fan-out* of the grammar, that is, the maximum number of spans that a single non-terminal can cover (Seki et al., 1991). While for PCFG parsing, k is 1, for PLCFRS, k will typically be ≈ 5. Sentences with lengths around 23 to 25 words require very high, unpractical parsing times (20 minutes and more per sentence) (van Cranenburgh et al., 2011; Kallmeyer and Maier, 2013).

One possibility to obtain faster parsing is to reduce the fan-out of the grammar by reducing the number of gaps in the trees from which the grammar is extracted. This has been done by Maier et al. (2012) who transform the trees of the German TIGER treebank such that a grammar fan-out of 2 is guaranteed. For unrestricted PLCFRS parsing, other solutions have been implemented. The parser of Kallmeyer and Maier (2013)[1] offers A^* parsing with outside estimates (Klein and Manning, 2003a). With this technique the practical sentence length limit is shifted upwards by 7 to 10 words. Kallmeyer and Maier also propose non-admissible estimates which provide a greater speed-up but also let the results degrade. The parser of van Cranenburgh (2012)[2] also does not maintain exact search. It implements a coarse-to-fine strategy in a more general PCFG is created from the treebank PLCFRS using the algorithm of Barthélemy et al. (2001). The PCFG chart is then used to filter the PLCFRS chart. While this approach in principle removes the limit[3] on sentence length, it also leads to degraded results.

Yet another solution for obtaining higher speeds could be to turn to parsing strategies which allow for the use of non-binary rules, such as directional CYK parsing or Earley parsing. The reasoning behind this assumption is as follows. Firstly, the longer the right-hand side of a rule, the easier it is to check during parsing on a symbolic basis if it is possible to use the rule

[1]See http://phil.hhu.de/rparse.
[2]See http://github.com/andreasvc/disco-dop.
[3]For very long sentences, i.e. > 60 words, the parser still has extreme memory requirements.

in a complete parse or not by checking the requirements of the rule with respect to the yet unparsed part of the input such as it is done, e.g., in Kallmeyer and Maier (2009).[4] A comparable strategy, the F grammar projection estimate (Klein and Manning, 2003a), has been employed in PCFG parsing. Secondly, practical experience with Range Concatenation Grammar (RCG) (Boullier, 1998) and Grammatical Framework (GF) (Ranta, 2004) parsing points in the same direction. Lazy computation of instantiations in RCG parsing as done in the TuLiPA system (Kallmeyer et al., 2009) and the SYNTAXE parser (Boullier and Deschamp, 1988) (Boullier, p.c.) seem to be less effective with shorter right-hand sides of rules because less constraints can be collected at once. Practical experiments with the GF[5] parser (Angelov, 2009), which implements an Earley strategy, indicate that certain optimizations loose their effect with binary grammars (Angelov, p.c.).

So why not just do directional parsing with the unbinarized treebank grammar? It is common knowledge that vanilla treebank PCFGs do not perform well. This also holds for PLCFRS. Markovization has been proven to be an effective remedy for both PCFG and PLCFRS parsing. It can be achieved through the probability model itself (Collins, 1999) or by annotating the treebank grammar (Klein and Manning, 2003b; Kallmeyer and Maier, 2013): Instead of using a unique non-terminal as in deterministic binarization, one uses a single non-terminal and adorns it with the vertical ("parent annotation", see Johnson (1998)) and horizontal context of the occurrence of the rule in the treebank. This augments the coverage of the grammar and helps to achieve better parsing results by adding a possibly infinite number of implicit non-binary rules.

Our main contribution in this article is a *debinarization* algorithm with which a non-binary LCFRS can be generated from a previously binarized LCFRS (which fulfills certain conditions). Given a markovized binarized grammar, the debinarized grammar contains non-binary productions obtained through markovization. We furthermore contribute a new compact notation for rules of a *treebank LCFRS*, i.e., of the variant of LCFRS obtained by treebank grammar extraction, and provide a formulation of deterministic binarization using this notation.

An implementation of the debinarization algorithm has been tested on the German NeGra treebank (Skut et al., 1997). First experimental results confirm that in practice, the debinarized grammar can perform better than than the plain treebank grammar.

The remainder of the article is structured as follows. In the following section, we define treebank LCFRS and introduce our new rule representation. We furthermore introduce the binarization, resp. markovization algorithm. In section 3, we introduce our new debinarization algorithm. Section 4 presents the experimental evaluation and section 5 closes the article.

2 LCFRS

2.1 Definition

In LCFRS (Vijay-Shanker et al., 1987), a single non-terminal can span $k \geq 1$ continuous blocks of a string. A CFG is simply a special case of an LCFRS in which $k = 1$. k is called the *fan-out* of the non-terminal. We notate LCFRS with a syntax of Simple Range Concatenation Grammars (SRCG) (Boullier, 1998), a formalism equivalent to LCFRS. In the following we define *treebank LCFRS*, the variant of LCFRS which is obtained by the grammar extraction algorithm of Maier and Søgaard (2008).

A *treebank LCFRS* is a tuple $G = (N, T, V, P, S)$ where

1. N is a finite set of non-terminals with a function $dim\colon N \to \mathbb{N}$ determining the *fan-out* of each $A \in N$;
2. T and V are disjoint finite sets of terminals and variables;
3. $S \in N$ is the start symbol with $dim(S) = 1$;
4. P is a finite set of rewriting rules where all $r \in P$ are either
   (a) rules with rank $m \geq 1$ of the form

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)}) \\ \cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

   where
   i. $A, A_1, \ldots, A_m \in N$, $X_j^{(i)} \in V$ for $1 \leq i \leq m, 1 \leq j \leq dim(A_i)$ and

[4]For such a check, it makes no difference if the rule is deterministically binarized.

[5]`http://grammaticalframework.org`

$A(a) \to \varepsilon$	($\langle a \rangle$ in yield of A)
$B(b) \to \varepsilon$	($\langle b \rangle$ in yield of B)
$C(X,Y) \to A(X)A(Y)$	(if $\langle X \rangle$ in the yield of A and $\langle Y \rangle$ in the yield of A, then $\langle X, Y \rangle$ in yield of C)
$S(XZY) \to C(X,Y)B(Z)$	(if $\langle X, Y \rangle$ in yield of C and $\langle Z \rangle$ in the yield of B, then $\langle XZY \rangle$ in yield of S)

$L = \{a^n b a^n \mid n > 0\}$

Figure 1: Yield example

ii. $\alpha_i \in V^+$ for $1 \leq i \leq dim(A)$ (we write $\alpha_i.j$, $1 \leq j \leq |\alpha_i|$ for the jth variable in α_i), or

(b) rules of rank 0 of the form $A(t) \to \varepsilon$ where $A \in N, t \in T$.

For all $r \in P$, every variable X that occurs in r occurs exactly once in the left-hand side (LHS) and exactly once in the right-hand side (RHS). Furthermore, if for two variables $X_1, X_2 \in V$, it holds that $X_1 \prec X_2$ on the RHS, then also $X_1 \prec X_2$ on the LHS.[6]

A rewriting rule describes how to compute the yield of the LHS non-terminal from the yields of the RHS non-terminals. The yield of S is the language of the grammar. See figure 1 for an example.

The *rank* of G is the maximal rank of any of its rules, its *fan-out* is the maximal fan-out of any of its non-terminals.

The properties which distinguish a treebank LCFRS from a regular LCFRS are the requirements that

1. all non-terminal arguments are variables except in terminating lexical rules in which the only argument of the LHS consists of a single terminal, and

2. the ordering property.

These properties allows us to notate rules in a more compact way. Let (N, T, V, P, S) be an LCFRS. A rule $r \in P$

$$A(\alpha_1, \ldots, \alpha_{dim(A)}) \to A_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)}) \cdots A_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

[6]This is the *ordering* property which Villemonte de la Clergerie (2002) defines for RCGs. LCFRSs with this property are called *monotone* (Michaelis, 2001), MCFGs *non-permuting* (Kracht, 2003).

can be represented as a tuple $(A \to A_1 \cdots A_m, \vec{\varphi})$, where $\vec{\varphi}$ is a *linearization vector* which is defined as follows. For all $1 \leq i \leq dim(A)$, $1 \leq j \leq |\alpha_i|$, $\vec{\varphi}[i]$ is a *subvector* of $\vec{\varphi}$ with $|\vec{\varphi}[i]| = |\alpha_i|$ and it holds that $\vec{\varphi}[i][j] = x$ iff $\alpha_i.j$ occurs in the arguments of A_x. Let us consider an example: The rule $A(X_1X_2, X_3X_4) \to B(X_1, X_3)C(X_2, X_4)$ can be fully specified by $(A \to BC, [[1,2],[1,2]])$.

We introduce additional notation for operations on a linearization vector $\vec{\varphi}$.

1. We write $\vec{\varphi}/(x, x')$ for some $x, x' \in \mathbb{N}$ for the substitution of all occurrences of x in $\vec{\varphi}$ by x'. $\vec{\varphi}/^s(x, x')$ denotes the same substitution procedure with the addition that after the substitution, all occurrences of x'^+ in $\vec{\varphi}$ are replaced by x'.

2. We write $\vec{\varphi} \setminus x$ for some $x \in \mathbb{N}$ for the deletion of all occurrences of x in $\vec{\varphi}$, followed by the deletion of subvectors which become empty.

3. $\vec{\varphi} \downarrow^x$ for some $x \in \mathbb{N}$ denotes the splitting of all subvectors of $\vec{\varphi}$ such that a vector boundary is introduced between all $\vec{\varphi}[i][j]$ and $\vec{\varphi}[i][j+1]$, $1 \leq i \leq |\vec{\varphi}|$, $1 \leq j \leq |\vec{\varphi}[i]| - 1$ iff $\vec{\varphi}[i][j] = x$.

Note that for every LCFRS, there is an equivalent LCFRS which fulfills the treebank LCFRS conditions (Kallmeyer, 2010).

A *probabilistic (treebank) LCFRS* (PLCFRS) is a tuple $\langle N, T, V, P, S, p \rangle$ such that $\langle N, T, V, P, S \rangle$ is a (treebank) LCFRS and $p : P \to [0..1]$ a function such that for all $A \in N$: $\Sigma_{A(\vec{x}) \to \vec{\Phi} \in P} p(A(\vec{x}) \to \vec{\Phi}) = 1$. The necessary counts for a Maximum Likelihood estimation can easily be obtained from the treebank (Kallmeyer and Maier, 2013).

2.2 Binarization and Markovization

Using the linearization vector notation, deterministic left-to-right binarization of treebank LCFRS can be accomplished as in algorithm 1. The algorithm creates binary top and bottom rules but can easily be extended to create unary rules in these places. Note that the algorithm behaves identically to other deterministic binarization algorithms such as, e.g., the one in Kallmeyer (2010).

The algorithm works as follows. We start with the linearization vector $\vec{\varphi}$ of the original rule. In each binarization step, we "check off" from $\vec{\varphi}$ the information which was used in the previous step. The

Algorithm 1 LCFRS binarization

```
Let $(N, T, V, P, S)$ be a treebank LCFRS
$P' = \emptyset$
**for all** $(A \to A_1 \dots A_m, \vec{\varphi}_r)$ in $P$ with $m > 2$ **do**
    pick new non-terminals $C_1, \dots, C_{m-2}$
    let $\vec{\varphi} = \vec{\varphi}_r$
    let $\vec{\varphi}_g = \vec{\varphi}_r /^s (x, 2)$ for all $x > 1$
    add the rule $A \to A_1 C_1, \vec{\varphi}_g$ to $P'$
    **for all** $i$, $1 \leq i < m - 2$ **do**
        let $\vec{\varphi} = \vec{\varphi}/(x, x-1)$ for all $x$
        let $\vec{\varphi} = (\vec{\varphi} \downarrow^0) \setminus 0$
        let $\vec{\varphi}_g = \vec{\varphi}/^s(x, 2)$ for all $x > 1$
        add the rule $(C_i \to A_{i+1} C_{i+1}, \vec{\varphi}_g)$ to $P'$
    let $\vec{\varphi} = \vec{\varphi}/(x, x-1)$ for all $x$
    let $\vec{\varphi} = (\vec{\varphi} \downarrow^0) \setminus 0$
    add the rule $(C_{m-2} \to A_{m-1} A_m, \vec{\varphi})$ to $P'$
set $P$ to $P'$
```

linearization vectors of the binary rules are obtained from the respective current state of $\vec{\varphi}$ by attributing all information which will covered by the new binarization non-terminal to exactly this non-terminal.

As an example, consider the binarization of the rule $A(X_1X_2, X_3X_4, X_5X_6) \to B(X_1, X_3)C(X_2, X_4)D(X_5)E(X_6)$. The compact representation of the rule is $(A \to BCDE, [[1,2][1,2][3,4]])$. The first step yields the rule $(A \to BC_1, [[1,2],[1,2],[2]])$. $\vec{\varphi}$ is set to the linearization vector of the original rule. Subsequently, we remove from $\vec{\varphi}$ the information which has already been used by subtracting one from all integers in $\vec{\varphi}$, splitting the vector at occurrences of 0 (i.e., new argument boundaries are introduced at positions which have been covered by the previous binarization step), and then removing all occurrences of 0. From the resulting vector $[[1],[1],[2,3]]$, we create the linearization vector for the second binary rule $(C_1 \to CC_2, [[1],[1],[2]])$ by attributing all material covered by C_2 to C_2. In other words, the last subvector can be reduced from $[2,3]$ to $[2]$ since only in the next and last binarization step we will distinguish between 2 and 3. In the subsequent last step, we again check off the already used material from $\vec{\varphi}$ and end up with $(C_2 \to DE, [[1,2]])$.

During binarization, a grammar can be markovized by changing the choice of binarization non-terminals. Instead of unique non-terminals $C_1, \dots, C_{m-2}$, we pick a single non-terminal $@$. At each binarization step, to this non-terminal, we append the vertical occurrence context from the treebank, i.e., we perform parent annotation (Johnson, 1998), and we append the horizontal context. More precisely, the markovization information for the binarization non-terminal that comprises original RHS elements $A_i \dots A_m$ are the first v elements of path from A_i to root vertically and the first h elements of $A_i \dots A_0$ horizontally.

3 Debinarization

The goal of the debinarization procedure is to obtain a grammar with non-binary rules from a previously binarized one. Algorithm 2 accomplishes the debinarization. It consists of a function *debin*, which, assuming a treebank LCFRS (N, T, V, P, S) binarized with algorithm 1, is called on all $A \in N$. During debinarization, linearization vectors must be recombined. This is notated as function *lin*.

Called on some non-terminal $A \in N$, the algorithm recursively substitutes all binarization non-terminals on the right-hand sides of all binary A rules with the right-hand sides of rules which have the same binarization non-terminal on its LHS. The base case of this recursion are rules which do not have binarization non-terminals on their right-hand side. At each substitution, we recombine the corresponding linearization vectors: When substituting the binarization non-terminal C on the right-hand side of a binary production by the RHS of a debinarized rule r with C on its LHS, roughly, we replace the ith occurrence of 2 in the linearization vector by the ith subvector of the linearization vector of r, adding 1 to all of its elements.[7]

The probability of a rule in which a binarization non-terminal on the RHS is substituted by the RHS of a rule with this binarization non-terminal on its LHS is simply the product of both probabilities.

