IWPT-05

Proceedings of the Ninth International Workshop on Parsing Technologies

9–10 October 2005
Vancouver, British Columbia, Canada

Production and Manufacturing by
Omnipress Inc.
Post Office Box 7214
Madison, WI 53707-7214

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
75 Paterson Street, Suite 9
New Brunswick, NJ 08901
USA
Tel: +1-732-342-9100
Fax: +1-732-342-9339
acl@aclweb.org

Preface

The 9^{th} International Workshop on Parsing Technologies continues the tradition that was started with the first workshop with that name, organized by Masaru Tomita in 1989 in Pittsburgh. IWPT'89 was followed by seven workshops in a biennial rhythm, only slightly disturbed by fear of millenium problems in 1999:

- IWPT'91 in Cancun, Mexico
- IWPT'93 in Tilburg, The Netherlands
- IWPT'95 in Prague, Czech Republic
- IWPT'97 in Cambridge (MA), USA
- IWPT 2000 in Trento, Italy
- IWPT 2001 in Beijing, China
- IWPT 2003 in Nancy, France

Over the years, the IWPT workshops have become the major forum for researchers in natural language parsing to meet and discuss advances in the field. Based on these workshops, four books on parsing technologies have been published by Kluwer Academic Publishers. The most recent one, based on IWPT 2000 and IWPT'01 was published last year (*New Developments in Parsing Technology,* Harry Bunt, John Carroll and Giorgio Satta, editors).

In 1994 the Special Interest Group on Parsing (SIGPARSE) was set up within ACL with the primary aim to give continuity to the IWPT series, and ever since the IWPT workshops have taken place under the auspices of SIGPARSE. IWPT 2005 also marks the return of IWPT to North America; the combined conferences on Human Language Technology and Empirical Methods in Natural Language Processing providing an excellent opportunity for this. I would like to thank Alon Lavie and the ACL organisation, in particular Priscilla Rasmussen, for their wonderful and professional help in organizing IWPT 2005 as a satellite event of the joint HLT/EMNLP conference.

Thanks are also due to the members of the Program Committee, and in particular to chairman Rob Malouf, for the careful work in reviewing submitted papers and designing the workshop program. Of a total of 42 submitted papers, 17 were accepted for presentation as a full paper, which gives an acceptance rate of 40%.

Harry Bunt
SIGPARSE Officer and IWPT 2005 General Chair

General Chair:

Harry Bunt (Tilburg University, Netherlands)

Program Chair:

Robert Malouf (San Diego State University, USA)

Logistic Arrangements Chair:

Alon Lavie (Carnegie-Mellon University, Pittsburgh, USA)

Program Committee:

Harry Bunt (Tilburg University, Netherlands)
Bob Carpenter (Alias-i, Inc., Brooklyn, NY, USA)
John Carroll (University of Sussex, Brighton, UK)
Eugene Charniak (Brown University, Providence, RI, USA)
Aravind Joshi (University of Pennsylvania, Philadelphia, USA)
Ronald Kaplan (Xerox Palo Alto Research Center, USA)
Martin Kay (Xerox Palo Alto Research Center, USA)
Dan Klein (UC Berkeley, USA)
Sadao Kurohashi (University of Tokyo, Japan)
Alon Lavie (Carnegie-Mellon University, Pittsburgh, USA)
Robert Malouf (San Diego State University, USA)
Yuji Matsumoto (Nara Institute of Science and Technology, Japan)
Paola Merlo (University of Geneva, Switzerland)
Bob Moore (Microsoft, Redmond, USA)
Anton Nijholt (University of Twente, Netherlands)
Gertjan van Noord (University of Groningen, Netherlands)
Stephan Oepen (University of Oslo, Norway)
Stefan Riezler (Xerox Palo Alto Research Center, USA)
Giorgio Satta (University of Padua, Italy)
Khalil Sima'an (University of Amsterdam, Netherlands)
Mark Steedman (University of Edinburgh, UK)
Hozumi Tanaka (Chukyo University, Japan)
Hans Uszkoreit (DFKI and Univeritat des Saarlandes, Germany)
Eric Villemonte de la Clergerie (INRIA, Rocquencourt, France)
K. Vijay-Shanker (University of Delaware, Newark, USA)
Dekai Wu (Hong Kong University of Science and Technology, China)

Additional Reviewers:

Rieks op de Akker (Twente University, Netherlands)

Daisuke Kawahara (University of Tokyo, Japan)
Manabu Okumura (Tokyo Institute of Technology, Japan)
Kenji Sagae (Carnegie-Mellon University, USA)
Libin Shen (University of Pennsylvania, USA)
Kiyoaki Shirai (Hokuriku Advanced Institute of Science and Technology, Japan)

Invited Speakers:

Jill Burstein (Educational Testing Service, USA)
Mari Ostendorf (University of Washington, USA)

Table of Contents

Short papers

Efficient and robust LFG parsing: SXLFG

Pierre Boullier and Benoît Sagot
INRIA-Rocquencourt, Projet Atoll,
Domaine de Voluceau, Rocquencourt B.P. 105
78 153 Le Chesnay Cedex, France
{pierre.boullier, benoit.sagot}@inria.fr

Abstract

In this paper, we introduce a new parser, called SXLFG, based on the *Lexical-Functional Grammars* formalism (*LFG*). We describe the underlying context-free parser and how functional structures are efficiently computed on top of the CFG shared forest thanks to computation sharing, lazy evaluation, and compact data representation. We then present various error recovery techniques we implemented in order to build a robust parser. Finally, we offer concrete results when SXLFG is used with an existing grammar for French. We show that our parser is both efficient and robust, although the grammar is very ambiguous.

1 Introduction

In order to tackle the algorithmic difficulties of parsers when applied to real-life corpora, it is nowadays usual to apply robust and efficient methods such as Markovian techniques or finite automata. These methods are perfectly suited for a large number of applications that do not rely on a complex representation of the sentence. However, the descriptive expressivity of resulting analyses is far below what is needed to represent, e.g., phrases or long-distance dependencies in a way that is consistent with serious linguistic definitions of these concepts. For this reason, we designed a parser that is compatible with a linguistic theory, namely LFG, as well as robust and efficient despite the high variability of language production.

Developing a new parser for LFG (*Lexical-Functional Grammars*, see, e.g., (Kaplan, 1989)) is not in itself very original. Several LFG parsers already exist, including those of (Andrews, 1990) or (Briffault et al., 1997). However, the most famous LFG system is undoubtedly the Xerox Linguistics Environment (XLE) project which is the successor of the Grammars Writer's Workbench (Kaplan and Maxwell, 1994; Riezler et al., 2002; Kaplan et al., 2004). XLE is a large project which concentrates a lot of linguistic and computational technology, relies on a similar point of view on the balance between shallow and deep parsing, and has been successfully used to parse large unrestricted corpora.

Nevertheless, these parsers do not always use in the most extensive way all existing algorithmic techniques of computation sharing and compact information representation that make it possible to write an efficient LFG parser, despite the fact that the LFG formalism, as many other formalisms relying on unification, is NP-hard. Of course our purpose is not to make a new XLE system but to study how robustness and efficiency can be reached in LFG parsing on raw text.

Building constituent structures (*c-structures*) does not raise any particular problem in theory,[1] because they are described in LFG by a context-free grammar (CFG), called *(CF) backbone* in this paper. Indeed, general parsing algorithms for CFGs are well-known (Earley, GLR,...). On the other hand, the efficient construction of functional structures (*f-structures*) is much more problematic. The first choice that a parser designer must face is that of when f-structures are processed: either during CF

[1] In practice, the availability of a *good* parser is sometimes less straightforward.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 1–10,
Vancouver, October 2005.

parsing (interleaved method) or in a second phase (two-pass computation). The second choice is between f-structures evaluation on single individual [sub-]parses ([sub-]trees) or on a complete representation of all parses. We choose to process all phrasal constraints by a CF parser which produces a shared forest[2] of polynomial size in polynomial time. Second, this shared forest is used, as a whole, to decide which functional constraints to process. For ambiguous CF backbones, this two pass computation is more efficient than interleaving phrasal and functional constraints.[3] Another advantage of this two pass vision is that the CF parser may be easily replaced by another one. It may also be replaced by a more powerful parser.[4] We choose to evaluate functional constraints directly on the shared forest since it has been proven (See (Maxwell and Kaplan, 1993)), as one can easily expect, that techniques which evaluate functional constraints on an enumeration of the resulting phrase-structure trees are a computational disaster. This article explores the computation of f-structures directly (without unfolding) on shared forests. We will see how, in some cases, our parser allows to deal with potential combinatorial explosion. Moreover, at all levels, error recovering mechanisms turn our system into a robust parser.