As an example, consider the debinarization of the non-terminal A given the binarized rules from our previous example. We have a single A rule, in which we would recursively substitute the C_1 on the right-hand side with $C\,C_2$ and in the following as base case C_2 with $D\,E$. In the base case, $\mathbf{lin}$ is not called, be-

[7]The algorithm would be slightly more complex if, during binarization, we would permit that binarization non-terminals end up on the left corner of the right-hand side of a binary production. Algorithm 1 guarantees that binarization non-terminals occur only on the right corner.

cause no linearization vector recombination is necessary. For the substitution of C_2, we combine the linearization vectors of the C_1 and C_2 rules. For this, we first add 1 to all elements of the C_2 vector, which gets us $[[2,3]]$. We then replace the only occurrence of 2 in the C_1 vector by $[2,3]$. The result is the vector $[[1],[1],[2,3]]$. For the substitution of C_1 in the A rule, we recombine our result vector with the vector of the A rule. For this, we first add 1 to all elements of the result of the vector recombination, which gets us $[[2],[2],[3,4]]$. We then replace all three occurrences of 2 in the vector $[[1,2],[1,2],[2]]$ of the binary A rule with the corresponding subvectors of $[[2],[2],[3,4]]$. This give us $[[1,2],[1,2],[3,4]]$, which is the linearization vector of the original unbinarized A rule.

In the case of deterministic binarization, the algorithm yields a grammar which is equivalent to the grammar before the binarization. With markovization, i.e., with non-deterministic binarization, it is more difficult. Since we do not choose our binarization non-terminals in a unique way, it is possible that a chain of substitutions can lead us in a cycle, i.e., that we reach the same binarization non-terminal again. A cycle corresponds to an infinite number of implicit non-binary rules. The smaller the binarization context, the more likely it is that such a cycle occurs.

Several strategies are possible to avoid an infinite loop of substitutions. The most obvious one is to use a cache which contains all binarization non-terminals seen during debinarization. If a substitution introduces a binarization non-terminal which is in the cache, the corresponding production is discarded. Another strategy is to discard rules with a probability lower than a certain threshold.

Algorithm 2 LCFRS debinarization

```
function $\mathbf{debin}(A \in N)$
let $\mathcal{R} = \emptyset$
for all $(A \to A_1 A_2, \vec{\varphi})$ in $P$ do
  if $A_1, A_2$ are not bin. non-terminals then
    add $(A \to A_1 A_2, \vec{\varphi})$ to $\mathcal{R}$
  else
    let $\mathcal{D} = \mathbf{debin}(A_2)$
    for all $(D \to D_0 \cdots D_m, \vec{\varphi}_D) \in \mathcal{D}$ do
      add $(A \to A_1 D_0 \cdots D_m, \mathbf{lin}(\vec{\varphi}, \vec{\varphi}_D, 2))$ to $\mathcal{R}$
return $\mathcal{R}$

function $\mathbf{lin}(\vec{\varphi}, \vec{\varphi}_D, s)$
let $c = 0$
let $\vec{\varphi}_D = \vec{\varphi}_D/(x, x+1)$ for all $x$
for all occurrences of $s$ in $\vec{\varphi}$ do
  replace occ. of $s$ with the content of $\vec{\varphi}_D[c]$
  $c = c + 1$
return $\vec{\varphi}$
```

4 Experiments

We have implemented the grammar extraction algorithm from Maier and Søgaard (2008), as well as the binarization and the debinarization algorithm. We implement the debinarization algorithm with a probability filter: A substitution of a binarization non-terminal will not take place if the probability of the resulting production falls under a certain threshold. We furthermore implement a filter which excludes rules with right-hand sides that exceed a certain length.

In order to experimentally test the debinarization algorithm, we perform experiments on the NeGra treebank (Skut et al., 1997) using rparse.[8] In a first step, we apply the algorithm for re-attaching elements to the virtual root node described in (Maier et al., 2012). All sentences longer than 20 words are excluded. For parsing, we split the data and use the first 90% of all sentences for training and the remainder for parsing. We then extract the grammar from the training part (results in 10,482 rules) and binarize them, using deterministic binarization and using markovization with vertical and horizontal histories of 1, resp. 2. The markovized grammar is debinarized, with an experimentally determined logarithmic rule weight of 15 for the filtering and a limit of 15 on the lengths of rule right-hand sides. This results in a grammar with 175,751 rules.

We then parse with the deterministically binarized treebank grammar, with the markovized binary grammar, and with debinarized grammar. Note that rparse currently offers no directional parser. Therefore we rebinarize the debinarized grammar using deterministic binarization. Using bracket scoring (evalb)[9], we obtain the 75.49 F_1 for the plain treebank grammar,

[8]See `http://phil.hhu.de/rparse`.

[9]See `http://github.com/wmaier/evalb-lcfrs`.

77.10 for the markovized grammar and 76.37 for the debinarized grammar. This shows that the debinarized markovized grammar can perform better than the plain treebank grammar. However, the large number of productions in the debinarized grammar indicates that the possibilities of filtering must be investigated more closely. This is left for future work.

5 Conclusion

We have presented a new compact representation for grammar rules of a treebank LCFRS together with a formulation of a binarization algorithm. Furthermore, we have presented a procedure for debinarizing a previously binarized LCFRS. The resulting grammar maintains the markovization information introduced during binarization and can be used with directional parsing strategies. Experiments have shown that the grammar can perform better than the plain treebank grammar. There remains potential for optimization.

We are currently working on the integration of the algorithm into a directional parsing strategy within a data-driven PLCFRS parser.

Acknowledgments

I would like Laura Kallmeyer, Miriam Kaeshammer and the three reviewers for comments and suggestions.

References

Krasimir Angelov. 2009. Incremental parsing with Parallel Multiple Context-Free Grammars. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 69–76, Athens, Greece.

François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric Villemonte de la Clergerie. 2001. Guided parsing of Range Concatenation Languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 42–49, Toulouse, France.

Pierre Boullier and Philippe Deschamp, 1988. *Le système SYNTAX^{TM} – manuel d'utilisation et de mise en oeuvre sous UNIX^{TM}*. `http://syntax.gforge.inria.fr/syntax3.8-manual.pdf`, January 4, 2012.

Pierre Boullier. 1998. A generalization of mildly context-sensitive formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 17–20, Philadelphia, PA.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Laura Kallmeyer and Wolfgang Maier. 2009. An incremental Earley parser for Simple Range Concatenation Grammar. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 61–64, Paris, France. Association for Computational Linguistics.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Laura Kallmeyer, Wolfgang Maier, and Yannick Parmentier. 2009. An Earley parsing algorithm for Range Concatenation Grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 9–12, Singapore.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer.

Dan Klein and Christopher D. Manning. 2003a. A* Parsing: Fast exact viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 40–47, Edmonton, Canada.

Dan Klein and Christopher D. Manning. 2003b. Accurate unlexicalized parsing. In *Proceedings of the 41th Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Marcus Kracht. 2003. *The Mathematics of Language*. Mouton de Gruyter, Berlin.

Wolfgang Maier and Laura Kallmeyer. 2010. Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Proceedings of the Tenth International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+10)*, pages 119–126, Yale University, New Haven, CT.

Wolfgang Maier and Timm Lichte. 2011. Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected*

Papers, volume 5591 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Berlin/Heidelberg/New York. Springer-Verlag.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In Philippe de Groote, editor, *Proceedings of the 13th Conference on Formal Grammar (FG-2008)*, pages 61–76, Hamburg, Germany. CSLI Publications.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. Data-driven PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, Paris, France. to appear.

Jens Michaelis. 2001. *On Formal Properties of Minimalist Grammars*. Ph.D. thesis, Potsdam University, Potsdam, Germany.

Aarne Ranta. 2004. Grammatical Framework, a typetheoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Hiroyuki Seki, Takahashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 88–95, Washington, DC.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. 2011. Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2011)*, pages 34–44, Dublin, Ireland.

Andreas van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon, France.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA. Association for Computational Linguistics.

Éric Villemonte de la Clergerie. 2002. Parsing Mildly Context-Sensitive Languages with Thread Automata. In *Proceedings of COLING 2002: The 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Towards Fully Lexicalized Dependency Parsing for Korean

Jungyeul Park
UMR 6074 IRISA
Université de Rennes 1
Lannion, France
jungyeul.park@univ-rennes1.fr

Daisuke Kawahara **Sadao Kurohashi**
Graduate School of Informatics
Kyoto University
Kyoto, Japan
{dk,kuro}@i.kyoto-u.ac.jp

Key-Sun Choi
Dept. of Computer Science
KAIST
Daejeon, Korea
kschoi@kaist.edu

Abstract

We propose a Korean dependency parsing system that can learn the relationships between Korean words from the Treebank corpus and a large raw corpus. We first refine the training dataset to better represent the relationship using a different POS tagging granularity type. We also introduce lexical information and propose an almost fully lexicalized probabilistic model with case frames automatically extracted from a very large raw corpus. We evaluate and compare systems with and without POS granularity refinement and case frames. The proposed lexicalized method outperforms not only the baseline systems but also a state-of-the-art supervised dependency parser.

1 Introduction

Korean dependency parsing has been studied more in comparison with constituent parsing owing to its relatively free word order in Korean (Chung, 2004; Lee and Lee, 2008; Oh and Cha, 2010). A dependency structure is less restricted by the word order because it does not require that one constituent is followed by another. Statistical parsing trained from an annotated dataset has been widespread. However, while there are manually annotated several Korean Treebank corpora such as the Sejong Treebank corpus (SJTree), only a few works on statistical Korean parsing have been conducted. For constituent parsing, (Sarkar and Han, 2002) used a very early version of the Korean Penn Treebank (KTB) to train lexicalized Tree Adjoining Grammars (TAG). (Chung et al., 2010) used context-free grammars and tree-substitution grammars trained on data from the KTB. Most recently, (Choi et al., 2012) proposed a method to transform the word-based SJTree into an entity-based Korean Treebank corpus to improve the parsing accuracy. For dependency parsing, (Chung, 2004) presented a model for dependency parsing using surface contextual information. (Oh and Cha, 2010) developed a parsing model with cascaded chunking by means of conditional random fields learning. (Choi and Palmer, 2011) used the Korean dependency Treebank converted automatically from the SJTree.

In this paper, we start with an unlexicalized Korean dependency parsing system as a baseline system that can learn the relationship between Korean words from the Treebank corpus. Then, we try to improve the parsing accuracy using internal and external resources. For internal resources, we can refine the training dataset for a better representation of the relationship by means of POS tagging granularity. For external resources, we introduce lexical information and propose a lexicalized probabilistic model with case frames. We automatically extract predicate-argument structures from a large raw corpus outside of the training dataset and collect them as case frames to improve parsing performance.

2 Dependency grammars

Converting phrase-structure grammars from the Treebank corpus into dependency grammars is not a trivial task (Wang, 2003; Gelbukh et al., 2005; Candito et al., 2010). We implement a word-to-word conversion algorithm for the Sejong Treebank corpus. Firstly, we assign an anchor for nonterminal nodes using bottom-up breadth-first search. An anchor is the terminal node where each nonterminal node can have as a lexical head node. We use lexical head rules described in (Park, 2006). It assigns only the lexical head for nonterminal nodes at the moment and finds dependencies

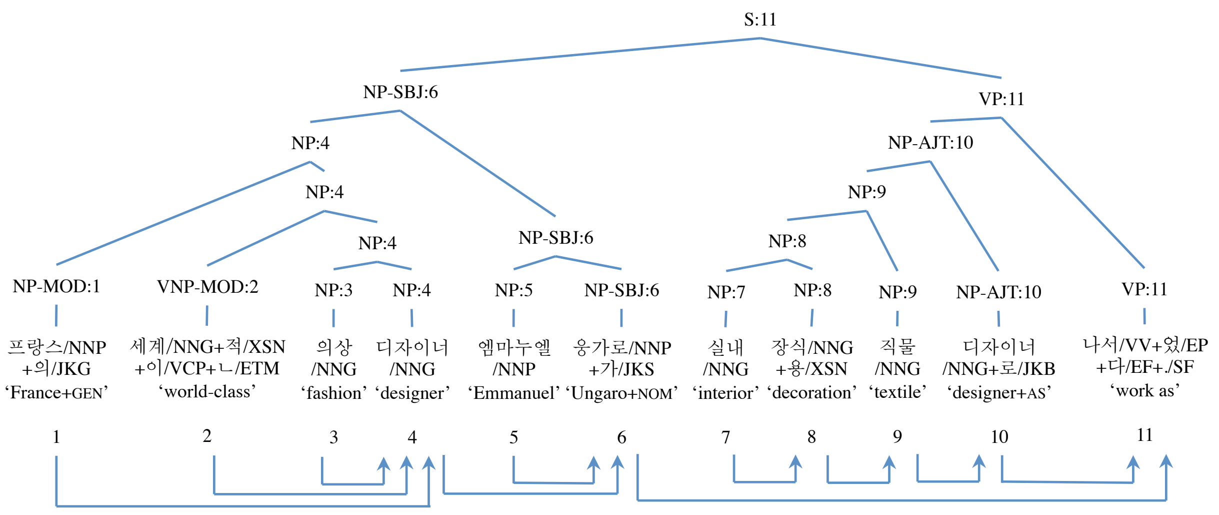


Figure 1: Example of the original SJTree (above) and its dependency representation (below) for the example sentence 'The world-class French fashion designer Emanuel Ungaro worked as an interior textile designer.': The address of terminal nodes (underneath) and the anchor of nonterminal node (on its right) are assigned using lexical head rules. The head of the terminal node 1 is the node 4, which is the anchor of the parent of the parent node (NP:4). The head of the terminal node 4 is the node 6 where the anchor of its ancestor node is changed from itself (NP-SBJ:6). The head of the terminal node 11 is itself where the anchor of the root node and itself are same (S:11).

would be in the next step. Lexical head rules give priorities to the rightmost child node, which inherits in general the same phrase tag. On the other hand, in the case of VP VP for the construction of the main predicate and the auxiliary verb, the leftmost child node is exceptionally assigned as an anchor.

Then, we can find dependency relations between terminal nodes using the anchor information. The head is the anchor of the parent of the parent node of the current node (For example, terminal nodes 1, 2, 3, 5 and 7 in Figure 1). If the anchor of the parent of the parent node is the current node and if the parent of the parent node does not have the right sibling, the head is itself (the anchor of the root node and itself are same) (Terminal node 11), or the head is the anchor of its ancestor node where the anchor is changed from itself to other node (Terminal nodes 4, 6, 8 and 10). If the anchor of the parent of the parent node is the current node and if the parent of the parent node has another right sibling, the head is the anchor of the right sibling. The last condition is for the case of an auxiliary verb construction where the leftmost child node is assigned as an anchor. Assigning the lexical anchor and finding dependencies at the separated step enables arguments for the verb to be correctly dependent on the main verb and the main verb to be dependent on the auxiliary verb in the ambiguous annotation scheme in the SJTree.[1] Figure 1 shows the original SJTree phrase structure and its corresponding converted representation in dependency grammars.