Our parser, called SXLFG, has been evaluated with two large-coverage grammars for French, on corpora of various genres. In the last section of this paper, we present quantitative results of SXLFG using one of these grammars on a general journalistic corpus.

2 The SXLFG parser: plain parsing

This section describes the parsing process for fully grammatical sentences. Error recovery mechanisms, that are used when this is not the case, are described in the next section.

[2]Informally, a shared forest is a structure which can represent a set of parses (even an unbounded number) in a way that shares all common sub-parses.

[3]This fact can be easily understood by considering that functional constraints may be constructed in exponential time on a sub-forest that may well be discarded later on by (future) phrasal constraints.

[4]For example, we next plan to use an RCG backbone (see (Boullier, 2004) for an introduction to RCGs), with the functional constraints being evaluated on the shared forest output by an RCG parser.

2.1 Architecture overview

The core of SXLFG is a general CF parser that processes the CF backbone of the LFG. It is an Earley-like parser that relies on an underlying left-corner automaton and is an evolution of (Boullier, 2003). The set of analyses produced by this parser is represented by a shared parse forest. In fact, this parse forest may itself be seen as a CFG whose productions are instantiated productions of the backbone.[5] The evaluation of the functional equations is performed during a bottom-up left-to-right walk in this forest. A disambiguation module, which discards unselected f-structures, may be invoked on any node of the forest including of course its root node.

The input of the parser is a word lattice (all words being known by the lexicon, including special words representing unknown tokens of the raw text). This lattice is converted by the *lexer* in a lexeme lattice (a lexeme being here a CFG terminal symbol associated with underspecified f-structures).

2.2 The context-free parser

The evolutions of the Earley parser compared to that described in (Boullier, 2003) are of two kinds: it accepts lattices (or DAGs) as input and it has syntactic error recovery mechanisms. This second point will be examined in section 3.1. Dealing with DAGs as input does not require, at least from a theoretical standpoint, considerable changes in the Earley algorithm.[6] Since the Earley parser is guided by a left-corner finite automaton that defines a regular super-set of the CF backbone language, this automaton also deals with DAGs as input (this corresponds to an intersection of two finite automata).

[5]If A is a non-terminal symbol of the backbone, A_{ij} is an instantiated non-terminal symbol if and only if $A_{ij} \overset{+}{\underset{G}{\Rightarrow}} a_{i+1} \dots a_j$ where $w = a_1 \dots a_n$ is the input string and $\overset{+}{\underset{G}{\Rightarrow}}$ the transitive closure of the *derives* relation.

[6]If i is a node of the DAG and if we have a transition on the terminal t to the node j (without any loss in generality, we can suppose that $j > i$) and if the Earley item $[A \rightarrow \alpha.t\beta, k]$ is an element of the table $T[i]$, then we can add to the table $T[j]$ the item $[A \rightarrow \alpha t.\beta, k]$ if it is not already there. One must take care to begin a PREDICTOR phase in a $T[j]$ table only if all Earley phases (PREDICTOR, COMPLETOR and SCANNER) have already been performed in all tables $T[i]$, $i < j$.

2.3 F-Structures computation

As noted in (Kaplan and Bresnan, 1982), if the number of CF parses (c-structures) grows exponentially w.r.t. the length of the input, it takes exponential time to build and check the associated f-structures. Our experience shows that the CF backbone for large LFGs may be highly ambiguous (cf. Section 4). This means that (full) parsing of long sentences would be intractable. Although in CF parsing an exponential (or even unbounded) number of parse trees can be computed and packed in polynomial time in a shared forest, the same result cannot be achieved with f-structures for several reasons.[7] However, this intractable behavior (and many others) may well not occur in practical NL applications, or some techniques (See Section 2.4) may be applied to restrict this combinatorial explosion.

Efficient computation of unification-based structures on a shared forest is still a evolving research field. However, this problem is simplified if structures are monotonic, as is the case in LFG. In such a case the support (i.e., the shared forest) does not need to be modified during the functional equation resolution. If we adopt a bottom-up left-to-right traversal strategy in the shared forest, information in f-structures is cumulated in a synthesized way. This means that the evaluation of a sub-forest[8] is only performed once, even when this sub-forest is shared by several parent nodes. In fact, the effect of a complete functional evaluation is to associate to each node of the parse forest a set of partial f-structures which only depends upon the descendants of that node (excluding its parent or sister nodes).

The result of our LFG parser is the set of (complete and consistent, if possible) *main f-structures* (i.e., the f-structures associated to the root of the shared forest), or, when a partial analysis occurs, the sets of (partial) f-structures which are associated with *maximal* internal nodes). Such sets of (partial or not) f-structures could be factored out in a single f-structure containing disjunctive values, as in XLE. We decided not to use these complex disjunctive values, except for some atomic types, but rather to associate to any (partial) f-structure a unique identifier: two identical f-structures will always have the same identifier throughout the whole process. Experiments (not reported here) show that this strategy is worth using and that the total number of f-structures built during a complete parse remains very tractable, except maybe in some pathological cases.

As in XLE, we use a lazy copying strategy during unification. When two f-structures are unified, we only copy their *common* parts which are needed to check whether these f-structures are unifiable. This restricts the quantity of copies between two daughter nodes to the parts where they interact. Of course, the original daughter f-structures are left untouched (and thus can be reused in another context).

2.4 Internal and global disambiguation

Applications of parsing systems often need a disambiguated result, thus calling for disambiguation techniques to be applied on the ambiguous output of parsers such as SXLFG. In our case, this implies developing disambiguation procedures in order to choose the most likely one(s) amongst the main f-structures. Afterwards, the shared forest is pruned, retaining only c-structures that are compatible with the chosen main f-structure(s).

On the other hand, on any internal node of the forest, a possibly huge number of f-structures may be computed. If nothing is done, these numerous structures may lead to a combinatorial explosion that prevents parsing from terminating in a reasonable time. Therefore, it seems sensible to allow the grammar designer to point out in his or her grammar a set of non-terminal symbols that have a linguistic property of (quasi-)saturation, making it possible to apply on them disambiguation techniques.[9] Hence, some non-terminals of the CF backbone that correspond

[7] As an example, it is possible, in LFG, to define f-structures which encode individual parses. If a polynomial sized shared forest represents an exponential number of parses, the number of different f-structures associated to the root of the shared forest would be that exponential number of parses. In other words, there are cases where no computational sharing of f-structures is possible.

[8] If the CF backbone G is cyclic (i.e., $\exists A$ s.t. $A \overset{+}{\underset{G}{\Rightarrow}} A$), the forest may be a general graph, and not only a DAG. Though our CF parser supports this case, we exclude it in SXLFG. Of course this (little) restriction does not mean that cyclic f-structures are also prohibited. SXLFG does support cyclic f-structures, which can be an elegant way to represent some linguistic relations.

[9] Such an approach is indeed more satisfying than a blind *skimming* that stops the full processing of the sentence whenever the amount of time or memory spent on a sentence exceeds a user-specified limit, replacing it by a partial processing that performs a bounded amount of work on each remaining non-terminal (Riezler et al., 2002; Kaplan et al., 2004).

to linguistically *saturated* phrases may be associated with an ordered list of disambiguation methods, each of these non-terminals having its own list. This allows for swift filtering out on relevant internal nodes of f-structures that could arguably only lead to inconsistent and/or incomplete main f-structures, or that would be discarded later on by applying the same method on the main f-structures. Concomitantly, this leads to a significant improvement of parsing times. This view is a generalization of the classical disambiguation method described above, since the pruning of f-structures (and incidentally of the forest itself) is not reserved any more to the axiom of the CF backbone. We call *global disambiguation* the pruning of the main f-structures, and *internal disambiguation* the same process applied on internal nodes of the forest. It must be noticed that neither disambiguation type necessarily leads to a unique f-structure. *Disambiguation* is merely a shortcut for *partial or total disambiguation*.

Disambiguation methods are generally divided into probabilistic and rule-based techniques. Our parsing architecture allows for implementing both kinds of methods, provided the computations can be performed on f-structures. It allows to associate a weight with all f-structures of a given instantiated non-terminal.[10] Applying a disambiguation rule consists in eliminating of all f-structures that are not optimal according to this rule. Each optional rule is applied in a cascading fashion (one can change the order, or even not apply them at all).

After this disambiguation mechanism on f-structures, the shared forest (that represent c-structures) is filtered out so as to correspond exactly to the f-structure(s) that have been kept. In particular, if the disambiguation is complete (only one f-structure has been kept), this filtering yields in general a unique c-structure (a tree).

[10]See (Kinyon, 2000) for an argumentation on the importance of performing disambiguation on structures such as TAG derivation trees or LFG f-structures and not constituent(-like) structures.

3 Techniques for robust parsing

3.1 Error recovery in the CF parser

The detection of an error in the Earley parser[11] can be caused by two different phenomena: the CF backbone has not a large enough coverage or the input is not in its language. Of course, although the parser cannot make the difference between both causes, parser and grammar developers must deal with them differently. In both cases, the parser has to be able to perform *recovery* so as to resume parsing as well as, if possible, to correctly parse valid portions of incorrect inputs, while preserving a *sensible* relation between these valid portions. Dealing with errors in parsers is a field of research that has been mostly addressed in the deterministic case and rarely in the case of general CF parsers.

We have implemented two recovery strategies in our Earley parser, that are tried one after the other. The first strategy is called *forward recovery*, the second one *backward recovery*.[12] Both generate a shared forest, as in the regular case.

The mechanism is the following. If, at a certain point, the parsing is blocked, we then *jump forward* a certain amount of terminal symbols so as to be able to resume parsing. Formally, in an Earley parser whose input is a DAG, an error is detected when, whatever the *active* table $T[j]$, items of the form $I = [A \rightarrow \alpha.t\beta, i]$ in this table are such that in the DAG there is no out-transition on t from node j. We say that a recovery is possible in k on β if in the suffix $\beta = \beta_1 X \beta_2$ there exists a derived phrase from the symbol X which starts with a terminal symbol r and if there exists a node k in the DAG, $k \geq j$, with an out-transition on r. If it is the case and if this possible recovery is selected, we put the item $[A \rightarrow \alpha t \beta_1 . X \beta_2, i]$ in table $T[k]$. This will ensure

[11]Let us recall here that the Earley algorithm, like the GLR algorithm, has the valid prefix property. This is still true when the input is a DAG.

[12]The combination of these two recovery techniques leads to a more general algorithm than the *skipping* of the GLR* algorithm (Lavie and Tomita, 1993). Indeed, we can not only skip terminals, but in fact replace any invalid prefix by a valid prefix (of a right sentential form) with an increased span. In other words, both terminals and non-terminals may be skipped, inserted or changed, following the heuristics described later on. However, in (Lavie and Tomita, 1993), considering only the skipping of terminal symbols was fully justified since their aim was to parse spontaneous speech, *full of noise and irrelevances that surround the meaningful words of the utterance*.

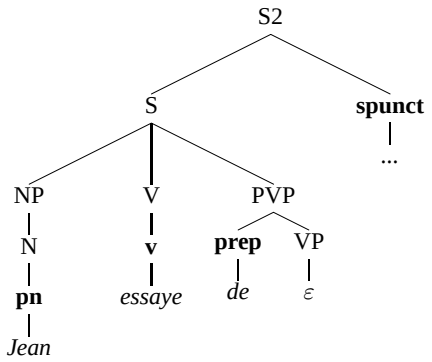


Figure 1: Simplified constituents structure for incomplete sentence *Jean essaye de...* (“*Jean tries to...*”). The derivation of VP in the empty string is the result of a forward recovery, and will lead to an incomplete functional structure (no “pred” in the sub-structure corresponding to node VP).

at least one valid transition from $T[k]$ on r. The effect of such a recovery is to assume that between the nodes j and k in the DAG there is a path that is a phrase generated by $t\beta_1$. We select only nodes k that are as close as possible to j. This *economy principle* allows to skip (without analysis) the smallest possible number of terminal symbols, and leads pretty often to no skipping, thus deriving β_1 into the empty string and producing a recovery in $k = j$. This recovery mechanism allows the parsing process to go forward, hence the name *forward recovery*.

If this strategy fails, we make use of *backward recovery*.[13] Instead of trying to apply the current item, we *jump backward* over terminal symbols that have already been recognized by the current item, until we find its calling items, items on which we try to perform a forward recovery at turn. In case of failure, we can go up recursively until we succeed. Indeed, success is guaranteed, but in the worst case it is obtained only at the axiom. In this extreme case, the shared forest that is produced is only a single production that says that the input DAG is derivable from the axiom. We call this situation *trivial recovery*. Formally, let us come back to the item $I = [A \rightarrow \alpha.t\beta, i]$ of table $T[j]$. We know that there exists in table $T[i]$ an item J of the form $[B \rightarrow \gamma.A\delta, h]$ on which we can hazard a forward recovery in l on δ, where $i \leq j \leq l$. If this fails, we go on coming back further and further in the past, until we reach the initial node of the DAG and the root item $[S' \rightarrow .S\$, 0]$ of table $T[0]$ ($ is the end-of-sentence mark and S' the super-axiom). Since any input ends with an $ mark, this strategy always succeeds, leading in the worst case to trivial recovery.

An example of an analysis produced is shown in Figure 1: in this case, no out-transition on **spunct** is available after having recognized **prep**. Hence a forward recovery is performed that inserts an “empty” VP after the **prep**, so as to build a valid parse.

3.2 Inconsistent or partial f-structures

The computation of f-structures fails if and only if no consistent and complete main f-structure is found. This occurs because unification constraints specified by functional equations could not have been verified or because resulting f-structures are inconsistent or incomplete. Without entering into details, inconsistency mostly occurs because sub-categorization constraints have failed.

$$\left[\begin{array}{l} \text{pred = \textbf{'essayer <subj, de-vcomp>'}, v[2..3]} \\ \\ \text{subj = } \left[\begin{array}{l} \text{pred = \textbf{'Jean <(subj)>'}, pn[1..2]} \\ \text{det = } + \\ \text{hum = } + \\ A_{ij} = \left\{R_9^{182}, R_{26}^{177}, R_{28}^{170}\right\} \end{array}\right]_{F68} \\ \\ \text{de-vcomp = } \left[\begin{array}{l} \text{pred = \textbf{'de <obj|...>'}, prep[3..4]} \\ \text{vcomp = } \left[\begin{array}{l} \text{subj = } []_{F68} \\ A_{ij} = \left\{R_{84}^{162}\right\}_2 \end{array}\right]_{F69} \\ \text{pcase = de} \\ A_{ij} = \{\}_2 \end{array}\right]_{F70} \\ \\ \text{number = sg} \\ \text{person = 3} \\ \text{mode = indicative} \\ \text{tense = present} \\ A_{ij} = \left\{R_{33}^{130}, R_{48}^{119}, R_{49}^{134}\right\} \end{array}\right]$$

Figure 2: Simplified incomplete functional structure for incomplete sentence *Jean essaye de...* (“*Jean tries to...*”). Sub-structure identifiers are indicated as subscripts (like $F70$). In the grammar, a rule can tell the parser to store the current instantiated production in the special field A_{ij} of its associated left-hand side structure. Hence, atoms of the form R_p^q represent instantiated production, thus allowing to link sub-structures to non-terminals of the c-structure.

[13] This second strategy could be also used before or even in parallel with the forward recovery.

A first failure leads to a second evaluation of f-structures on the shared forest, during which consistency and completeness checks are relaxed (an example thereof is given in Figure 2). In case of success, we obtain inconsistent or incomplete main f-structures. Of course, this second attempt can also fail. We then look in the shared forest for a set of *maximal nodes* that have f-structures (possibly incomplete or inconsistent) and whose mother nodes have no f-structures. They correspond to partial disjoint analyses. The disambiguation process presented in section 2.4 applies to all maximal nodes.

3.3 Over-segmentation of unparsable sentences

Despite all these recovery techniques, parsing sometimes fails, and no analysis is produced. This can occur because a time-out given as parameter has expired before the end of the process, or because the Earley parser performed a trivial recovery (because of the insufficient coverage of the grammar, or because the input sentence is simply too far from being correct: grammatical errors, incomplete sentences, too noisy sentences, ...).

For this reason, we developed a layer over SXLFG that performs an *over-segmentation* of ungrammatical sentences. The idea is that it happens frequently that portions of the input sentence are analyzable as sentences, although the full input sentence is not. Therefore, we split in *segments* unparsable sentences (level 1 segmentation); then, if needed, we split anew unparsable level 1 segments[14] (level 2 segmentation), and so on with 5 segmentation levels.[15] Such a technique supposes that the grammar recognizes both chunks (which is linguistically justified, e.g., in order to parse nominal sentences) and isolated terminals (which is linguistically less relevant). In a way, it is a generalization of the use of a FRAGMENT grammar as described in (Riezler et al., 2002; Kaplan et al., 2004).