3 Parsing Model

Our parsing model gives a probability to each possible dependency tree T for a sentence $S = e_1, e_2, ..., e_n$, where e_i is a Korean word. The model finally selects the dependency tree T^* that maximizes $P(T|S)$ as follows:

$$T^* = \arg\max_T P(T|S). \quad (1)$$

[1](Oh and Cha, 2010; Choi and Palmer, 2011) also introduced an conversion algorithm of dependency grammars for the SJTree. (Choi and Palmer, 2011) proposed head percolation rules for the SJTREE. However, we found some errors such as S related rules, where it gives lower priority to S than VP. It would fail to assign a head node correctly for S → VP S. Moreover, they did not consider auxiliary verb constructions annotated as VP in the SJTREE. According to their head rules, arguments for the main verb are dependent on the auxiliary verb instead of the main verb because of the annotation of the corpus (in general, VP → VP VP where the former VP in RHS is for the main verb and the latter VP is for the auxiliary verb). (Oh and Cha, 2010) corrected such ambiguities as a post-processing step (personal communication, August 2012).

We use the CKY algorithm to decode the dependency trees by employing bottom-up parsing and dynamic programming. $P(T|S)$ is defined as the product of probabilities as follows:

$$P(T|S) = \prod_{E_{pa} \in T} P(E_{pa}, dist|e_h), \tag{2}$$

where E_{pa} represents a clause dominated by a predicate or a genitive nominal phrase, e_h represents the head Korean word of E_{pa}, and $dist$ is the distance between E_{pa} and e_h. Instead of specifying the actual distance, it is classified into three bins: 1, 2 – 5, and 6 –. If the dependent Korean word appears right next to the head, the distance is 1. If it appears between 2 and 5, the distance is 2. If it appears past 6, the distance is 6. $P(T|S)$ is calculated in the similar way as (Kawahara and Kurohashi, 2006a). We describe the outline of this model below. Each probability in equation (2) is decomposed into two ways according to the type of E_{pa}. If E_{pa} is a clause dominated by a predicate, it is decomposed into a predicate-argument structure (content part) PA_m and a function part f_m. e_h is also decomposed into a content part c_h and a function part f_h.

$$\begin{aligned} P(E_{pa}, dist|e_h) &= P(PA_m, f_m, dist|c_h, f_h) \\ &= P(PA_m|f_m, dist, c_h, f_h) \times P(f_m, dist|c_h, f_h) \\ &\approx P(PA_m|f_m, c_h) \times P(f_m, dist|f_h) \end{aligned} \tag{3}$$

The first term in the last equation represents a fully-lexicalized generative probability of the predicate-argument structure. This probability is calculated based on automatically compiled case frames in the same way as (Kawahara and Kurohashi, 2006a). The second term of the last equation is a generative probability of function morphemes, in which f_m and f_h are defined as the POS patterns of morphemes for the predicate of E_{pa} and the head Korean word e_h, respectively. This probability is estimated from training data using maximum likelihood estimation. If E_{pa} is a genitive nominal phrase, it consists of a Korean word that is decomposed into c_m and f_m. Its probability is defined similarly as follows:

$$\begin{aligned} P(E_{pa}, dist|e_h) &= P(c_m, f_m, dist|c_h, f_h) \\ &= P(c_m|f_m, dist, c_h, f_h) \times P(f_m, dist|c_h, f_h) \\ &\approx P(c_m|c_h) \times P(f_m, dist|f_h). \end{aligned} \tag{4}$$

The first term in the last equation represents a fully-lexicalized generative probability of the genitive nominal phrase. This probability is calculated from the constructed database of N_1 *ui* N_2 (N_2 *of* N_1) structures. The second term is the same as the second term in equation (3). In our experiments, we use an unlexicalized parsing model as a baseline. This unlexicalized model regards the above lexicalized probabilities as uniform and actually calculates the product of generative probabilities of function morphemes, $P(f_m, dist|f_h)$.

4 POS Sequence Granularity

Given that Korean is an agglutinative language, a combination of Korean words is very productive and exponential. Actually, a larger dataset would not alleviate this issue. The number of POS patterns would not converge even with a corpus of 10 million words in the Sejong morphologically analyzed corpus. The wide range of POS patterns in words is mainly due to the fine-grained morphological analysis results, where they show all possible segmentations divided into lexical and functional morphemes. For example, most Korean language resources to represent Korean morphological analyses including the SJTree would analyze the word *kimkyosunim* ('Professor Kim+HON') as *kim*/NNP + *kyosu*/NNG + *nim*/XSN ('Kim + professor + HON'). Instead of keeping the fine-grained morphological analysis results, we simplify POS sequences as much as possible using the linguistically motivated method. It would be helpful if we can refine the dataset for a better representation of the relationship. We introduce four level POS granularity: PUNC, MERG, CONT and FUNC.

PUNC: Punctuation marks (denoted as SF for periods, question marks and exclamation points, SP for commas and SE for ellipsis) and non-punctuation marks (for example, a period in the number is equally denoted as SF such as 3/SN + ./SF + 14/SN) are distinguished. Recurrent punctuation marks such as .../SE + .../SE in the word are also merged into a single symbol.

MERG: Special characters such as mathematical characters denoted as SW are merged into an entity with adjacent morphemes. Other non-Korean characters such as SL (Roman letters), SH (Chinese characters) and SN (cardinal numbers) are either merged into an entity with adjacent morphemes or are considered as nouns when they appear alone. Secondly, all suffixes are merged with adjacent morphemes. Functional morpheme-related refining rules are described as follows with the number of occurrences in the SJTree.

The nominal prefix (XPN) and suffix (XSN) with adjacent morphemes are merged into the POS of the corresponding morphemes (17,955 cases). The noun derivational suffix (ETN) with precedent morphemes is merged into the noun (5,186 cases). The non-autonomous lexical root (XR) is merged into the following POS (5,322 cases). The verb and adjective derivational suffix (XSV and XSA) with precedent morphemes are merged into the verb and adjective (20,178 and 9,096 cases, respectively). The adjective and the adverbial derivation *gey*/EC are merged into the adverb (2,643 cases). Refinement rules are applied recursively to all POS tags until there are no rules to apply. For example, *soljk*/XR + *ha*/XSA + *gey*/EC is applied both according to the XR rule and the adverbial derivation rule to become *soljikhagey*/MAG ('frankly').

CONT: All content morphemes in the word are merged together. For example, the sequence of the different type of nouns in a word is merged as a single noun with the priority of proper noun (NNP) > common noun (NNG) > dependent noun (NNB). For example, the sequence of NNP and NNG such as *masan*/NNP + *yek*/NNG ('Masan station'), is merged into an NNP. The sequence of the different type of verbs in a word is also merged as a single verb. The difference between MERG and CONT is the nature of merged morphemes. MERG concerns about merging functional morphemes and CONT about merging lexical morphemes.

FUNC: All functional morphemes are merged together. For example, *eoss*/EP (PAST) + *da*/EF ('DECL') is merged into a single verbal ending *eossda*/EF.

5 Exploiting Lexical Information

This section aims at exploiting lexical information and proposes a lexicalized probabilistic model with case frames aggregated from predicate-argument structures and the database of *N* of *N* structures to improve the parsing system.

5.1 Constructing case frames

It is difficult to make wide-coverage predicate-argument structures manually. Therefore, it is necessary to compile them automatically from a large corpus for our purpose. We introduce two methods using POS patterns and parsed corpora to extract case frames automatically from a raw corpus. We then apply clustering to the extracted predicate-argument structures to produce case frames.

Firstly, we use POS patterns to select predicate-argument structures after automatically assigning POS tags to a raw corpus. The key criteria for determining the predicate-argument structures are the appearance of the final or conjunctive verbal endings (denoted as EC and EF, respectively). Using functional morphemes, we are able to detect the end of predicate-argument structures in the sentence. In Figure 1, we can find two case markers agglutinated to NPs for the predicate *naseo+ss+da*: *-ga* and *-ro* for nominative and adverbial case markers (JKS and JKB). Therefore, we can select the predicate-argument structure composed of *ungaro+ga* ('Ungaro+NOM') and *designer+ro* ('designer+AS') as arguments for the verb *naseo* ('work as'). Our algorithm for selecting predicate-argument structures using POS patterns is described below. All arguments with case markers except JKG (genitive) and JC (connective postpositions) are extracted as a predicate-argument structure. JX (auxiliary postpositions) are not extracted because they can be interpreted either nominative or accusative and it becomes ambiguous.

```
var pa
while $w_i$ in the sentence do
    if $w_i$ ends with case markers then
        pa += $w_i$;
    else if $w_i$ contains final or conjunctive verbal
    endings && pa is not NULL then
        print and initialize pa;
    else if $w_i$ contains other verbal endings then
        initialize pa;
    else
        do nothing;
    end
end
```

Secondly, to use parsed corpora, we employ the method proposed in (Kawahara and Kurohashi, 2006b) and re-implement it to extract case frames. A large corpus is automatically parsed and case frames are constructed from modifier-head examples in the resulting parsed corpus. Then we extract dependency lists depending on their head as follows. Dependency lists consist of *modifier*$_1$ ... *modifier*$_n$ *head* where $n >= 1$. Then, we select dependency lists only if the head is a predicate such as *unggaroga_6 dijaineoro_10 naseoeossda_11*.

Thereafter, predicate-argument structures are clustered to merge similar ones, as described in (Kawahara, 2005). We distinguish predicate-argument structures by the predicate and its closest case instance to the predicate as described in Figure 2[2]: In order to merge predicate-argument structures we introduce similarities between two structures calculated by the production of the similarities between arguments and the ratio of common instances.

$$Similarty_{case\_frames} = sim_{cf} \cdot ratio_{ci} \tag{5}$$

We use the semantic hierarchy of nouns from Korean CoreNet (Choi, 2003) for the similarities between two instances of arguments. CoreNet is composed of 2,937 concepts represented by *kortermnum* (k_{num}). A cipher of k_{num} tells a hierarchy depth. For instance, COUNTRY (k_{num}: 11125) has ORGANIZATION (1112) as a parent concept (hypernym). *hanguk* (11125, 'Korea) and *namhan* (11125, 'South Korea) share COUNTRY (11125) as a concept. Therefore, similarity between two instances is obtained as follows. $common$ is the shared length of k_{num} for i_1 and i_2[3]:

$$sim_{inst} = \frac{len_{knum}(common * 2)}{len_{knum}(i_1) + len_{knum}(i_2)} \tag{6}$$

Then, we calculate similarities between arguments of the same case marker in two predicate-argument structure as follows:

$$sim_{arg} = \frac{\sum_{x=1} \sum_{y=1} sim_{inst} \cdot \sqrt{|e_x||e_y|}}{\sum_{x=1} \sum_{y=1} \sqrt{|e_x||e_y|}} \tag{7}$$

where e_x and e_y are the number of the occurrences of the instance example e of the same case maker. The ratio of common instances is calculated as follows:

$$ratio_{ci} = \frac{\sum_{i=1} \sqrt{|e_x|}}{\sum_{j=1} \sqrt{|e_y|}} \tag{8}$$

where i is the number of the occurrences of the instance examples of the same case marker and j is the number of the occurrences of the instance examples of the all case marker.

5.2 Constructing the database of N_1 *ui* N_2 structures

We also integrate lexical information on Korean noun phrases of the form N_1 *ui* N_2, which roughly corresponds to N_2 *of* N_1 in English. Even though Korean genitive marker *ui* does not have a broad usage as much as *no* in Japanese as described in (Kurohashi and Sakai, 1999), it sometime does not modify the immediate constituent such as *Kyungjiui meylonhyang binwuleul* ('Melon-flavored soap of Kyungji') where *Kyungjiui* modifies *binwuleul* instead of *meylonhyang*. The N_1 *ui* N_2 structure is very useful to recognize the meaning of natural language text can improve head-modifier relationships between genitive nouns.

6 Experiment and Results

6.1 Parsing results

We use the Sejong Treebank corpus (SJTree) in our experiment.[4] We use standard dataset split for training, development and testing. We report here final evaluation results on the baseline unlexicalized parsing and different POS granularities. We crawl news articles published in 2007 from the website of *Chosun Ilbo*[5] (literally, 'Korea Daily News'), which is one of the major newspapers in Korea to integrate lexical information. We collect 212,401 pages and extract Korean sentences. We acquire a raw corpus with over three million sentences. Then, we use the Espresso POS Tagger and Dependency Parser for Korean to assign POS and parse sentences to extract POS patterned and parsed case frames.[6] We extract the database of

[2] {*inbu*$_3$} ('worker') means that the instance *inbu* has 3 occurrences.

[3] $common = 0$ if either i_1 or i_2 is not included in CoreNet.

[4] Differently from other Korean Treebank corpora, the SJTree contains non-sentences such as noun phrases. We select only complete sentences. We also remove erroneous sentences in the SJTree using heuristics such as non-defined POS tags and not-well-formed morpheme and POS tag pairs.

[5] http://www.chosun.com

[6] http://air.changwon.ac.kr/research/software

CF$_1$:	$\{inbu_3\}$:*i*	$\{cha_2, teuleok_1\}$:*ey*	$\{gabang_5\}$:*eul*	*silneunda.*
	{worker}:NOM	{car,truck}:LOC	{bag}:ACC	load
CF$_2$:		$\{teuleok_2\}$:*ey*	$\{jim_3\}$:*eul*	*silneunda.*
		{truck}:LOC	{baggage}:ACC	load

Figure 2: Predicate-argument structures distinguished by the predicate and its closest case instance

		UAS
Baseline system		71.735%
POS granularity	PUNC	73.714%
	MERG	76.993%
	CONT	81.515%
	FUNC	82.702%

Table 1: Evaluation results on unlexicalized parsing

		UAS
Lexical information	CF-parsed + NoN	86.037%
	CF-pos + NoN	86.433%

Table 2: Evaluation results on lexicalized parsing with FUNC

N_1 *ui* N_2 structures from the same corpus. During building case frame structures, we ignore JX postpositions (mostly for topic markers) which can be interpreted as either NOM or ACC. Instead, we explicitly specify this ambiguity in the input to let the parser consider both cases to select the correct one. For case frame structures extracted without the subject, we intentionally insert the dummy subject to represent the complete construction of the sentence without any missing constituents.

6.2 Discussion

The basic parsing model is directly based on the POS patterns of words. If some sentences have POS patterns that are not seen in the training dataset, our baseline system cannot handle them. By introducing POS sequence granularity we can increase recall and eventually it makes the dataset more parsable with less untrained POS sequences. Integrating lexical information is prominent. We can increase precision and it can fix many predicate-argument dependency errors in unlexicalized parsing. Results with case frames extracted from the automatically parsed corpus are slightly lower than results with POS patterned case frames because the nature of the corpus. The automatically parsed corpus contains inevitably much more errors than the POS tagged corpus. Moreover, the simpler method using POS patterns can guarantee less errors contained case frames. Filtering out erroneously parsed sentences and building case frame structures only using reliable sentences would yield better results.

Only small numbers of research projects about statistical parsing have been conducted using the same Treebank corpus. (Oh and Cha, 2010; Choi and Palmer, 2011) used the early version of the Sejong Treebank and obtained up to 86.01% F$_1$ score and 85.47% UAS, respectively. (Choi et al., 2012) obtained 78.74% F$_1$ score for phrase structure parsing. Our current results outperform previous work. We also test MaltParser[7] on the same dataset and we obtain 85.41% for UAS. It still shows the better performance of our proposed method. The advantage of our proposed system is the capability of adding lexicalized information from external corpora.