[14] A sentence can be split into two level 1 segments, the first one being parsable. Then only the second one will be over-segmented anew into level 2 segments. And only unparsable level 2 segments will be over-segmented, and so on.

[15] The last segmentation level segments the input string into isolated terminals, in order to guarantee that any input is parsed, and in particular not to abandon parsing on sentences in which some level 1 or 2 segments are parsable, but in which some parts are only parsable at level 5.

4 Quantitative results

4.1 Grammar, disambiguation rules, lexicon

To evaluate the SXLFG parser, we used our system with a grammar for French that is an adaptation of an LFG grammar originally developed by Clément for his XLFG system (Clément and Kinyon, 2001). In its current state, the grammar has a relatively large coverage. Amongst complex phenomena covered by this grammar are coordinations (without ellipsis), juxtapositions (of sentences or phrases), interrogatives, post-verbal subjects and double subjects (*Pierre dort-il ?*), all kinds of verbal kernels (including clitics, auxiliaries, passive, negation), completives (subcategorized or adjuncts), infinitives (including raising verbs and all three kinds of control verbs), relatives or indirect interrogatives, including when arbitrarily long-distance dependencies are involved. However, comparatives, clefts and elliptical coordinations are not specifically covered, inter alia. Moreover, we have realized that the CF backbone is too ambiguous (see below).

Besides the grammar itself, we developed a set of disambiguation heuristics. Following on this point (Clément and Kinyon, 2001), we use a set of rules that is an adaptation and extension of the three simple principles they describe and that are applied on f-structures, rather than a stochastic model.[16] Our rules are based on linguistic considerations and can filter out functional structures associated to a given node of the forest. This includes two special rules that eliminate inconsistent and incomplete f-structures either in all cases or when consistent and complete structures exist (these rules are not applied during the second pass, if any). As explained above, some non-terminals of the CF backbone, that correspond to linguistically saturated phrases, have been associated with an ordered list of these rules, each of these non-terminal having its own list.[17].

[16] As sketched before, this could be easily done by defining a rule that uses a stochastic model to compute a weight for each f-structure (see e.g., (Miyao and Tsujii, 2002)) and retains only those with the heaviest weights (Riezler et al., 2002; Kaplan et al., 2004). However, our experiments show that structural rules can be discriminative enough to enable efficient parsing, without the need for statistical data that have to be acquired on annotated corpora that are rare and costly, in particular if the considered language is not English.

[17] Our rules, in their order of application on main f-structures, i.e. on the axiom of the backbone, are the following (note that

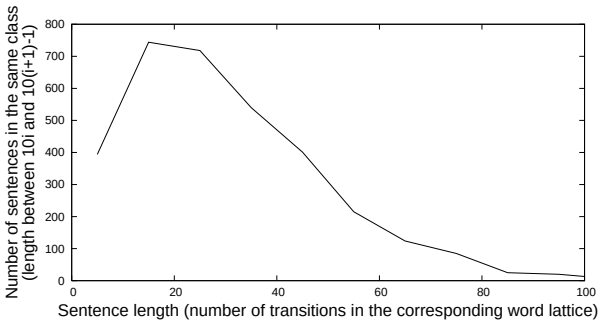


Figure 3: Repartition of sentences of the test corpus w.r.t. their length. We show the cardinal of classes of sentences of length $10i$ to $10(i+1)-1$, plotted with a centered x-coordinate ($10(i+1/2)$).

The lexicon we used is the latest version of Le*fff* (Lexique des formes fléchies du français[18]), which contains morphosyntactic and syntactic information for more than 600,000 entries corresponding to approximately 400,000 different tokens (words or components of multi-word units).

The purpose of this paper is not however to validate the grammar and these disambiguation rules, since the grammar has only the role of enabling evaluation of parsing techniques developed in the current work.

4.2 Results

As for any parser, the evaluation of SXLFG has been carried out by testing it in a real-life situation. We used the previously cited grammar on a raw journalistic corpus of 3293 sentences, not filtered and processed by the SXPipe pre-parsing system described in (Sagot and Boullier, 2005). The repartition of sentences w.r.t. their length is plotted in Figure 3.

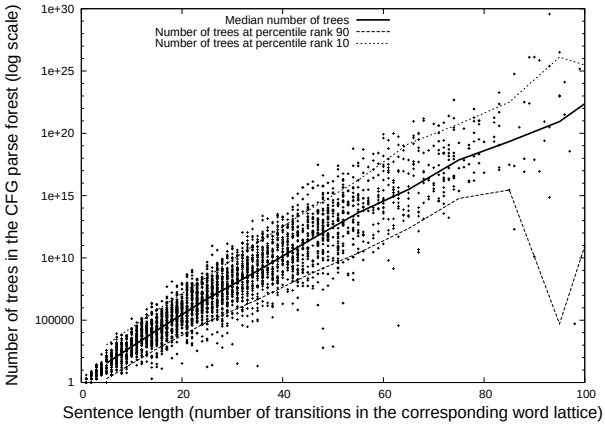


Figure 4: CFG ambiguity (medians are computed on classes of sentences of length $10i$ to $10(i+1)-1$ and plotted with a centered x-coordinate ($10(i+1/2)$).

In all Figures, the x-coordinate is bounded so as to show results only on statistically significant data, although we parse all sentences, the longest one being of length 156.

However, in order to evaluate the performance of our parser, we had to get rid of, as much as possible, the influence of the grammar and the corpus in the quantitative results. Indeed, the performance of the SXLFG parser does not depend on the quality and the ambiguity of the grammar, which is an *input* for SXLFG. On the contrary, our aim is to develop a parser which is as efficient and robust as possible given the input grammar, and in spite of its (possibly huge) ambiguity and of its (possibly poor) intrinsic coverage.

4.2.1 CFG parser evaluation

Therefore, Figure 4 demonstrates the level of ambiguity of the CF backbone by showing the median number of CF parses given the number of transitions in the lattice representing the sentence. Although the number of trees is excessively high, Figure 5 shows the efficiency of our CF parser[19] (the maximum number of trees reached in our corpus is as high as $9.12\,10^{38}$ for a sentence of length 140, which

when used on other non-terminal symbols than the axiom, some rules may not be applied, or in a different order):

Rule 1: *Filter out inconsistent and incomplete structures, if there is at least one consistent and complete structure*.

Rule 2: *Prefer analyses that maximize the sum of the weights of involved lexemes*; amongst lexical entries that have a weight higher than normal are multi-word units.

Rule 3: *Prefer nominal groups with a determiner.*

Rule 4: *Prefer arguments to modifiers, and auxiliary-participle relations to arguments* (the computation is performed recursively on all (sub-)structures).

Rule 5: *Prefer closer arguments* (same remark).

Rule 6: *Prefer deeper structures.*

Rule 7: *Order structures according to the mode of verbs* (we recursively prefer structures with indicative verbs, subjunctive verbs, and so on).

Rule 8: *Order according to the category of adverb governors.*

Rule 9: *Choose one analysis at random* (to guarantee that the output is a unique analysis).

[18] Lexicon of French inflected forms

[19] Our experiments have been performed on a AMD Athlon 2100+ (1.7 GHz).

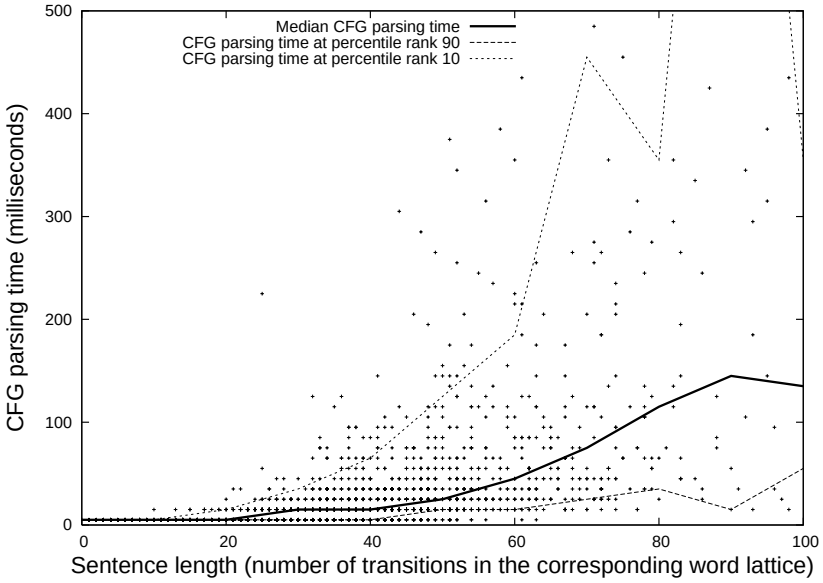


Figure 5: CF parsing time (same remark as for Fig. 4).

is parsed in only 0.75 s). Moreover, the error recovery algorithms described in section 3.1 are successful in most cases where the CF backbone does not recognize the input sentences: out of the 3292 sentences, 364 are not recognized (11.1%), and the parser proposes a non-trivial recovery for all but 13 (96.3%). We shall see later the relevance of the proposed recovered forests. We should however notice that the ambiguity of forests is significantly higher in case of error recovery.

4.2.2 Evaluation of f-structures computation

Although the CF backbone is massively ambiguous, results show that our f-structures evaluation system is pretty efficient. Indeed, with a timeout of 20 seconds, it takes only 6 301 seconds to parse the whole corpus, and only $5,7\%$ of sentences reach the timeout before producing a parse. These results can be compared to the result with the same grammar on the same corpus, but without internal disambiguation (see 2.4), which is 30 490 seconds and 41.2% of sentences reaching the timeout.

The coverage of the grammar on our corpus with internal disambiguation is 57.6%, the coverage being defined as the proportion of sentences for which a consistent and complete main f-structure is output by the parser. This includes cases where the sentence was agrammatical w.r.t. the CF backbone, but for which the forest produced by the error recovery techniques made it possible to compute a consistent and complete main f-structure (this concerns 86 sentences, i.e., 2.6% of all sentences, and 24.5% of all agrammatical sentences w.r.t. the backbone; this shows that CF error recovery gives relevant results).

The comparison with the results with the same grammar but without internal disambiguation is interesting (see Table 1): in this case, the high proportion of sentences that reach the timeout before being parsed leads to a coverage as low as 40.2%. Amid the sentences covered by such a system, 94.6% are also covered by the full-featured parser (with internal disambiguation), which means that only 72 sentences covered by the grammar are lost because of the internal disambiguation. This should be compared with the 645 sentences that are not parsed because of the timeout when internal disambiguation is disabled, but that are covered by the grammar and correctly parsed if internal disambiguation is used: the risk that is taken by pruning f-structures during the parsing process is much smaller than the benefit it gives, both in terms of coverage and parsing time.

Since we do not want the ambiguity of the CF backbone to influence our results, Figure 6 plots the total parsing time, including the evaluation of features structures, against the number of trees produced by the CF parser.

Results	With internal disambiguation		Without internal disambiguation	
Total number of sentences	3293			
Recognized by the backbone	2929		88.9%	
CF parsing with non-trivial recovery	351		10.6%	
CF parsing with trivial recovery	13		0.4%	
Consistent and complete main f-structure	1896	57.6%	1323	40.2%
Inconsistent and incomplete main f-structure	734	22.3%	316	9.6%
Partial f-structures	455	13.8%	278	8.4%
No f-structure	6	0.2%	6	0.2%
No result (trivial recovery)	13	0.4%	13	0.4%
Timeout (20 s)	189	5.7%	1357	40.2%

Table 1: Coverage results with and without internal ranking, with the same grammar and corpus.

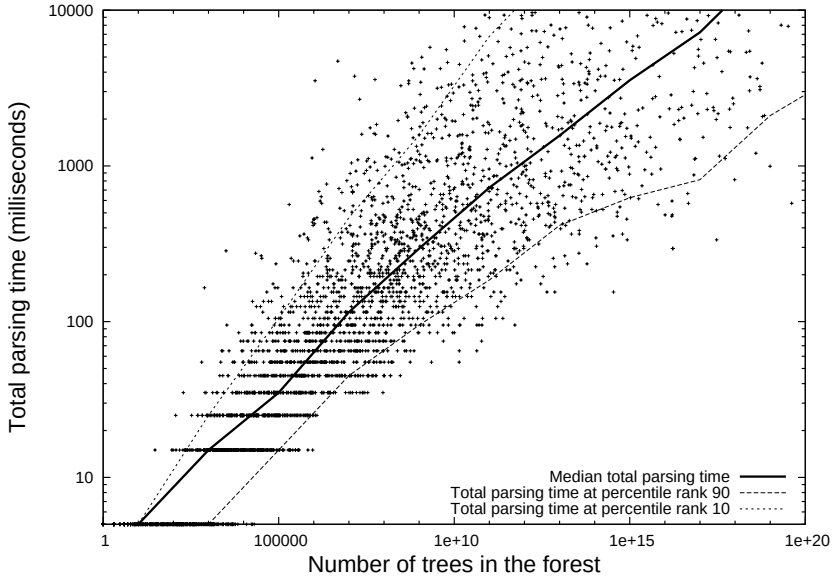


Figure 6: Total parsing time w.r.t. the number of trees in the forest produced by the CF backbone (medians are computed on classes of sentences whose number of trees lies between 10^{2i} and $10^{2i+2} - 1$ and plotted with a centered x-coordinate (10^{2i+1})).

5 Conclusion

This paper shows several important results.

It shows that wide-coverage unification-based grammars can be used to define natural languages and that their parsers can, in practice, analyze raw text.

It shows techniques that allow to compute feature structures efficiently on a massively ambiguous shared forest.

It also shows that error recovery is worth doing both at the phrasal and functional levels. We have shown that a non-negligible portion of input texts that are not in the backbone language can nevertheless, after CF error recovery, be qualified as valid sentences for the functional level.

Moreover, the various robustness techniques that are applied at the functional level allow to gather (partial) useful information. Note that these robust techniques, which do not alter the overall efficiency of SXLFG, apply in the two cases of incomplete grammar (lack of covering) and agrammatical phrases (w.r.t. the current definition), though it seems to be more effective in this latter case.

References

Avery Andrews. 1990. Functional closure in LFG. Technical report, The Australian National University.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03)*, pages 43–54, Nancy, France, April.

Pierre Boullier. 2004. Range concatenation grammars. In *New developments in parsing technology*, pages 269–289. Kluwer Academic Publishers.

Xavier Briffault, Karim Chibout, Gérard Sabah, and Jérôme Vapillon. 1997. An object-oriented linguistic engineering environment using LFG (Lexical-Functional Grammar) and CG (Conceptual Graphs). In *Proceedings of Computational Environments for Grammar Development and Linguistic Engineering, ACL'97 Workshop*.

Lionel Clément and Alexandra Kinyon. 2001. XLFG – an LFG parsing scheme for French. In *Proceedings of LFG'01*, Hong Kong.

Ronald Kaplan and Joan Bresnan. 1982. Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Ronald M. Kaplan and John T. Maxwell. 1994. Grammar writer's workbench, version 2.0. Technical report, Xerox Corporation.

Ronald Kaplan, Stefan Riezler, Tracey King, John Maxwell, Alex Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT/NAACL*, Boston, Massachusetts.

Ronald Kaplan. 1989. The formal architecture of lexical functionnal grammar. *Journal of Informations Science and Engineering*.

Alexandra Kinyon. 2000. Are structural principles useful for automatic disambiguation ? In *Proceedings of in COGSCI'00*, Philadelphia, Pennsylvania, United States.

Alon Lavie and Masaru Tomita. 1993. GLR* – an efficient noise-skipping parsing algorithm for context-free grammars. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 123–134, Tilburg, Netherlands and Durbuy, Belgium.

John Maxwell and Ronald Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of HLT*, San Diego, California.

Stefan Riezler, Tracey King, Ronald Kaplan, Richard Crouch, John Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the Annual Meeting of the ACL*, University of Pennsylvania.

Benoît Sagot and Pierre Boullier. 2005. From raw corpus to word lattices: robust pre-parsing processing. In *Proceedings of L&TC 2005*, Poznań, Pologne.

Parsing Linear Context-Free Rewriting Systems

Håkan Burden
Dept. of Linguistics
Göteborg University
cl1hburd@cling.gu.se

Peter Ljunglöf
Dept. of Computing Science
Göteborg University
peb@cs.chalmers.se

Abstract

We describe four different parsing algorithms for Linear Context-Free Rewriting Systems (Vijay-Shanker et al., 1987). The algorithms are described as deduction systems, and possible optimizations are discussed.