7 Conclusion

In this paper, we improved Korean dependency parsing accuracy using various factors, including POS granularity changes and lexical information. We refined the training dataset for a better representation of the relationship between words. We also introduced the use of lexical information. The accuracy was improved and it shows promising factors. The lexical knowledge extracted from a much bigger corpus would be interesting to pursue when seeking further improvement opportunities pertaining to the deep processing of Korean sentences.

Acknowledgments

This work was supported by the Industrial Technology International Cooperation Program (FT-1102, Creating Knowledge out of Interlinked Data) of MKE/KIAT, and the IT R&D program of MSIP/KEIT (10044494, WiseKB: Big data based self-evolving knowledge base and reasoning platform).

[7]http://www.maltparser.org

References

Marie Candito, Benoît Crabbé, and Pascal Denis. 2010. Statistical French Dependency Parsing: Treebank Conversion and First Results. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May. European Language Resources Association (ELRA).

Jinho D. Choi and Martha Palmer. 2011. Statistical Dependency Parsing in Korean: From Corpus Generation To Automatic Parsing. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 1–11, Dublin, Ireland, October. Association for Computational Linguistics.

DongHyun Choi, Jungyeul Park, and Key-Sun Choi. 2012. Korean Treebank Transformation for Parser Training. In *Proceedings of the ACL 2012 Joint Workshop on Statistical Parsing and Semantic Processing of Morphologically Rich Languages*, pages 78–88, Jeju, Republic of Korea, July 12. Association for Computational Linguistics.

Key-Sun Choi. 2003. CoreNet: Chinese-Japanese-Korean WordNet with Shared Semantic Hierarchy. In *Proceedings of Natural Language Processing and Knowledge Engineering*, pages 767–770, 26-29 October, 2003.

Tagyoung Chung, Matt Post, and Daniel Gildea. 2010. Factors Affecting the Accuracy of Korean Parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 49–57, Los Angeles, CA, USA, June. Association for Computational Linguistics.

Hoojung Chung. 2004. *Statistical Korean Dependency Parsing Model based on the Surface Contextual Information*. Ph.D. thesis, Korea University.

Alexander Gelbukh, Sulema Torres, and Hiram Calvo. 2005. Transforming a Constituency Treebank into a Dependency Treebank. *Procesamiento del Lenguaje Natural*, 35:145–152.

Daisuke Kawahara and Sadao Kurohashi. 2006a. A Fully-Lexicalized Probabilistic Model for Japanese Syntactic and Case Structure Analysis. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 176–183, New York City, USA, June. Association for Computational Linguistics.

Daisuke Kawahara and Sadao Kurohashi. 2006b. Case Frame Compilation from the Web using High-Performance Computing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC2006)*, pages 1344–1347.

Daisuke Kawahara. 2005. *Automatic Construction of Japanese Case Frames for Natural Langage Understanding*. Ph.D. thesis, Kyoto University, July.

Sadao Kurohashi and Yasuyuki Sakai. 1999. Semantic Analysis of Japanese Noun Phrases - A New Approach to Dictionary-Based Understanding. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 481–488, College Park, Maryland, USA, June. Association for Computational Linguistics.

Yong-Hun Lee and Jong-Hyeok Lee. 2008. Korean Parsing using Machine Learning Techniques. In *Proceedings of the Korea Computer Congress (KCC) 2008*, pages 285–288, Jeju, Korea.

Jin-Young Oh and Jeong-Won Cha. 2010. High Speed Korean Dependency Analysis Using Cascaded Chunking. *Simulation Journal*, 19(1):103–111.

Jungyeul Park. 2006. *Extraction automatique d'une grammaire d'arbres adjoints à partir d'un corpus arboré pour le coréen*. Ph.D. thesis, Université Paris 7 - Denis Diderot, mai.

Anoop Sarkar and Chung-Hye Han. 2002. Statistical Morphological Tagging and Parsing of Korean with an LTAG Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 6)*, pages 48–56, Venice, Italy.

Wen Wang. 2003. *Statistical Parsing and Language Modeling based on Constraint Dependency Grammar*. Ph.D. thesis, Purdue University, December.

Document Parsing: Towards Realistic Syntactic Analysis

Rebecca Dridan[♠], **Stephan Oepen**[♠♡]
[♠] University of Oslo, Department of Informatics
[♡] Potsdam University, Department of Linguistics
`{rdridan|oe}@ifi.uio.no`

Abstract

In this work we take a view of syntactic analysis as processing 'raw', running text instead of idealised, pre-segmented inputs—a task we dub *document parsing*. We observe the state of the art in sentence boundary detection and tokenisation, and their effects on syntactic parsing (for English), observing that common evaluation metrics are ill-suited for the comparison of an 'end-to-end' syntactic analysis pipeline. To provide a more informative assessment of performance levels and error propagation throughout the full pipeline, we propose a unified evaluation framework and gauge document parsing accuracies for common processors and data sets.

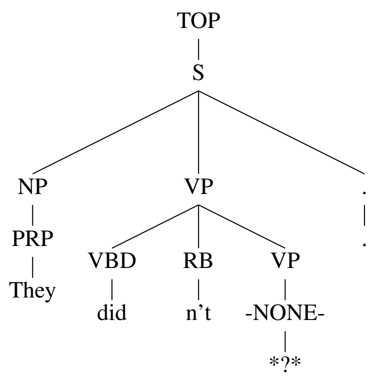
1 Introduction

The term *parsing* is used somewhat ambiguously in Natural Language Processing. In its 'pure', isolated interpretation, parsing maps from a sequence (or maybe a lattice) of token objects to syntactic analyses, say trees; from a more 'practical' point of view, however, parsing is also used to refer to a complete pipeline for syntactic analysis, starting with just running text (i.e. a string of characters) as its input. The two interpretations are of course closely related, but the underlying difference of perspective seems closely correlated with the distinction between parser *developers* vs. parser *users*. The parsing research literature and most contrastive benchmarking have their focus on parsing in isolation; at the same time, parsing software is most commonly applied to 'raw' string inputs. In this work, we take the practical, document parsing point of view and seek to inform parser users about configuration choices in a complete syntactic analysis pipeline, as well as about what they can expect in terms of end-to-end performance.

Parser evaluation, complementary to its utility in system development and scholarly comparison, has as one of its central purposes an aim of providing an indication of how a specific system or configuration might be expected to perform on unseen text. Genre and domain variation have been observed as important factors in predicting parser performance (Gildea, 2001; Zhang & Wang, 2009; Petrov & McDonald, 2012), but more fundamental issues related to pre-processing of parser inputs have largely been ignored.[1] That is, parser evaluation so far has been predominantly performed on perfectly segmented inputs. Indeed, the de-facto standard for evaluating phrase structure parsers is the PARSEVAL metric (Black et al., 1991, as implemented in the `evalb` tool), which assumes that both the gold standard and the parser output use the same tokenisation—an unrealistic idealisation from the user point of view.

Recent research (Dridan & Oepen, 2012; Read et al., 2012) has shown that many standard NLP systems, and even tools specialised for sentence and token segmentation perform well below 100% on these fundamental tasks. However, the effects of errors so early in the pipeline on end-to-end parser accuracy are currently unexplored. The goal of this paper is to discover and document these effects, and also to promote a complementary tradition of parsing real documents, rather than the idealised, pre-segmented token sequences used in parser development. For these purposes, we propose a unified evaluation perspective applicable across all sub-tasks of syntactic analysis (§ 2) and contrast it with earlier work on parser evaluation under ambiguous tokenisation (§ 3). In § 4 and § 5 we apply this framework to a number of parsing pipelines built from state-of-the-art components and demonstrate that differences in pre-processing can yield large differences in parser performance, varying across domains and parsers—larger in fact than incre-

[1]Until recently, it was in fact difficult to obtain raw, unsegmented text for the most widely used English parser evaluation data, Wall Street Journal articles from 1989.



⟨0, 12, TOP⟩ ⟨0, 12, S⟩ ⟨0, 4, NP⟩
⟨5, 11, VP⟩ ⟨11, 11, VP⟩

Figure 1: Parse tree and span labels for *They didn't.*

mental PARSEVAL improvements frequently reported for 'pure' statistical parsing in the past decade.

2 Robust Evaluation of Syntactic Analysis

Parsing gold-standard token sequences into trees, common training and testing data, and a standard evaluation method have given the research community a well-defined and stable task over the past two decades that has enabled great advances in statistical parsing. In order to evaluate document parsing, however, we need to generalise the evaluation method somewhat.

PARSEVAL evaluates spans of text defined by token index. If the tokens are no longer considered as given, this definition is invalid. We take the obvious step and opt for character-based span indexing, with the slight twist of using inter-character positions from the raw text, rather than character counts. This allows greater flexibility and precision in describing spans. Figure 1 shows a parse tree and the phrase structure spans extracted from it, assuming this sentence started at character position 0 in the document.

While character-based token indexing is a simple concept, it is complicated by the fact that both treebanks and parsers 'normalise' raw text in a variety of ways, including Unicode/ASCII mappings, quote disambiguation and bracket escaping. In order to calculate the character spans for a token and be able to match to a system output that might use a different normalisation, we have implemented a tool that aligns between the tokens of an analysis and the raw input to the parsing process. Since we are not assuming gold sentence segmentation, this requires aligning full documents, making a simple implementation of string edit distance, for example, intractable. Instead, we have implemented the space-efficient longest common subsequence algorithm of Myers (1986), and calculate the shortest edit script with consideration of the most commonly accepted mappings. This enables an efficient alignment that is even robust to occasional missing analyses (due to parse failure).

Phrase structure is not the only annotation that can be represented as a labelled span. Many standard NLP tasks, including part-of-speech (PoS) tagging, tokenisation, and sentence segmentation can also be viewed this way. For PoS tagging, the parallel is obvious. Tokenisation and sentence segmentation have previously been evaluated in a variety of manners (Palmer & Hearst, 1997; Kiss & Strunk, 2006; Read et al., 2012), but are also amenable to being seen as span labelling, allowing us to use a consistent generalised evaluation framework. Thus, we represent each annotation as a triple consisting of span start and end positions, together with a label. This is particularly useful for our present purpose of evaluating the effect of these pre-processing tasks on phrase structure parsing. Figure 2 shows all such triples that can be extracted from our running example.[2]

3 Previous Work

A character-based version of `evalb`, dubbed SParseval, was earlier used for parser evaluation over the output from speech recognisers, which cannot be expected (or easily forced) to adhere to gold standard sentence and token segmentation (Roark et al., 2006). As word recognition errors can lead to very different parse yields, this approach required a separate, external word alignment step, far more complicated than our in-built alignment stage.

Parsing morphologically rich languages is another area where expecting gold standard parse yields is unrealistic. In this case, parse yields are often morpheme sequences, ambiguously derived from the input strings. Tsarfaty et al. (2012) propose an eval-

[2]Note that, at this point, we include several elements of the gold-standard annotation of Figure 1 that are commonly suppressed in parser evaluation, notably the root category TOP, the empty node at position $\langle 11, 11\rangle$, and the final punctuation mark. In § 4 below, we return to some of these and reflect on the common practice of ignoring parts of the gold standard, against our goal of 'realistic' parser evaluation.

1:	⟨0, 4, POS:PRP⟩	2:	⟨0, 4, TOK⟩
3:	⟨5, 8, POS:VBD⟩	4:	⟨5, 8, TOK⟩
5:	⟨8, 11, POS:RB⟩	6:	⟨8, 11, TOK⟩
7:	⟨11, 11, POS:-NONE-⟩	8:	⟨11, 11, TOK⟩
9:	⟨11, 12, POS:.⟩	10:	⟨11, 12, TOK⟩
11:	⟨0, 12, TOP⟩	12:	⟨0, 12, S⟩
13:	⟨0, 4, NP-SBJ⟩	14:	⟨5, 11, VP⟩
15:	⟨11, 11, VP⟩	16:	⟨0, 12, SENT⟩

Figure 2: All triples from the tree in Figure 1.

uation metric (TEDEVAL) based on tree edit distance for Hebrew, rejecting character-based evalb as insufficient because it cannot (a) evaluate morphemes unrealised in the surface form; or (b) enforce only yields that break at whitespace and correspond to a path through the lattice of the morphological analyser. For languages like English, where the raw string is conventionally a concatenation of the parse yield, observation (a) is not an issue. However, we note that zero-length spans, made possible by our decision to record spans in terms of inter-character positions, can preserve linear precedence without requiring explicit realisation in the raw text. As for the second objection of Tsarfaty et al. (2012), that is specific to their situation of allowing ambiguous morphological segmentation only within pre-segmented tokens. In our more general case, we consider enumerating all possible tokenisations of a sentence (or document) neither practical or desirable, nor do we wish to enforce token breaks at whitespace, since different tokenisation conventions such as those described by Fares et al. (2013) allow mid-token spaces.

4 Experimental Setup

To evaluate end-to-end parsing, we selected three commonly used phrase structure parsers: the Charniak and Johnson reranking parser (C&J; Charniak & Johnson, 2005), the Berkeley parser (Petrov, Barrett, Thibaux, & Klein, 2006), and the Stanford parser (Klein & Manning, 2003). For each parser, we used the recommended pre-trained English model, and mostly default settings.[3] All parsers tokenise their input by default and the Stanford parser also includes a sentence splitting component. Since both the C&J and Berkeley parsers require sentence-segmented text, we pre-segmented for those parsers using `tokenizer`, the top-performing, lightweight, rule-based segmenter in the recent survey of Read et al. (2012).

For parser evaluation, the standard test data is Section 23 of the venerable WSJ portion of the Penn Treebank (PTB, Marcus et al., 1993). In addition to this data, we also use the full Brown portion of the PTB (often used for experiments in domain variation) and the relatively new English Web Treebank (EWTB, LDC #2012T13), which comprises several web genres. To evaluate *document parsing*, we start from the raw, running text. In the case of the EWTB this was provided with the treebank; for the other corpora, we use the raw texts distributed by Read et al. (2012). Corpora sizes are included in Table 1.

Our reference implementation for evaluation instantiates the triple-based framework outlined in § 2 above. For compatibility with much previous work, our tool supports the same configuration options as evalb, to discard tree nodes or consider different labels as equivalent. Somewhat ambivalently (since we experience these 'tweaks' as obstacles to transparency and replicability), we mimic the standard practice of discarding triples labelled TOP (the root category) and -NONE- (empty elements, absent in most parser outputs), as well as truncating labels at the first hyphen (i.e. ignoring function tags); furthermore, we allow the commonly accepted equivalence of ADVP and PRT, even though we have been unable to trace the origins of this convention. However, we do not follow the convention of ignoring tokens tagged as punctuation, for two reasons. First, Black et al. (1991) suggested ignoring punctuation for independently developed, hand-built parsers, to reduce variation across linguistic theories; in evaluating statistical parsers trained directly on PTB structures, however, it is natural to include syntactic information related to punctuation in the evaluation—as is common practice in data-driven dependency parsing since the 2007 CoNLL Shared Task. Second, the evalb technique of identifying punctuation nodes merely by PoS tags is problematic in the face of tagging errors; and even if one were to 'project' gold-standard PoS tags onto parser outputs, there would be ambiguity about how to count erroneous labels involving punctuation spans.