The only parsing algorithms presented for *linear context-free rewriting systems* (LCFRS; Vijay-Shanker et al., 1987) and the equivalent formalism *multiple context-free grammar* (MCFG; Seki et al., 1991) are extensions of the CKY algorithm (Younger, 1967), more designed for their theoretical interest, and not for practical purposes. The reason for this could be that there are not many implementations of these grammar formalisms. However, since a very important subclass of the Grammatical Framework (Ranta, 2004) is equivalent to LCFRS/MCFG (Ljunglöf, 2004a; Ljunglöf, 2004b), there is a need for practical parsing algorithms.

In this paper we describe four different parsing algorithms for Linear Context-Free Rewriting Systems. The algorithms are described as deduction systems, and possible optimizations are discussed.

1 Introductory definitions

A *record* is a structure $\Gamma = \{r_1 = a_1; \ldots; r_n = a_n\}$, where all r_i are distinct. That this can be seen as a set of feature-value pairs. This means that we can define a simple version of *record unification* $\Gamma_1 \sqcup \Gamma_2$ as the union $\Gamma_1 \cup \Gamma_2$, provided that there is no r such that $\Gamma_1.r \neq \Gamma_2.r$.

We sometimes denote a sequence $X_1, \ldots, X_n$ by the more compact $\vec{X}$. To update the ith record in a list of records, we write $\vec{\Gamma}[i := \Gamma]$. To substitute a variable B_k for a record Γ_k in any data structure Γ, we write $\Gamma[B_k/\Gamma_k]$.

1.1 Decorated Context-Free Grammars

The context-free approximation described in section 4 uses a form of CFG with decorated rules of the form $f : A \to \alpha$, where f is the name of the rule, and α is a sequence of terminals and categories subscripted with information needed for post-processing of the context-free parse result. In all other respects a decorated CFG can be seen as a straight-forward CFG.

1.2 Linear Context-Free Rewriting Systems

A *linear context-free rewriting system* (LCFRS; Vijay-Shanker et al., 1987) is a linear, non-erasing *multiple context-free grammar* (MCFG; Seki et al., 1991). An MCFG rule is written[1]

$$A \to f[B_1 \ldots B_\delta] \quad := \quad \{\, r_1 = \alpha_1; \ldots; r_n = \alpha_n \,\}$$

where A and B_i are categories, f is the name of the rule, r_i are record labels and α_i are sequences of terminals and argument projections of the form $B_i.r$. The *language* $\mathcal{L}(A)$ of a category A is a set of string records, and is defined recursively as

$$\begin{aligned} \mathcal{L}(A) \quad = \quad \{ \quad & \Phi[B_1/\Gamma_1, \ldots, B_\delta/\Gamma_\delta] \quad \mid \\ & A \to f[B_1 \ldots B_\delta] := \Phi, \\ & \Gamma_1 \in \mathcal{L}(B_1),\ \ldots,\ \Gamma_\delta \in \mathcal{L}(B_\delta) \quad \} \end{aligned}$$

It is the possibility of discontinuous constituents that makes LCFRS/MCFG more expressive than context-free grammars. If the grammar only consists of single-label records, it generates a context-free language.

Example A small example grammar is shown in figure 1, and generates the language

$$\mathcal{L}(S) \quad = \quad \{\, s\, s_{hm} \mid s \in (a \cup b)^* \,\}$$

where s_{hm} is the homomorphic mapping such that each a in s is translated to c, and each b is translated to d. Examples of generated strings are ac, $abcd$ and $bbaddc$. However, neither abc nor $abcdabcd$ will be

[1] We borrow the idea of equating argument categories and variables from Nakanishi et al. (1997) , but instead of tuples we use the equivalent notion of records for the linearizations.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 11–17,
Vancouver, October 2005.

Figure 1: An example grammar describing the language $\{ s\, s_{hm} \mid s \in (a \cup b)^* \}$

$$
\begin{array}{rcl}
S \rightarrow f[A] & := & \{ s = A.p\ A.q \} \\
A \rightarrow g[A_1\ A_2] & := & \{ p = A_1.p\ A_2.p;\ q = A_1.q\ A_2.q \} \\
A \rightarrow ac[\,] & := & \{ p = a;\ q = c \} \\
A \rightarrow bd[\,] & := & \{ p = b;\ q = d \}
\end{array}
$$

generated. The language is not context-free since it contains a combination of multiple and crossed agreement with duplication.

If there is at most one occurrence of each possible projection $A_i.r$ in a linearization record, the MCFG rule is *linear*. If all rules are linear the grammar is linear. A rule is *erasing* if there are argument projections that have no realization in the linearization. A grammar is erasing if it contains an erasing rule. It is possible to transform an erasing grammar to non-erasing form (Seki et al., 1991).

Example The example grammar is both linear and non-erasing. However, given that grammar, the rule

$$E \rightarrow e[A] \quad := \quad \{ r_1 = A.p;\ r_2 = A.p \}$$

is both non-linear (since $A.p$ occurs more than once) and erasing (since it does not mention $A.q$).

1.3 Ranges

Given an input string w, a *range* ρ is a pair of indices, (i, j) where $0 \leq i \leq j \leq |w|$ (Boullier, 2000). The entire string $w = w_1 \ldots w_n$ spans the range $(0, n)$. The word w_i spans the range $(i - 1, i)$ and the substring $w_{i+1}, \ldots, w_j$ spans the range (i, j). A range with identical indices, (i, i), is called an empty range and spans the empty string.

A record containing label-range pairs,

$$\Gamma \quad = \quad \{ r_1 = \rho_1, \ldots, r_n = \rho_n \}$$

is called a *range record*. Given a range $\rho = (i, j)$, the *ceiling* of ρ returns an empty range for the right index, $\lceil \rho \rceil = (j, j)$; and the *floor* of ρ does the same for the left index $\lfloor \rho \rfloor = (i, i)$. Concatenation of two ranges is non-deterministic,

$$(i, j) \cdot (j', k) \quad = \quad \{ (i, k) \mid j = j' \}$$

.

1.3.1 Range restriction

In order to retrieve the ranges of any substring s in a sentence $w = w_1 \ldots w_n$ we define *range restriction* of s with respect to w as $\langle s \rangle^w = \{ (i, j) \mid s = w_{i+1} \ldots w_j \}$, i.e. the set of all occurrences of s in w. If w is understood from the context we simply write $\langle s \rangle$.

Range restriction of a linearization record Φ is written $\langle \Phi \rangle$, which is a set of records, where every terminal token s is replaced by a range from $\langle s \rangle$. The range restriction of two terminals next to each other fails if range concatenation fails for the resulting ranges. Any unbound variables in Φ are unaffected by range restriction.

Example Given the string $w = abba$, range restricting the terminal a yields

$$\langle a \rangle^w \quad = \quad \{ (0, 1),\ (3, 4) \}$$

Furthermore,

$$\langle a\ A.r\ a\ b\ B.q \rangle^w \quad = \quad \{ (0, 1)\ A.r\ (0, 2)\ B.q,\ (3, 4)\ A.r\ (0, 2)\ B.q \}$$

The other possible solutions fail since they cannot be range concatenated.

2 Parsing as deduction

The idea with *parsing as deduction* (Shieber et al., 1995) is to deduce parse items by inference rules. A parse item is a representation of a piece of information that the parsing algorithm has acquired. An inference rule is written

$$\frac{\gamma_1 \ldots \gamma_n \quad C}{\gamma}$$

where γ is the consequence of the antecedents $\gamma_1 \ldots \gamma_n$, given that the side conditions in C hold.

2.1 Parsing decorated CFG

Decorated CFG can be parsed in a similar way as standard CFG. For our purposes it suffices to say that the algorithm returns items of the form,

$$[f : A/\rho \rightarrow B_1/\rho_1\ \ldots\ B_n/\rho_n \bullet\,]$$

saying that A spans the range ρ, and each daughter B_i spans ρ_i.

The standard inference rule *combine* might look like this for decorated CFG:

Combine

$$\frac{\begin{array}{l}[f : A/\rho \rightarrow \alpha \bullet B_x\ \beta] \\ [g : B/\rho' \rightarrow \ldots \bullet\,] \\ \rho'' \in \rho \cdot \rho'\end{array}}{[f : A/\rho \rightarrow \alpha\ B_x/\rho'' \bullet \beta]}$$

Note that the subscript x in B_x is the decoration that will only be used in post-processing.

3 The Naïve algorithm

Seki et al. (1991) give an algorithm for MCFG, which can be seen as an extension of the CKY algorithm (Younger, 1967). The problem with that algorithm is that it has to find items for all daughters at the same time. We modify this basic algorithm to be able to find one daughter at the time.

There are two kinds of items. A *passive* item $[A;\ \Gamma]$ has the meaning that the category A has been found spanning the range record Γ. An *active* item for the rule $A \to f[\vec{B}\ \vec{B'}] := \Psi$ has the form

$$[A \to f[\vec{B} \bullet \vec{B'}];\ \Phi;\ \vec{\Gamma}]$$

in which the categories to the left of the dot, $\vec{B}$, have been found with the linearizations in the list of range records $\vec{\Gamma}$. Φ is the result of substituting the projections in Ψ with ranges for the categories found in $\vec{B}$.

3.1 Inference rules

There are three inference rules, Predict, Combine and Convert.

Predict

$$\frac{\begin{array}{l} A \to f[\vec{B}] := \Psi \\ \Phi \in \langle \Psi \rangle \end{array}}{[A \to f[\bullet\ \vec{B}];\ \Phi;\ ]}$$

Prediction gives an item for every rule in the grammar, where the range restriction Φ is what has been found from the beginning. The list of daughters is empty since none of the daughters in $\vec{B}$ have been found yet.

Combine

$$\frac{\begin{array}{l} [A \to f[\vec{B} \bullet B_k\ \vec{B'}];\ \Phi;\ \vec{\Gamma}] \\ [B_k;\ \Gamma_k] \\ \quad \Phi' \in \Phi[B_k/\Gamma_k] \end{array}}{[A \to f[\vec{B}\ B_k \bullet \vec{B'}];\ \Phi';\ \vec{\Gamma},\ \Gamma_k]}$$

An active item looking for B_k and a passive item that has found B_k can be combined into a new active item. In the new item we substitute B_k for Γ_k in the linearization record. We also add Γ_k to the new item's list of daughters.

Convert

$$\frac{\begin{array}{l} [A \to f[\vec{B} \bullet\ ];\ \Phi;\ \vec{\Gamma}] \\ \quad \Gamma \equiv \Phi \end{array}}{[A;\ \Gamma]}$$

Every fully instantiated active item is converted into a passive item. Since the linearization record Φ is fully instantiated, it is equivalent to the range record Γ.

Figure 2: The example grammar converted to a decorated CFG

$$\begin{array}{rcl} f : (S.s) & \to & (A.p)\ (A.q) \\ g : (A.p) & \to & (A.p)_1\ (A.p)_2 \\ g : (A.q) & \to & (A.q)_1\ (A.q)_2 \\ ac : (A.p) & \to & a \\ ac : (A.q) & \to & b \\ bd : (A.p) & \to & c \\ bd : (A.q) & \to & d \end{array}$$

The subscripted numbers are for distinguishing the two categories from each other, since they are equivalent. Here $A.q$ is a context-free category of its own, not a record projection.

4 The Approximative algorithm

Parsing is performed in two steps in the approximative algorithm. First we parse the sentence using a context-free approximation. Then the resulting context-free chart is recovered to a LCFRS chart.

The LCFRS is converted by creating a decorated context-free rule for every row in a linearization record. Thus, the rule

$$A \to f[\vec{B}] \quad := \quad \{\, r_1 = \alpha_1;\ \ldots;\ r_n = \alpha_n \,\}$$

will give n context-free rules $f : A.r_i \to \alpha_i$. The example grammar from figure 1 is converted to a decorated CFG in figure 2.

Parsing is now initiated by a context-free parsing algorithm returning decorated items as in section 2.1. Since the categories of the decorated grammar are projections of LCFRS categories, the final items will be of the form

$$[f : (A.r)/\rho \to \ldots (B.r')_x/\rho' \ldots \bullet\ ]$$

Since the decorated CFG is over-generating, the returned parse chart is unsound. We therefore need to retrieve the items from the decorated CFG parse chart and check them against the LCFRS to get the discontinuous constituents and mark them for validity.

The *initial parse items* are of the form,

$$[A \to f[\vec{B}];\ r = \rho;\ \vec{\Gamma}]$$

where $\vec{\Gamma}$ is extracted from a corresponding decorated item $[f : (A.r)/\rho \to \beta]$, by partitioning the daughters in β such that $\Gamma_i = \{\, r = \rho \mid (B.r)_i/\rho \in \beta \,\}$. In other words, Γ_i will consist of all $r = \rho$ such that $B.r$ is subscripted by i in the decorated item.

Example Given $\beta = (A.p)_2/\rho'\ (B.q)_1/\rho''\ (A.q)_2/\rho'''$, we get the two range records $\Gamma_1 = \{q = \rho''\}$ and $\Gamma_2 = \{p = \rho';\ q = \rho'''\}$.

Apart from the initial items, we use three kinds of parse items. From the initial parse items we first build *LCFRS items*, of the form

$$[A \to f[\vec{B}];\ \Gamma \bullet r_i \dots r_n;\ \vec{\Gamma}]$$

where $r_i \dots r_n$ is a list of labels, $\vec{\Gamma}$ is a list of $|\vec{B}|$ range records, and Γ is a range record for the labels $r_1 \dots r_{i-1}$.

In order to recover the chart we use *mark items*

$$[A \to f[\vec{B} \bullet \vec{B'}];\ \Gamma;\ \vec{\Gamma} \bullet \vec{\Gamma'}]$$

The idea is that $\vec{\Gamma}$ has been verified as range records spanning the daughters $\vec{B}$. When all daughters have been verified, a mark item is converted to a *passive item* $[A;\ \Gamma]$.

4.1 Inference rules

There are five inference rules, Pre-Predict, Pre-Combine, Mark-Predict, Mark-Combine and Convert.

Pre-Predict

$$\frac{\begin{array}{l} A \to f[\vec{B}] := \{r_1 = \alpha_1; \dots; r_n = \alpha_n\} \\ \vec{\Gamma}_\delta = \{\,\}, \dots, \{\,\} \end{array}}{[A \to f[\vec{B}];\ \bullet\, r_1 \dots r_n;\ \vec{\Gamma}_\delta]}$$

Every rule $A \to f[\vec{B}]$ is predicted as an LCFRS item. Since the context-free items contain information about $\alpha_1 \dots \alpha_n$, we only need to use the labels $r_1, \dots, r_n$. $\vec{\Gamma}_\delta$ is a list of $|\vec{B}|$ empty range records.

Pre-Combine

$$\frac{\begin{array}{l} [R;\ \Gamma \bullet r\, r_i \dots r_n;\ \vec{\Gamma}] \\ [R;\ r = \rho;\ \vec{\Gamma}'] \\ \vec{\Gamma}'' \in \vec{\Gamma} \sqcup \vec{\Gamma}' \end{array}}{[R;\ \{\Gamma; r = \rho\} \bullet r_i \dots r_n;\ \vec{\Gamma}'']}$$

If there is an initial parse item for the rule R with label r, we can combine it with an LCFRS item looking for r, provided the daughters' range records can be unified.

Mark-Predict

$$\frac{[A \to [\vec{B}];\ \Gamma \bullet\,;\ \vec{\Gamma}]}{[A \to [\,\bullet \vec{B}];\ \Gamma;\ \bullet \vec{\Gamma}]}$$

When all record labels have been found, we can start to check if the items have been derived in a valid way by marking the daughters' range records for correctness.

Mark-Combine

$$\frac{\begin{array}{l} [A \to f[\vec{B} \bullet B_i\, \vec{B'}];\ \Gamma;\ \vec{\Gamma} \bullet \Gamma_i\, \vec{\Gamma'}] \\ [B_i;\ \Gamma_i] \end{array}}{[A \to f[\vec{B}\, B_i \bullet \vec{B'}];\ \Gamma;\ \vec{\Gamma}\, \Gamma_i \bullet \vec{\Gamma'}]}$$

Record Γ_i is correct if there is a correct passive item for category B_i that has found Γ_i.

Convert

$$\frac{[A \to f[\vec{B} \bullet];\ \Gamma;\ \vec{\Gamma} \bullet\,]}{[A;\ \Gamma]}$$

An item that has marked all daughters as correct is converted to a passive item.

5 The Active algorithm

The active algorithm parses without using any context-free approximation. Compared to the Naïve algorithm the dot is used to traverse the linearization record of a rule instead of the categories in the right-hand side.