[3]We used the `-accurate` setting for the Berkeley parser, to improve coverage on our out-of-domain data. Full configuration details and other background, including an open-source implementation of our evaluation framework will be distributed through a companion website.

Corpus	# Gold Sentences	# Gold Tokens	C&J Gold	C&J Auto	Berkeley Gold	Berkeley Auto	Stanford Gold	Stanford Auto
Brown	13919	234375	82.7	81.2	82.2	82.3	77.1	77.7
EWTB	16622	254830	78.3	74.8	76.2	76.5	73.6	71.6
WSJ_{23}	2416	56684	90.9	86.4	89.8	88.4	85.4	83.8

Table 1: Phrase structure accuracy (labelled F_1) for three parsers (Charniak and Johnson, Berkeley and Stanford) over three different data sets, first using gold sentence segmentation and tokenisation, and then in the default string-input mode for each parser. Since only the Stanford parser performs sentence segmentation, the `tokenizer` sentence splitter was used to pre-process running text for the other two parsers.

	Sentences		Tokens							
Corpus	Tokenizer	Stanford	C&J		Berkeley		Stanford		REPP	
	F_1	F_1	F_1	SA	F_1	SA	F_1	SA	F_1	SA
Brown	96.0	95.7	98.3	77.0	99.5	85.8	99.5	86.1	99.6	86.5
EWTB	85.2	70.3	96.9	77.8	97.3	83.1	97.8	85.9	97.3	81.4
WSJ_{23}	94.4	93.5	99.1	86.8	99.6	94.8	99.6	94.9	99.9	98.0

Table 2: Preprocessing accuracy: Sentence segmentation is evaluated as the F_1 over sentence spans for `tokenizer` and the Stanford parser. Token accuracy is reported both as F_1 and sentence accuracy (SA), the percentage of sentences with completely correct tokenisation, for each of the parser-internal tokenisers, plus the stand-alone tokeniser REPP.

5 Experimental Results

End-to-end results of our first attempt at document parsing for each parser are shown in Table 1 (**Auto** columns), alongside the results of parsing idealised inputs with gold sentence and token segmentation (**Gold**). Accuracy is in terms of F_1 over phrase structure triples. The drop in F_1 from earlier published results on WSJ_{23} with gold tokens is due to our inclusion of punctuation and evaluation of all sentences, regardless of length.

We see a wide variation in how automatic preprocessing affects parser accuracy for the different parsers and domains. Focussing on WSJ_{23}, the in-domain set for all parser models, we see a drop in parser accuracy for all parsers when moving away from gold tokenisation. However, the size of the drop varies across parsers, with the C&J parser particularly affected. Indeed, with automatic pre-processing the Berkeley parser is now the more accurate for this data set. Over the out-of-domain sets, the C&J parser again loses accuracy switching from gold tokens, but the drop is not so severe, even though we would expect more pre-processing errors in, for example, the web text of EWTB.

For the Brown data set, the numbers show a slight *increase* in accuracy for both Berkeley and Stanford parsers when using automatically processed input, an unexpected effect. Berkeley again out-performs the C&J parser when not using gold tokens.

The low scores on EWTB are not surprising, since none of the parsers were tuned to this quite different genre, but the relative difference between idealised and document parsing is unexpected, giving how difficult one might expect this text to be for sentence segmentation and tokenisation. Again, we see that the Berkeley parser has actually improved in accuracy with automatic pre-processing, but the Stanford parser shows a similar drop to that it produced over WSJ_{23}.

In order to drill down to the reasons behind some of these variations, Table 2 shows the accuracy of the sentence segmentation and tokenisation used in the automatic configurations above.

Looking at sentence segmentation in Table 2, we see that accuracy is much lower over EWTB, reflecting the more informal nature of these texts, particularly with regards to capitalisation and punctuation. Our numbers are lower than in Read et al. (2012), due to differences in the evaluation method,[4] but we also see that `tokenizer` is the more accurate sentence splitter. The Stanford sentence segmentation is mod-

[4]While they evaluated only boundary points (i.e. sentence end points), we are interested in the effect of incorrect sentence segmentation on parsing and so evaluate the full sentence span (i.e. start and end points jointly). Thus, one incorrect sentence boundary point can lead to two incorrect sentence spans.

erately less accurate for the edited text, but radically under-segments the messier web text, returning 12914 sentences, compared to 14954 from `tokenizer`.

For tokenisation accuracy, there's very little difference between Berkeley and Stanford, which is not surprising, since the Berkeley parser tokeniser was built using the Stanford code. The in-built tokeniser in the C&J parser however is significantly less accurate. While the drop in F_1 doesn't seem too large, examining the sentence accuracy shows a much bigger effect. This echoes the findings of Dridan and Oepen (2012) which demonstrate that the built-in tokeniser of the C&J parser is well below state-of-the-art. For completeness, we also looked at their best performing tokeniser (REPP) on `tokenizer`-segmented sentences, adding those token accuracy numbers to Table 2. Here we see that relative performance of the better tokenisers varies with domain. As Dridan and Oepen (2012) found, REPP outperforms Stanford over Wall Street Journal text, but the performance difference over Brown is slight, and on EWTB, the advantage is to the Stanford tokeniser.

We re-parsed using the best tokeniser and sentence segmenter for each domain, with the results shown in Table 3. By comparing back to Table 1, we can see that there are a few times that the best pre-processing accuracy doesn't lead to the best parsing results, but the differences are slight. While the `tokenizer`+REPP combination gave the best performance for all parsers on WSJ_{23}, the results over Brown are surprising. The best input for each parser was the default, except for the C&J parser, where `tokenizer`+REPP led to the best results. The automatic tokenisers led to equal *or better* parse accuracy than using gold tokens. Our hypothesis to explain this phenomenon is that subtle differences in the tokenisation of the Brown corpus and the EWTB, compared to the WSJ portion of the PTB, confuse the parsers, which have been tuned not just to the WSJ text type, but also to the idiosyncrasies of its annotation. Giving the parsers 'more familiar' inputs can thus lead to more accurately bracketed analyses.

Sentence segmentation is still a major impact on the comparatively low parse accuracies on EWTB. Switching to `tokenizer` improved Stanford parser accuracy by almost 1 point, but further experiments using gold sentence boundaries with automatic tokenisation showed a boost of 2–4 points F_1, leaving further room for improvement there.

Corpus	C&J		Berkeley		Stanford	
	Gold	Auto	Gold	Auto	Gold	Auto
Brown	82.7	82.7	82.2	82.2	77.1	77.6
EWTB	78.3	77.7	76.2	76.3	73.6	72.4
WSJ_{23}	90.9	89.9	89.8	88.8	85.4	84.5

Table 3: Final document parsing phrase structure accuracies, showing the results of using the best combination of sentence segmentation and tokenisation for each domain. Gold tokenisation F_1numbers are repeated for easy comparison.

Overall, our results show that document parsing is a viable task, leading to parser accuracies only slightly below the state of the art over gold tokens for the (potentially overfitted) WSJ_{23}, given the right preprocessors. Furthermore, by actually evaluating the end-to-end pipeline, as well as the performance, in-situ, of the various preprocessors we have discovered that parser accuracy can actually be improved by matching the tokenisation to the expectations of the parser, rather than merely using the tokenisation of the treebank. This subtle tuning of annotation style, rather than domain could even help explain previous cross-domain results (McClosky, Charniak, & Johnson, 2006) that showed better accuracy from *not* using labelled Brown data for parser training.

6 Conclusions and Outlook

Through this work, we hope to call the syntactic analysis research community to arms and initiate a shift of emphasis towards practical use cases and *document parsing*—with the aim of mitigating the risks of overestimating parser performance and over-tuning to individual treebank idiosyncrasies. Our triple-based proposal for Robust Evaluation of Syntactic Analysis synthesises earlier work and existing metrics into a uniform framework that (a) encompasses the complete analysis pipeline; (b) enables comparison between analyses that use different normalisation conventions of the same text; and (c) allows one to quantify performance levels of individual sub-tasks as well as degrees of error propagation. Furthermore, we demonstrate how this setup supports 'mixing and matching' of gold-standard and system outputs, to isolate the effects of individual sub-tasks on end-to-end performance, which can bring to light unexpected effects.

Our evaluation framework is supported by an open-source reference implementation, with configuration options very similar to `evalb`, that can be used as a plug-in replacement too for evaluation of the complete syntactic analysis pipeline.[5] Immediate future work on this tool is planned to also allow evaluation of dependency analysis under ambiguous tokenisation by also evaluating tuples consisting of two character-defined sub-spans, plus a dependency label.

Acknowledgments

We are grateful to our colleagues at the Oslo Language Technology Group for many helpful discussions and suggestions, as well as to our three anonymous reviewers for insightful comments. This work is in part funded by the Norwegian Research Council through its WeSearch project. Large-scale experimentation is made possible through access to the ABEL high-performance computing facilities at the University of Oslo, and we are grateful to the Scientific Computing staff at UiO, as well as to the Norwegian Metacenter for Computational Science, and the Norwegian tax payer.

[5]Please see `http://www.delph-in.net/resa` for further information on how to obtain and run our tool.

References

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., … Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the workshop on speech and natural language* (p. 306 – 311). Pacific Grove, USA.

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics* (p. 173 – 180). Ann Arbor, MI, USA.

Dridan, R., & Oepen, S. (2012). Tokenization. Returning to a long solved problem. A survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics* (p. 378 – 382). Jeju, Republic of Korea.

Fares, M., Oepen, S., & Zhang, Y. (2013). Machine learning for high-quality tokenization. Replicating variable tokenization schemes. In *Computational linguistics and intelligent text processing* (p. 231 – 244). Springer.

Gildea, D. (2001). Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing* (p. 167 – 202). Pittsburgh, USA.

Kiss, T., & Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, *32*(4), 485 – 525.

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics* (p. 423 – 430). Sapporo, Japan.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpora of English: The Penn Treebank. *Computational Linguistics*, *19*, 313 – 330.

McClosky, D., Charniak, E., & Johnson, M. (2006). Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics* (p. 337 – 344). Sydney, Australia.

Myers, E. W. (1986). An O(ND) difference algorithm and its variations. *Algorithmica*, *1*(1-4), 251 – 266.

Palmer, D. D., & Hearst, M. A. (1997). Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, *23*(2), 242 – 267.

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics* (p. 433 – 440). Sydney, Australia.

Petrov, S., & McDonald, R. (2012). *Overview of the 2012 Shared Task on Parsing the Web.* Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language. Montreal, Canada.

Read, J., Dridan, R., Oepen, S., & Solberg, L. J. (2012). Sentence boundary detection: A long solved problem? In *Proceedings of the 24th International Conference on Computational Linguistics.* Mumbai, India.

Roark, B., Harper, M., Charniak, E., Dorr, B., Johnson, M., Kahn, J. G., … Yung, L. (2006). SParseval: Evaluation metrics for parsing speech. In *Proceedings of the 5th International Conference on Language Resources and Evaluation.* Genoa, Italy.

Tsarfaty, R., Nivre, J., & Andersson, E. (2012). Joint evaluation of morphological segmentation and syntactic parsing. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics* (p. 6 – 10). Jeju, Republic of Korea.

Zhang, Y., & Wang, R. (2009). Cross-domain dependency parsing using a deep linguistic grammar. In *Proceedings of the 47th Meeting of the Association for Computational Linguistics* (p. 378 – 386). Suntec, Singapore.

Neo-Davidsonian Semantics in Lexicalized Grammars

Petr Homola
Codesign, s.r.o.
Czech Republic
phomola@codesign.cz

Abstract

We present a method of semantic parsing within lexicalized grammars that generates neo-Davidsonian logical forms. We augment an existing LFG grammar with predicate logic formulae in the lexicon and logical annotations in the context-free rules to obtain logical forms of a sequence of sentences that can be used to construct a model, i.e., formulae with resolved anaphoric and indexical expressions, by skolemization and discourse resolution based on salience.

1 Introduction

Semantic parsing is important for many NLP applications. We present results of experiments with semantic parsing within the framework of Lexical-Functional Grammar (LFG), a well-researched unification-based formalism (Kaplan and Bresnan, 1982; Dalrymple et al., 1995a; Butt et al., 1999; Bresnan, 2001; Dalrymple, 2001; Nordlinger and Bresnan, 2011) for which wide-coverage grammars for a variety of typologically different languages are available.

The method we have developed can be used with any semantic formalism that can be expressed in first-order logic with equality (FOL). The approach of Parsons (1990) is usually called "neo-Davidsonian" because it is a variation of the event-based formal representation of Davidson (1967). The formalism used in this paper is a combination of that of Parsons and Hobbs (1985; 2003) who extends Davidson's approach to all predications. We show how simple and complex predicates are represented, how semantic representations can be incrementally created in LFG via codescription and how discourse and context (anaphoric and indexical expressions) can be treated. While we used LFG in the experiments, the method is flexible enough to be used in any rule-based grammar formalism based on context-free or categorial grammars such as (Uszkoreit, 1986) or (Kay, 1979; Kay, 1984).

In the next section we survey recent approaches to semantics in LFG. In Section 3 the neo-Davidsonian approach to semantics is presented. Section 4 expounds the way how neo-Davidsonian logical forms can be incrementally built up using LFG machinery. Section 5 focuses on how discourse models can be constructed from logical forms by resolving anaphoric and indexical expressions. Finally, we conclude in Section 6.

2 Related Work

Virtually all approaches to formal semantics assume the Principle of Compositionality, formally formulated by Partee (1995) as follows: "The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined." This means that semantic representation can be incrementally built up when constituents are put together during parsing. Since c(onstituent)-structure expresses sentence topology rather than grammatical relations, the rules that combine the meanings of subphrases frequently refer to the underlying syntactic structure, that is, f(unctional)-structure in LFG. Indeed, Halvorsen and Kaplan (1995) in their account of semantics within LFG define the s(emantic)-structure as a projection of the c-structure (through the correspondence function σ) but they refer to grammatical functions (GFs) by means of the compound function

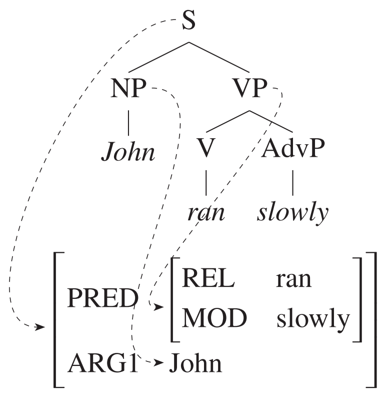
$\sigma\phi^{-1}$,[1] as in the following example:

(1) John ran slowly.

The corresponding lexical entry for the verb is

(2)
$$\begin{aligned}&(\sigma\mathcal{M} * \text{REL}) = \text{ran}\\&(\sigma\mathcal{M} * \text{ARG1}) = \sigma\phi^{-1}(\uparrow \text{SUBJ})\\&(\sigma\mathcal{M} * \text{ARG2}) = \sigma\phi^{-1}(\uparrow \text{OBJ})\end{aligned}$$

and the resulting correspondence between the c-structure and the s-structure is

(3)


Note that Halvorsen and Kaplan (1995) represent s-structures as feature structures (since they use functional annotations to construct them). (3) can be more conventionally expressed as $slowly(ran)(John)$.

Recent approaches to semantics in LFG are based on the so-called "glue semantics" (Dalrymple et al., 1993; Dalrymple et al., 1995b; Dalrymple, 2001). Consider the sentence

(4) Bill obviously kissed Hillary

Its semantic form is, according to Dalrymple et al. (1993), $obviously(kiss(Bill, Hillary))$. Glue semantics uses linear logic; lexical entries are assigned "meaning constructors" that consist of a logical expression and instructions for how the meaning is put together. For *to kiss*, for example, we have

(5)
$$\begin{aligned}&\forall X, Y.(f \text{ SUBJ})_\sigma \rightsquigarrow X \otimes (f \text{ OBJ})_\sigma \rightsquigarrow Y\\&\quad\multimap f_\sigma \rightsquigarrow kiss(X, Y)\end{aligned}$$

In words, (5) means that if the meaning of $(f \text{ SUBJ})_\sigma$ is X and the meaning of $(f \text{ OBJ})_\sigma$ is Y, then the meaning of f_σ is $kiss(X, Y)$. For brevity, meaning constructors are sometimes written as $[\![\textbf{word}]\!]$. For (4), then, we get

(6)
$$\begin{aligned}&[\![\textbf{Bill}]\!] \otimes [\![\textbf{obviously}]\!]\\&\otimes[\![\textbf{kissed}]\!] \otimes [\![\textbf{Hillary}]\!]\\&\vdash f_\sigma \rightsquigarrow obviously(kiss(Bill, Hillary))\end{aligned}$$

The idea behind glue semantics is that the lexicon and the rules for syntactic assembly provide meaning constructors that are interpreted as soon as all expressions on the left-hand side of the linear implication ($\multimap$) are available.[2]

Note that both Halvorsen and Kaplan (1995) and glue semantics use higher-order logic. In the next section we go on to outline an account of semantics that, while using codescription, relies on pure FOL for representation and on conjunction as the means of meaning assembly, as advocated on Minimalist grounds by Pietroski (2005).

3 Neo-Davidsonian Logical Representation

The sentence *John loves Mary* can be logically expressed (disregarding tense for the sake of simplicity) using a binary predicate for the verb and constants for its arguments:

(7) $love(John, Mary)$

Davidson (1967) has introduced "events" into the description of logical forms of sentences to be able to refer to "actions" by means of FOL (i.e., events are treated as individuals). We use the notation and terminology of Hobbs (1985; 2003) who introduced the "nominalization operator" and the term "eventuality" to refer to "possible events". The predicate *love* in (7) can be "nominalized" and we assume that the following equivalence holds:

(8) $love(x, y) \equiv \exists e.love'(e, x, y) \wedge Rexist(e)$

The newly introduced variable e is the eventuality of John's loving Mary and Hobbs' predicate $Rexist$

[1] ϕ is the correspondence function from c-structures to f-structures.

[2] More recent work on glue semantics uses a slightly different notation. (5) would be written as

$$\lambda X.\lambda Y.kiss(X, Y) : (\uparrow \text{SUBJ})_\sigma \multimap [(\uparrow \text{OBJ})_\sigma \multimap \uparrow_\sigma]$$

expresses that the eventuality is realized (this predicate is discussed in (Hobbs, 1985; Hobbs, 2003), we do not use it in the remainder of the paper).

Parsons (1990), too, uses events but he proposes unary predicates for actions and special predicates for their arguments. In this spirit, we use the following notation:

(9)
$$\begin{aligned} &love'(e,x,y) \equiv \\ &\equiv love''(e) \wedge actor(e,x) \wedge patient(e,y) \end{aligned}$$

Rather than θ-roles, we use what is called "protoroles" (Dowty, 1991), "tectogrammatical roles" (Sgall et al., 1986) or "roles on the action tier" (Jackendoff, 1990) in the literature on formal semantic analysis of natural languages. Thus $actor(e,x)$ means that x has the role "actor" in the eventuality e.

In the remainder of the paper, we refer to "neo-Davidsonian" formulae (with actions denoted by double-primed predicates) as *logical forms* (LF).[3]

4 Logical Forms in LFG

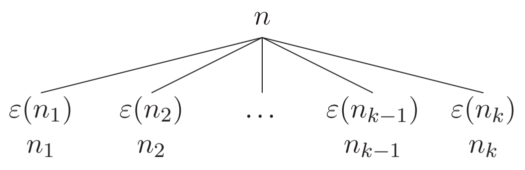
In LFG, parsing is driven by a context-free grammar that conforms to the X′-theory (Jackendoff, 1977). In this section we show how LFs can be incorporated formally within LFG. Notationally, we follow Kaplan (1995).[4]

In the formal architecture of LFG, N is the set of nodes, F is the set of f-structures, by Φ we denote the set of formulae (LFs) and V denotes the set of variables that may occur in LFs. In standard LFG, the mapping $M : N \to N$ maps nodes to their mother node and $\phi : N \to F$ maps nodes to f-structures. We introduce $\xi : N \to \Phi$ that maps nodes to formulae and $\tau : N \to V$ that maps nodes to variables.

For terminal nodes, ξ and τ are defined in the lexicon. For example, the morpholexical entry for the proper name *John* contains, besides the usual information, i.e., its category (N) and f-structure [PRED 'John'], a formula ($\xi(n) \overset{df}{=} x = \text{John}$) and a designated variable ($\tau(n) \overset{df}{=} x$ where x is the variable from $\xi(n)$). For a noun, such as *dog*, we have $\xi(n) \overset{df}{=} dog(x)$ and $\tau(n) \overset{df}{=} x$. Likewise, the morpholexical entry for the finite form of the verb *to see* contains its category, its f-structure [PRED 'see⟨(↑ SUBJ), (↑ OBJ)⟩', …], a formula ($\xi(n) \overset{df}{=} see''(e)$) and a designated variable ($\tau(n) \overset{df}{=} e$ where e is the variable from $\xi(n)$).

The formula of a nonterminal node is composed from the formulae of its daughter nodes. In LFG, context-free rules are augmented with functional annotations. Likewise, we augment them with logical annotations. Since Hobbs' (1985; 2003) "ontologically promiscuous" formulae are conjunctions of atomic formulae, we combine the formulae of the daughter nodes using the logical connective $\wedge$.

In (10), n_i are the daughter nodes of n and $\varepsilon(n_i)$ are the corresponding logical annotations. Note that $\{n_1, \ldots, n_k\} = M^{-1}(n)$.

(10)


Obviously, $M^{-1}(n) \neq \varnothing$ holds for n, for it is a nonterminal node. We define $\xi(n)$ for nonterminal nodes as follows:

(11)
$$\xi(n) = \bigwedge_{m \in M^{-1}(n)} \xi(m) \wedge \varepsilon(m)$$

We also introduce a new variable, $\tau(n)$. For ease of exposition, we give all formulae in an equivalent prenex normal form, i.e., $\mathrm{Q}_1 x_1 \ldots \mathrm{Q}_n x_n . \varphi$ where Q_i are quantifiers over variables x_i and φ is open (i.e., it contains no quantifiers).

To refer to terms in the formulae associated with nodes in the subtree induced by a rule, we use two metavariables in the logical annotations defined in the context of $\varepsilon(n_i)$ as follows:

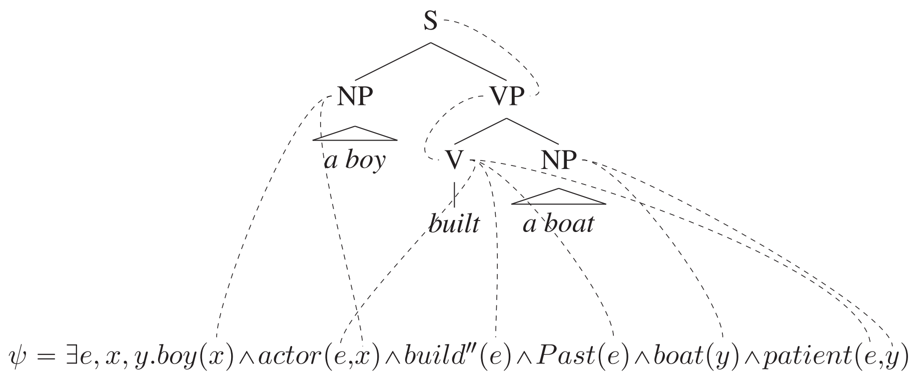
(12)
$$\begin{aligned} \triangle &= \tau(M(n_i)) \\ \nabla &= \tau(n_i) \end{aligned}$$

[3]We could go even further and reify the double-primed predicates along the scheme

$$P(e) \equiv act(p, e)$$

where p is an "action constant" that corresponds to the predicate P. While such a notation would surely be useful in formulating an axiomatic theory for reasoning, the difference between the notations has no impact on the presently described compositional mechanism.

[4]In the remainder of the paper we assume that we have a LFG grammar that produces c- and f-structures and rules out ill-formed sentences.



$$\psi = \exists e, x, y.boy(x) \wedge actor(e,x) \wedge build''(e) \wedge Past(e) \wedge boat(y) \wedge patient(e,y)$$

$$\varphi = \exists e_1, e_2, e_3, x, y.boy(x) \wedge actor(e_3,x) \wedge build''(e_1) \wedge Past(e_1) \wedge boat(y) \wedge patient(e_2,y) \wedge e_1{=}e_2 \wedge e_2{=}e_3$$

Figure 1: Logical form of *A boy built a boat*.

Thus $\triangle$ and $\triangledown$ are to logical annotations what $\uparrow$ and $\downarrow$ are to functional annotations.

Let us consider the sentence *A boy built a boat* (slightly adapted from (Hobbs, 1985)). The logical part of the lexicon, used to build up the LF of the sentece, is given in (13).

(13)

word	ξ	τ
a	$\top$	z
boy	$boy(x)$	x
boat	$boat(y)$	y
built	$build''(e_1) \wedge Past(e_1)$	e_1

Because of the definition (11) of ξ for nonterminal nodes, every node must have a formula (i.e., ξ is defined for all nodes), thus we use $\top$ as the formula of the article *a*.

The context-free rules and the corresponding functional and logical annotations are given in (14) (we omit the rule 'NP $\rightarrow$ Det N' as it is irrelevant for the present discussion).

(14)

$$\begin{array}{llcc}
\text{S} & \rightarrow & \text{NP} & \text{VP} \\
 & & (\uparrow \text{SUBJ}) = \downarrow & \uparrow = \downarrow \\
 & & actor(\triangle, \triangledown) & \triangle = \triangledown \\
 & & & \\
\text{VP} & \rightarrow & \text{V} & \text{NP} \\
 & & \uparrow = \downarrow & (\uparrow \text{OBJ}) = \downarrow \\
 & & \triangle = \triangledown & patient(\triangle, \triangledown)
\end{array}$$

The c-structure and the corresponding formula induced by the logical annotations (more precisely, its existential closure) are given in Figure 1. The prenex normal form we obtain is the formula φ but since $e_1 = e_2$ and $e_2 = e_3$, we can use $\psi = \varphi[e_1/e, e_2/e, e_3/e]$ where e is a newly introduced variable that embraces the eventualities e_1, e_2, e_3.

4.1 Complex Predicates

Complex predicates have been subject to research within LFG since the 1990s (Alsina, 1996; Alsina, 1997; Alsina et al., 1997; Broadwell, 2003). Consider the sentence *John made Mary cry* with a syntactically formed causative. The sentence is represented by one f-structure (i.e., the f-structures of *made* and *cry* are unified and the corresponding nodes are coheads) with a complex (hierarchically organized) PRED value:

(15) $\quad$ cause⟨(↑SUBJ),cry⟨(↑OBJ)⟩⟩

The f-structure of *John made Mary cry* is syntactically monoclausal:

(16)

$$\left[\begin{array}{ll}
\text{PRED} & \text{cause}\langle(\uparrow\text{SUBJ}),\text{cry}\langle(\uparrow\text{OBJ})\rangle\rangle \\
\text{SUBJ} & [\text{“John”}] \\
\text{OBJ} & [\text{“Mary”}]
\end{array}\right]$$

The LF of *John made Mary cry* is given in (17). Note that two eventualities are introduced (e_1, e_2) as the expression, although syntactically monoclausal, is semantically complex.

(17)

$$\begin{array}{l}
\exists e_1, e_2.cause''(e_1) \wedge cry''(e_2) \wedge \\
\wedge actor(e_1, John) \wedge patient(e_1, e_2) \wedge \\
\wedge actor(e_2, Mary)
\end{array}$$

Alsina (1997) proposes the creation of complex PRED values using the so-called restriction operator (Kaplan and Wedekind, 1991;

Kaplan and Wedekind, 1993; Wedekind and Kaplan, 1993). Note that as for LFs, no special machinery is needed, syntactically formed complex predicates can by modelled by rules using ordinary logical annotations.

4.2 Syntactic Pivots

The sentences

(18) a. John sold a car to Paul.
b. Paul bought a car from John.

have the same content if interpreted according to a theory that models the common relationship between *to sell* and *to buy*. However

(19) a. I made John sell a car to Paul.
b. I made Paul buy a car from John.

differ in their content although both entail (18a) and (18b). To express the different perspectives of (18a) and (18b) we use the predicate $Pivot$ that distinguishes the individual around which a clause revolves syntactically ($Pivot(e, John)$ and $Pivot(e, Paul)$ in (18a) and (18b), respectively).[5]

Similarly, adverbial modifiers are pivot-sensitive. Consider the following two sentences:

(20) a. John painted Mary.
b. Mary was painted by John.

Using the approach sketched above, we arrive at the following LF:

(21) $\exists e.paint''(e) \wedge Past(e) \wedge$
$\wedge actor(e, John) \wedge patient(e, Mary)$

While (21) represents the meaning of both (20a) and (20b), note that the meanings of the two sentences differ if we add the phrase *with great pleasure* because the corresponding predicate $with\_great\_pleasure$ modifies not only the eventuality but also the entity which is the syntactic pivot of the sentence. In English, the syntactic pivot is the subject, i.e., $John$ in (20a) and $Mary$ in (20b). To express the difference between the two sentences, we add the conjunction $Pivot(e, x) \wedge x = y$ to the formulae where x is a newly introduced variable and y is the variable associated with the subject.[6]

[5]For a description of pivots at the level of f-structure see (Falk, 1998; Falk, 2000).

[6]Note that LFs need not be purely semantic (for this reason we called them *logical* rather then *semantic*). Pivots are a syntactic notion but they affect semantic interpretation. This is consonant with the approach of Hobbs (2003) who uses, for example, a syntactic predicate (nn) to represent noun-noun compounds in LFs.

5 From Logical Forms to Models

To obtain a model from a sequence of LFs, we have to resolve all anaphoric and indexical expressions and replace all existentially quantified variables with constants. In contrast to parsing, the conversion of LFs into a model was implemented procedurally in our experiments.

Anaphoric expressions refer to an entity in the same sentence or in a preceding one. Anaphora resolution uses morphological (agreement), syntactic (switch-reference), semantic (logical acceptability) or pragmatic (topic/focus articulation) information provided by c-structures, f-structures and i-structures (King, 1997).

According to Kaplan (1979; 1989), the character (linguistic meaning) of an indexical expression is a function from contexts that delivers the expression's content at each context. We assume that the information about the current speaker, hearer, time, and place is available to the system and this contextual information is used to resolve expressions such as *I, you, my, here, now, Past*, etc.

Consider the following two sentences that represent a simple mini-discourse (a coherent sequence of utterances):

(22) a. I saw a dog.
b. The dog barked.

The corresponding LFs, without any intersentential relations, are given in (23).

(23) a. $\exists e_1, x.see''(e_1) \wedge Past(e_1) \wedge$
$\wedge actor(e_1, I) \wedge patient(e_1, x) \wedge$
$\wedge dog(x)$
b. $\exists e_2, y.bark''(e_2) \wedge Past(e_2) \wedge$
$\wedge actor(e_2, y) \wedge dog(y)$

To convert the LFs to a model, we instantiate the existentially bound variables with constants that satisfy the LFs using the information provided by f-structures (morphological, syntactic, semantic, and pragmatic). For e_1 and e_2, we introduce constants c_{e_1} and c_{e_2}. The constant I represents the indexical pronoun *I*. We use contextual information to replace it with a proper name, say *Peter*. The predicate $Past$ is indexical, too. We replace $Past(c_{e_i})$ with $Timepoint(c_{e_i}, t_i) \wedge Before(t_i, n)$ where t_i is a newly introduced constant which represents a time point, and n is the constant for "now" (taken from the context). x is replaced with c_x, a newly introduced constant.

In the second sentence, y is recognized as referentially anchored (we sketch the algorithm for anaphora resolution in the following subsection) and replaced by c_x. The resolved formulae are given in (24) and the situation described in (22) is their conjunction. Since Hobbs (1985; 2003) does not use the connective $\neg$, the formulae are always satisfiable, i.e., a model always exists.

(24) a. $see''(c_{e_1}) \wedge Timepoint(c_{e_1}, t_1) \wedge Before(t_1, n) \wedge actor(c_{e_1}, Peter) \wedge \wedge patient(c_{e_1}, c_x) \wedge dog(c_x)$
b. $bark''(c_{e_2}) \wedge Timepoint(c_{e_2}, t_2) \wedge \wedge Before(t_2, n) \wedge actor(c_{e_2}, c_x) \wedge \wedge dog(c_x)$

5.1 Anaphora Resolution

The parser operates on isolated sentences, disregarding intersentential coreferences. To resolve intrasentential and intersentential anaphora in coherent sequences of utterances, we formalize salience of entities at the level of pragmatics.

For the purposes of this subsection we assume that we have a discourse that consists of sentences $s_1, \ldots, s_n$ and the corresponding feature structures $f_1, \ldots, f_n$. An entity is a feature structure that represents a person, an object or an event. Every entity has a special attribute, INDEX, to represent coreferences.

The discourse context is formally a list of indices (values of the INDEX attribute) $C = \langle i_1, \ldots, i_m \rangle$. The sentences are processed one by one. We start with $C = \emptyset$. For every f-structure f_i, we proceed as follows:

1. For every entity in f_i, we try to find its referent in C (using available morphological, syntactic, semantic, and pragmatic information). If a referent was found for an entity, its index in C is moved to the front of the list. Otherwise, a new index is assigned to the entity and prepended to the list.
2. The indices of entities that belong to the focus are moved to the front of C because of their high salience.[7]

6 Conclusions

The paper presents a method of integrating neo-Davidsonian event semantics within the formal framework of LFG. We showed how logical forms can be obtained from LFG rules augmented with logical annotations. Further, we showed how anaphoric and indexical expressions can be resolved so that we can construct a (possibly partial) model of the processed text.

Our approach differs from glue semantics in using conjunction instead of linear logic for meaning assembly. Moreover, unlike glue semantics we do not use higher-order logic in semantic forms for practical reasons (mainly in order to be able to use FOL-based theorem provers and model builders for reasoning over LFs).

In further work we will extend our grammar to account for more phenomena and in the next step we will focus on inferential entailment in models.

References

Alex Alsina, Joan Bresnan, and Peter Sells. 1997. Complex Predicates: Structure and Theory. In Alex Alsina, Joan Bresnan, and Peter Sells, editors, *Complex Predicates*, pages 1–12.

Alex Alsina. 1996. *The Role of Argument Structure in Grammar: Evidence from Romance*. CSLI Publications.

Alex Alsina. 1997. A Theory of Complex Predicates: Evidence from Causatives in Bantu and Romance. In Alex Alsina, Joan Bresnan, and Peter Sells, editors, *Complex Predicates*, pages 203–246.

Joan Bresnan. 2001. *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics, New York.

George Aaron Broadwell. 2003. Complex predication and parallel structures in optimality-theoretic syntax. Handout from a talk given at the 7th Workshop on Optimality Theory Syntax in Nijmegen, Netherlands.

Miriam Butt, Tracy Holloway King, María-Eugenia Niño, and Frédérique Segond. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

[7]We assume that the parser provides information about topic/focus articulation. We use i-structures (King, 1997).

Mary Dalrymple, John Lamping, and Vijat Saraswat. 1993. LFG Semantics via Constraints. In *Proceedings of Sixth Meeting of the European ACL*.

Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen. 1995a. *Formal Issues in Lexical-Functional Grammar*. CSLI Publications.

Mary Dalrymple, John Lamping, Fernando Pereira, and Vijat Saraswat. 1995b. Linear logic for meaning assembly. In *Proceedings of Computational Logic for Natural Language Processing*.

Mary Dalrymple. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press.

Donald Davidson. 1967. The logical form of action sentences. In *The logic of decision and action*, pages 81–95. University of Pittsburg Press.

David Dowty. 1991. Thematic Proto-Roles and Argument Selection. *Language*, 67:547–619.

Yehuda N. Falk. 1998. On Pivots and Subjects. MS, Department of English, The Hebrew University of Jerusalem.

Yehuda N. Falk. 2000. Pivots and the Theory of Grammatical Functions. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG Conference*.

Per-Kristian Halvorsen and Ronald M. Kaplan. 1995. Projections and Semantic Description in Lexical-Functional Grammar. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, pages 279–292. CSLI.

Jerry R. Hobbs. 1985. Ontological Promiscuity. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 61–69.

Jerry R. Hobbs. 2003. Discourse and Inference. MS, Information Sciences Institute, University of Southern California, Marina del Rey.

Ray Jackendoff. 1977. *X-Bar Syntax: A Study of Phrase Structure*. MIT Press, Cambridge, Massachusetts.

Ray Jackendoff. 1990. *Semantic Structures*. MIT Press, Cambridge, Massachusetts.

Ronald M. Kaplan and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *Mental Representation of Grammatical Relations*. MIT Press, Cambridge.

Ronald M. Kaplan and Jürgen Wedekind. 1991. Restriction and Structure Misalignment. MS, Xerox Palo Alto Research Center.

Ronald M. Kaplan and Jürgen Wedekind. 1993. Restriction and Correspondence-based Translation. In *Proceedings of the 6th EACL Conference*, pages 193–202.

David Kaplan. 1979. On the Logic of Demonstratives. *Journal of Philosophical Logic*, 8(1):81–98.

David Kaplan. 1989. Demonstratives: An Essay on the Semantics, Logic, Metaphysics, and Epistemology of Demonstratives and Other Indexicals. In *Themes from Kaplan*, pages 481–563. Oxford University Press.

Ronald M. Kaplan. 1995. The Formal Architecture of Lexical-Functional Grammar. In *Formal Issues in Lexical-Functional Grammar*, pages 7–28. CSLI Publications.

Martin Kay. 1979. Functional Grammar. In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*.

Martin Kay. 1984. Functional Unification Grammar: a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting of the Association for Computational Linguistics*.

Tracy Holloway King. 1997. Focus Domains and Information Structure. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG Conference*.

Rachel Nordlinger and Joan Bresnan. 2011. Lexical-Functional Grammar: Interactions Between Morphology and Syntax. In Robert D. Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 112–140. Wiley-Blackwell Publishing.

Terence Parsons. 1990. *Events in the Semantics of English: A Study in Subatomic Semantics*. MIT Press.

Barbara H. Partee. 1995. Lexical Semantics and Compositionality. In Lila R. Gleitman and Mark Liberman, editors, *Invitation to Cognitive Science. Language*, volume 1, pages 311–360. MIT Press.

Paul M. Pietroski. 2005. *Events and Semantic Architecture*. Oxford University Press.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. D. Reider Publishing Company.

Hans Uszkoreit. 1986. Categorial Unification Grammars. In *Proceedings of COLING*.

Jürgen Wedekind and Ronald M. Kaplan. 1993. Type-driven Semantic Interpretation of F-Structures. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*.

Effective Parsing for Human Aided NLP Systems

Naman Jain **Sambhav Jain** **Dipti Misra Sharma**
Language Technologies Research Centre
IIIT Hyderabad
naman.jain@students.iiit.ac.in, sambhav.jain@research.iiit.ac.in
dipti@iiit.ac.in

Abstract

In this paper, we present our efforts towards identifying probable incorrect edges and then suggesting *k*-best alternates for the same in a typed-dependency framework. Such a setup is beneficial in human aided NLP systems where the decisions are largely automated with minimal human intervention. Minimizing the human intervention calls for automatic identification of ambiguous cases. We have employed an entropy based confusion measure to capture uncertainty exerted by the parser oracle and later flag the highly uncertain predictions. To further assist human decisions, *k*-best alternatives are supplied in the order of their likelihood. Our experiments, conducted for Hindi, establish the effectiveness of the proposed approach towards increasing the label accuracy with economically viable manual intervention. This work leads to new directions for parser development and also in the human-aided NLP systems.

1 Introduction

Last decade has witnessed an increasing interest in dependency-based syntactic analysis of sentences (Tsarfaty et al., 2013). It is noticed that morphologically rich and free word order languages are better handled using the dependency based framework than the constituency based one (Mel'čuk, 1988; Bharati et al., 1995).

The fundamental notion of dependency is based on the idea that the syntactic structure of a sentence consists of binary asymmetrical relations between the words, termed as dependencies. In a typed dependency framework, the relation between a pair of words, is marked by a *dependency label*, where one of the nodes is *head* and other is *dependent* (Tesnière and Fourquet, 1959). Figure 1 shows an example sentence from Hindi, along with syntactic relations and dependency labels[1] marked along the edges.

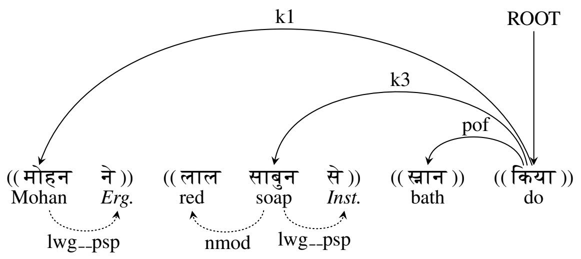


Figure 1: Dependency Tree with Syntactic Relations and Labels.

A major goal of dependency parsing research is to develop quality parsers, which can provide reliable syntactic analysis for various NLP applications such as natural language generation (Koller and Striegnitz, 2002), machine translation (Ding and Palmer, 2004), ontology construction (Snow et al., 2004), etc. Despite extensive advancements in parsing research, it is observed that parsers perform clumsily when incorporated in NLP applications (Kolachina and Kolachina, 2012). The remedies addressing the shortcomings in the past have adopted building further high quality parsers with domain adaptations (Blitzer et al., 2006; McClosky et al., 2006). However, it is practically impossible to account for all the domains and build an ideal universal parser. This has been a major reason for exploring Human Aided NLP systems which aims at providing quality output with minimal human intervention for crucial decisions.

The practical impact of parsing errors, at applica-

[1]k1: Doer, k3: Instrument, k7p: Place, pof: Part-of (complex predicate), ccof: co-ordination and sub-ordination, nmod: Noun Modifier, lwg__psp: Local-word-group post-position

tion's end, may be more severe as depicted by the accuracy of a parser. Popel (2011), in the context of machine translation, pointed out that an acutely incorrect parse can degrade the quality of output.

In this paper, we explore human assisted automated parsing, with human intervention limited to only those cases which are difficult for the statistical model (oracle) to disambiguate. Our focus has been to minimize human intervention in terms of effort and time. The scheme involves running a data driven dependency parser and later involving a human validation step for the probably incorrect decisions which are also identified and flagged by the system.

Minimizing the human intervention calls for automatic identification of ambiguous cases. We have employed an entropy based confusion measure to capture uncertainty exerted by the parser oracle and later flag the highly uncertain predictions. To further assist human decision, we also provide *k* probable alternatives in the order of their likelihood. In all, the approach comprises of the following two steps:-

- Identification of probable incorrect predictions.
- Selection of *k*-best alternates.

2 Background and Motivation

We have worked with Hindi, a relatively free-word-order and morphologically rich, Indo-Aryan language. In previous attempts to parse Hindi(Ambati et al., 2010), it has been observed that UAS[2] is greater than LS[3] by $\sim$ 6%, which is reconfirmed by our baseline parser (later described in Section 5) where UAS is 6.22% more than LS. The UAS in our baseline is well above 90% (92.44%) while the LS is still 86.21%. This drives us to focus on improving LS, to boost the overall accuracy(LA[4]) of the parser.

Dependency annotation scheme followed in Hindi Dependency Treebank (Bhatt et al., 2009) consists tag-set of $\sim$ 95 dependency labels which is comparatively larger than the tag-set for other languages[5], like Arabic($\sim$ 10), English($\sim$ 55), German($\sim$ 45) etc. This apparently is a major reason behind the observed gap between LS and UAS for Hindi parsing. One of the frequent labeling errors that the parser makes is observed to be between closely related dependency tags, for eg. $k7$ (abstract location) and $k7p$ (physical location) are often interchangeably marked (Singla et al., 2012). We have reasons to believe that such a decision is comparatively tougher for an automatic parser to disambiguate than a human validator.

In the past, annotation process has benefited from techniques like Active Learning(Osborne and Baldridge, 2004) where unannotated instances exhibiting high confusions can be prioritized for manual annotation. However, in Active Learning, the annotators or validators generally have no information about the potentially wrong sub-parts of a parse and thus full parse needs to be validated. Even if the the annotators are guided to smaller components (as in Sassano and Kurohashi (2010)), the potentially correct alternates are not endowed. In our approach the validator is informed about the edges which are likely to be incorrect and to further assist the correction k best potential label-replacements are also furnished. So, effectively just partial corrections are required and only in worst case (when a correction triggers correction for other nodes also) a full sentence needs to be analyzed. The efforts saved in our process are tough to be quantified, but the following example provides a fair idea of efficacy of our proposition. In figure 2, second parse has information of the probable incorrect label and also has 2 options to correct the incorrect label to guide a human validator.

(1) ... जेलों में समस्या हल करने ...
... prisons in problems solve do ...
... *solve problems in prisons ...*

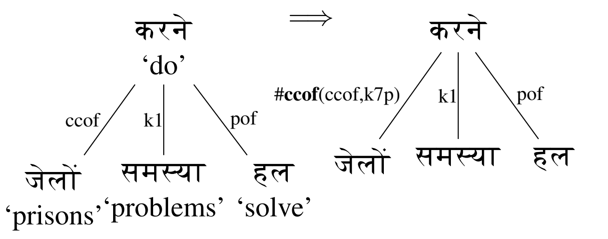


Figure 2: Example showing output from conventional parser v/s output from our approach. Arc-label with '#' represents incorrect arc label (confusion score > θ) along with 2-best probable arc labels.

[2]UAS = Unlabeled Attachment Score
[3]LS = Label Accuracy Score
[4]LA = Labeled Attachment Score
[5]As observed on the CoNLL-X and CoNLL2007 data for the shared tasks on dependency parsing.

3 Methodology

Recently there has been focus on the confidence estimation of attachments and arc-labels of a dependency parse. Mejer and Crammer (2012) have worked with MSTParser (McDonald et al., 2005) to give confidence scores for attachments while Jain and Agrawal (2013) have worked with MaltParser (Nivre et al., 2007) to render the confusion scores for arc-labels. Since our focus is on arc-labels we follow the approach proposed in Jain and Agrawal (2013). They captured the confusion exerted on the parser's oracle while predicting a parser action and propagated it to the arc-label of the dependency tree. The quantification of confusion is done by calculating entropy with the class membership probabilities of the parser actions.

We obtained the confusion score for each arc-label in our data. Next, we obtained a threshold ($\theta = 0.137$) for which the maximum F_1-$score$ is observed for incorrect label identification on the development set(Figure 3). In figure 2, the edge with the label 'ccof' has been flagged (#) because the confusion score is greater than θ, which signifies that it is probably incorrect. The proposition is indeed correct as the correct label is 'k7p' instead of 'ccof'.

The additional details about the correctness of an arc-label, can duly indicate the cases where the probability of the arc-label to be incorrect is high. In our efforts to minimize the human intervention, we propose to subject the reviewer only to the cases where the confusion score is above θ. At this stage the reviewer will be required to judge if the flagged label is indeed incorrect and if it is, then choose the corresponding correct label among all the remaining labels.

To further assist human decision, we also provide *k* probable alternatives in the order of their likelihood as proposed by the oracle. The reason behind this hypothesis is that it is likely that the correct label exists among the top label candidates. This, potentially, can give quick alternates to the reviewer for choosing the correct label and thereby speedup the review process.

4 *k*-Best Dependency Labels for the Flagged Arc-Labels

The likelihood of the arc-labels is obtained and ranked using the following three strategies:-

- $Voting$: The list of predicted labels, using voting mechanism, is sorted in decreasing order of number of votes, obtained during classification. The label with maximum number of votes is emitted as the resultant dependency label in the output parse. Broadly, this can be viewed as predicting *1*-best label using *voting* strategy which can easily be extended to predict *k*-best labels.

- $Probability$: The calculation of confusion scores demand for class membership probabilities for arc-label (refer section 3). The posterior probabilities for the candidate labels can also be alternatively used to emit out the resultant dependency label. Similar to voting scheme, the labels are sorted in decreasing order of their probabilities. The sorted list of predicated labels may differ from that of *voting* mechanism, which motivate us to consider *probability* for choosing the *k*-best dependency labels.

- $Voting + Probability$: A tie can occur between two or more labels in the list of *k*-best candidate labels if their votes/posterior probabilities are same. However, the phenomenon is unlikely in case of probabilities due to the real valued nature calculated up-to 10 decimal places. On the other hand *votes* are integer-values ($\{0, ..., {}^{n}C_2\}$, where *n* is number of labels) and are much more susceptible to ties. The tie in voting can be resolved using complement information from probabilities (and vice-versa).

5 Experiments

In our experiments, we focus on correctly establishing dependency relations between the chunk[6] heads which we henceforth refer as *inter-chunk* parsing. The relations between the tokens of a chunk (*intra-chunk* dependencies) are not considered for experimentation. The decision is driven by the fact that the *intra-chunk* dependencies can easily be predicated automatically using a finite set of rules (Kosaraju et al., 2012). Moreover we also observed the high learnability of *intra-chunk* relations from a pilot experiment. We found the accuracies of *intra-chunk* dependencies to

[6]A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

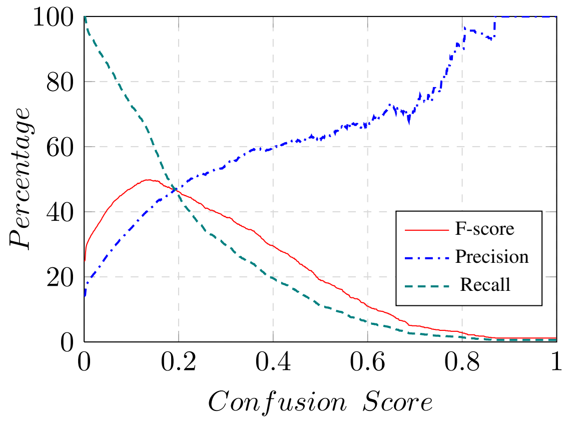


Figure 3: *Precision*, *Recall* and F_1-*score* for various values of confusion score on 'Hindi' development set.

be more than 99.00% for both LA and UAS.

Each experiment assumes the availability of a human expert for validation of the machine parsed data, who, when queried for a potential incorrect edge label, responds with the correct edge label. The experiments aim to measure the assistance provided to human expert by our approach. We varied the list of k-best labels from k=1 to k=5.

We setup a baseline parser on the lines of Singla et al. (2012) with minor modifications in the parser *feature model*. We employ MaltParser version-1.7 and Nivre's Arc Eager algorithm for all our experiments reported in this work. All the results reported for overall parsing accuracy are evaluated using *eval07.pl*[7]. We use MTPIL (Sharma et al., 2012) dependency parsing shared task data. Among the features available in the FEATS column of the CoNLL format data, we only consider *Tense, Aspect, Modality (tam)*, *post-positions(vib)* and *chunkId* while training the baseline parser. Other columns like POS, LEMMA, etc. are used as such.

In case of typed-dependency parsing, the accuracy can be LA, UAS or LS. However, in our case, we are focusing on the correct prediction of arc-labels, the results are on LS. In terms of strategies mentioned in Section 4, baseline system is generated using *Voting* strategy with $k = 1$. The LS is 86.21% as shown in Table 1.

[7]http://nextens.uvt.nl/depparse-wiki/SoftwarePage/#eval07.pl

6 Evaluation and Discussion

The evaluation of an interactive parse correction is a complicated task due to intricate cognitive, physical and conditional factors associated with a human annotator. Since a human may not match the consistency of a machine, we have to resort to few compelling assumptions which would give an idea of the approximate benefit from our proposed approach. We have assumed a perfect human oracle who always identifies incorrect label and picks the correct label from the available k-best list, if correct label is present in the list. The simulation of the perfect human oracle is done using the gold annotation. It is also assumed that the decision of the correct label can be taken with the information of local context and the whole sentence is not reviewed(which is not always true in case of a human annotator). This gives the upper bound of the accuracies that can be reached with our approach. The validation of the results obtained by automatic evaluation is done by performing a separate human evaluation for 100 nodes with the highest confusion score.

In our dataset, we found $\sim$23% (4,902 edges) of total (21,165) edges having confusion score above θ and thus marked as potentially incorrect arc-labels. Table 1 exhibits LS improved by perfect human oracle, for k-best experiments where k=1 to 5 on $\sim$23% potentially incorrect identified arc-labels.

k	Voting(%)	Probability(%)	Voting+Probability(%)
1	86.21	**86.35**	86.28
2	90.86	**90.96**	90.91
3	92.13	**92.24**	92.18
4	92.72	**92.86**	92.74
5	92.97	**93.16**	93.04

Table 1: k-Best improved LS on inspecting $\sim$23% ($> \theta$) edges.

Table 1 also depicts that as the value of k increases, the label accuracy also increases. The best results are obtained for *Probability* scheme. There is a substantial increment in LS moving from 1-best to 2-best in all the schemes. The amount of gain, however, decreases with increase in k.

Ideally to achieve maximum possible LS, all the edges should be reviewed. Table 2 confirms that if all the edges are reviewed, an LS of 93.18% to 96.57% is achievable for k, ranging over 2 to 5. But practically this would be too costly in terms of time and effort. In order to economize, we wish to only review the cases

k	**Voting(%)**	**Probability(%)**	**Voting+Probability(%)**
1	86.21	**86.35**	86.28
2	93.19	93.18	**93.26**
3	95.10	95.11	**95.14**
4	95.94	**96.06**	95.97
5	96.41	**96.57**	96.49

Table 2: *k*-Best improved LS on inspecting 100% edges.

which are probable enough to be incorrect. Confusion scores give a prioritized list of edges, which dictates the cases that should be dispatched first for review. To relate the review cost against LS gain we present a metric AGI$_x$ defined as:-

AGI$_x$:*"Accuracy Gain on Inspecting top $x\%$ edges"* corresponds to the ratio of accuracy gain from baseline by inspecting top $x\%$ of total edges, when sorted in decreasing order of their *confusion score*. The metric takes into account the human effort that goes into validation or revision, and thus gives a better overview of ROI(Return on Investment).

$$AGI_x = \frac{\text{Accuracy after validating top } x\% \text{ edges} - \text{Baseline accuracy}}{x}$$

From Table 1 and Table 2 we observe for *k* = 2 and *probability* scheme that the improved LSs are 90.96% and 93.18% on inspecting 23% and 100% edges respectively. Although the latter is greater than former by ∼ 2% but this additional increment requires an extra inspection of additional ∼77% edges, which is economically inviable. The fact is better captured in Table 3, where AGI$_{23}$ subdues AGI$_{100}$ for different values of *k* using different 'schemes'.

Further to incorporate the fact that *'larger the candidate list more will be the human efforts required to pick the correct label'*, we also present the results of **AGI$_x$/k**, which can govern the choice of *k*, best suited in practice. While taking this into account, we assume that the human efforts are inversely proportional to *k*. Results for **AGI$_{23}$/k** on improved LS, over all the experiments are reported in Table 4.

As shown in Table 3, AGI$_{23}$ increases with increase in the value of *k*, but it is practically inefficient to keep large value of *k*. Optimum choice of *k* is observed to be 2 from the metric AGI$_x$/k, as shown in Table 4, where the maximum value for **AGI$_{23}$/k** is ∼ 0.10 for all the 'schemes', which corresponds *k* = 2.

	Voting		**Probability**		**Voting+Probability**	
k	**AGI$_{23}$**	**AGI$_{100}$**	**AGI$_{23}$**	**AGI$_{100}$**	**AGI$_{23}$**	**AGI$_{100}$**
1	0.0000	0.0000	**0.0060**	0.0014	**0.0030**	0.0007
2	**0.2008**	0.0698	**0.2051**	0.0697	**0.2029**	0.0705
3	**0.2556**	0.0889	**0.2604**	0.0890	**0.2578**	0.0893
4	**0.2811**	0.0973	**0.2871**	0.0985	**0.2820**	0.0976
5	**0.2919**	0.1020	**0.3001**	0.1036	**0.2949**	0.1028

Table 3: ***AGI$_{23}$*** and ***AGI$_{100}$*** for *k*=1 to 5

k	**AGI$_{23}$/k** (Voting)	**AGI$_{23}$/k** (Probability)	**AGI$_{23}$/k** (Voting+Probability)
1	0.0000	0.0060	0.0030
2	**0.1004**	**0.1025**	**0.1015**
3	0.0852	0.0868	0.0859
4	0.0703	0.0718	0.0705
5	0.0584	0.0600	0.0590

Table 4: ***AGI$_{23}$/k*** for *k*=1 to 5

From the above analysis, we can establish that with 2 probable alternatives, a perfect human oracle can increase the LS by 4.61%, inspecting top ∼ 23% of total edges. The corresponding LA increase is 4.14%(earlier 83.39% to now 87.53%).

The validation of the observation is done by a human expert who confirmed of the assistance from the above methodology over the default procedure. He was given with 2-best alternatives for the top 100 edges that are obtained using *probability* scheme. The LS gain on his evaluation is approximately 10% which matches the expected gain.

7 Conclusion

In this paper we explored the possibility of human intervention to achieve higher accuracies in parsing. A major hurdle in the process is to effectively utilize the valuable human resources. We employed an entropy based confusion measure to capture uncertainty exerted by the parser oracle and later flag the highly uncertain labels. We further asserted that with 2 probable alternatives, a human expert can increase the label accuracy by 4.61%, inspecting ∼ 23% of total edges. In future we would also like to study the effectiveness of our approach on attachment validation in parsing.

References

A. Bharati, V. Chaitanya, R. Sangal, and KV Ramakrishnamacharyulu. 1995. *Natural Language Processing: A Paninian Perspective*. Prentice-Hall of India.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 17–24.

Avihai Mejer and Koby Crammer. 2012. Are you sure?: confidence in prediction of dependency tree edges. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 573–576. Association for Computational Linguistics.

Bharat Ram Ambati, Samar Husain, Sambhav Jain, Dipti Misra Sharma, and Rajeev Sangal. 2010. Two methods to incorporate local morphosyntactic features in Hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 22–30. Association for Computational Linguistics.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics.

Dipti Misra Sharma, Prashanth Mannem, Joseph vanGenabith, Sobha Lalitha Devi, Radhika Mamidi, and Ranjani Parthasarathi, editors. 2012. *Proceedings of the Workshop on Machine Translation and Parsing in Indian Languages*. The COLING 2012 Organizing Committee, Mumbai, India, December.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kubler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128. Association for Computational Linguistics.

Karan Singla, Aniruddha Tammewar, Naman Jain, and Sambhav Jain. 2012. Two-stage approach for hindi dependency parsing using maltparser. *Training*, 12041(268,093):22–27.

Lucien Tesnière and Jean Fourquet. 1959. *Eléments de syntaxe structurale*, volume 1965. Klincksieck Paris.

Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for japanese dependency parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 356–365.

Martin Popel, David Mareček, Nathan Green, and Zdeněk Žabokrtskỳ. 2011. Influence of parser choice on dependency-based MT. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 433–439. Association for Computational Linguistics.

Mel'čuk. 1988. *Dependency syntax: theory and practice*. State University Press of New York.

Miles Osborne and Jason Baldridge. 2004. Ensemble-based active learning for parse selection. In *HLT-NAACL*, pages 89–96.

Prudhvi Kosaraju, Samar Husain, Bharat Ram Ambati, Dipti Misra Sharma, and Rajeev Sangal. 2012. Intra-chunk dependency annotation: expanding Hindi inter-chunk annotated treebank. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 49–56. Association for Computational Linguistics.

R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D.M. Sharma, and F. Xia. 2009. A multi-representational and multi-layered treebank for hindi/urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics.

Reut Tsarfaty, Djamé Seddah, Sandra Kübler, and Joakim Nivre. 2013. Parsing Morphologically Rich Languages: Introduction to the Special Issue. *Computational Linguistics*, 39(1):15–22.

Rion Snow, Daniel Jurafsky, and Andrew Y Ng. 2004. Learning syntactic patterns for automatic hypernym discovery. *Advances in Neural Information Processing Systems 17*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Sambhav Jain and Bhasha Agrawal. 2013. A dynamic confusion score for dependency arc labels. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1237–1242, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

Sudheer Kolachina and Prasanth Kolachina. 2012. Parsing any domain english text to conll dependencies. In *LREC*, pages 3873–3880.

Yuan Ding and Martha Palmer. 2004. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical mt. In *Workshop on Recent Advances in Dependency Grammars (COLING)*, pages 90–97.