For this algorithm we use a special kind of range, ρ^ϵ, which denotes simultaneously all empty ranges (i, i). Range restricting the empty string gives $\langle\epsilon\rangle = \rho^\epsilon$. Concatenation is defined as $\rho \cdot \rho^\epsilon = \rho^\epsilon \cdot \rho = \rho$. Both the ceiling and the floor of ρ^ϵ are identities, $\lceil \rho^\epsilon \rceil = \lfloor \rho^\epsilon \rfloor = \rho^\epsilon$.

There are two kinds of items. *Passive items* $[A;\ \Gamma]$ say that we have found category A inside the range record Γ. An *active item* for the rule

$$A \to f[\vec{B}] \quad := \quad \{\Phi;\ r = \alpha\beta;\ \Psi\}$$

is of the form

$$[A \to f[\vec{B}];\ \Gamma,\ r = \rho \bullet \beta,\ \Psi;\ \vec{\Gamma}]$$

where Γ is a range record corresponding to the linearization rows in Φ and α has been recognized spanning ρ. We are still looking for the rest of the row, β, and the remaining linearization rows Ψ. $\vec{\Gamma}$ is a list of range records containing information about the daughters $\vec{B}$.

5.1 Inference rules

There are five inference rules, Predict, Complete, Scan, Combine and Convert.

Predict

$$\frac{\begin{array}{l} A \to f[\vec{B}] := \{r = \alpha; \Phi\} \\ \vec{\Gamma}_\delta = \{\,\}, \dots, \{\,\} \end{array}}{[A \to f[\vec{B}];\ \{\},\ r = \rho^\epsilon \bullet \alpha,\ \Phi;\ \vec{\Gamma}_\delta]}$$

For every rule in the grammar, predict a corresponding item that has found the empty range. $\vec{\Gamma}_\delta$ is a list of $|\vec{B}|$ empty range records since nothing has been found yet.

Complete

$$\frac{[R;\ \Gamma,\ r = \rho \bullet \epsilon,\ \{r' = \alpha; \Phi\};\ \vec{\Gamma}]}{[R;\ \{\Gamma; r = \rho\},\ r' = \rho^\epsilon \bullet \alpha, \Phi;\ \vec{\Gamma}]}$$

When an item has found an entire linearization row we continue with the next row by starting it off with the empty range.

Scan

$$\frac{[R;\,\Gamma,\,r=\rho\bullet s\,\alpha,\,\Phi;\,\vec{\Gamma}]\quad \rho'\in\rho\cdot\langle s\rangle}{[R;\,\Gamma,\,r=\rho'\bullet\alpha,\,\Phi;\,\vec{\Gamma}]}$$

When the next symbol to read is a terminal, its range restriction is concatenated with the range for what the row has found so far.

Combine

$$\frac{[A\to f[\vec{B}];\,\Gamma,\,r=\rho\bullet B_i.r'\,\alpha,\,\Phi;\,\vec{\Gamma}]\quad [B_i;\,\Gamma']\quad \rho'\in\rho\cdot\Gamma'.r'\quad \Gamma_i\subseteq\Gamma'}{[A\to f[\vec{B}];\,\Gamma,\,r=\rho'\bullet\alpha,\,\Phi;\,\vec{\Gamma}[i:=\Gamma']]}$$

If the next thing to find is a projection on B_i, and there is a passive item where B_i is the category, where Γ' is consistent with Γ_i, we can move the dot past the projection. Γ_i is updated with Γ', since it might contain more information about the ith daughter.

Convert

$$\frac{[A\to f[\vec{B}];\,\Gamma,\,r=\rho\bullet\epsilon,\,\{\};\,\vec{\Gamma}]}{[A;\,\{\Gamma;r=\rho\}]}$$

An active item that has fully recognized all its linearization rows is converted to a passive item.

6 The Incremental algorithm

An incremental algorithm reads one token at the time and calculates all possible consequences of the token before the next token is read[2]. The Active algorithm as described above is not incremental, since we do not know in which order the linearization rows of a rule are recognized. To be able to parse incrementally, we have to treat the linearization records as sets of feature-value pairs, instead of a sequence.

The items for a rule $A\to f[\vec{B}]:=\Phi$ have the same form as in the Active algorithm:

$$[A\to f[\vec{B}];\,\Gamma,\,r=\rho\bullet\beta,\,\Psi;\,\vec{\Gamma}]$$

However, the order between the linearization rows does not have to be the same as in Φ. Note that in this algorithm we do not use passive items. Also note that since we always know where in the input we are, we cannot make use of a distinguished ϵ-range. Another consequence of knowing the current input position is that there are fewer possible matches for the Combine rule.

[2]See e.g. the ACL 2004 workshop "Incremental Parsing: Bringing Engineering and Cognition Together".

6.1 Inference rules

There are four inference rules, Predict, Complete, Scan and Combine.

Predict

$$\frac{A\to f[\vec{B}]:=\{\Phi;r=\alpha;\Psi\}\quad 0\le k\le|w|}{[A\to f[\vec{B}];\,\{\},\,r=(k,k)\bullet\alpha,\,\{\Phi;\Psi\};\,\vec{\Gamma}_\delta]}$$

An item is predicted for every linearization row r and every input position k. $\vec{\Gamma}_\delta$ is a list of $|\vec{B}|$ empty range records.

Complete

$$\frac{[R;\,\Gamma,\,r=\rho\bullet\epsilon,\,\{\Phi;r'=\alpha;\Psi\};\,\vec{\Gamma}]\quad \lceil\rho\rceil\le k\le|w|}{[R;\,\{\Gamma;r=\rho\},\,r'=(k,k)\bullet\alpha,\,\{\Phi;\Psi\};\,\vec{\Gamma}]}$$

Whenever a linearization row r is fully traversed, we predict an item for every remaining linearization row r' and every remaining input position k.

Scan

$$\frac{[R;\,\Gamma,\,r=\rho\bullet s\,\alpha,\,\Phi;\,\vec{\Gamma}]\quad \rho'\in\rho\cdot\langle s\rangle}{[R;\,\Gamma,\,r=\rho'\bullet\alpha,\,\Phi;\,\vec{\Gamma}]}$$

If the next symbol in the linearization row is a terminal, its range restriction is concatenated to the range for the partially recognized row.

Combine

$$\frac{[R;\,\Gamma,\,r=\rho\bullet B_i.r'\,\alpha,\,\Phi;\,\vec{\Gamma}]\quad [B_i\to\ldots;\,\Gamma',\,r'=\rho'\bullet\epsilon,\,\ldots;\,\ldots]\quad \rho''\in\rho\cdot\rho'\quad \Gamma_i\subseteq\{\Gamma';r'=\rho'\}}{[R;\,\Gamma,\,r=\rho''\bullet\alpha,\,\Phi;\,\vec{\Gamma}[i:=\{\Gamma';r'=\rho'\}]]}$$

If the next item is a record projection $B_i.r'$, and there is an item for B_i which has found r', then move the dot forward. The information in Γ_i must be consistent with the information found for the B_i item, $\{\Gamma';r'=\rho'\}$.

7 Discussion

We have presented four different parsing algorithms for LCFRS/MCFG. The algorithms are described as deduction systems, and in this final section we discuss some possible optimizations, and complexity issues.

7.1 Different prediction strategies

The Predict rule in the above described algorithms is very crude, predicting an item for each rule in the grammar (for the Incremental algorithm even for each input position). A similar context-free prediction rule is called *bottom-up Earley* by Sikkel and Nijholt (1997). Such crude predictions are only intended for educational purposes, since they lead to lots of uninteresting items, and waste of computing power. For practical purposes there are two standard context-free prediction strategies, top-down and bottom-up (see e.g. Wirén (1992)) and they can be adapted to the algorithms presented in this paper.

The main idea is that an item for the rule $A \to f[\vec{B}]$ with the linearization row $r = \alpha$ is only predicted if...

(Top-down prediction) ...there is another item looking for $A.r$.

(Bottom-up prediction) ...there is an passive item that has found the first symbol in α.

For a more detailed description of these prediction strategies, see Ljunglöf (2004a).

7.2 Efficiency and complexity of the algorithms

The theoretical time complexity for these algorithms is not better than what has been presented earlier.[3] The complexity arguments are similar and the reader is referred to Seki et al. (1991).

However, theoretical time complexity does not say much about practical performance, as is already clear from context-free parsing, where the theoretical time complexity has remained the same ever since the first publications (Kasami, 1965; Younger, 1967). There are two main ways of improving the efficiency of existing algorithms, which can be called *refinement* and *filtering* (Sikkel and Nijholt, 1997). First, one wants to be able to locate existing parse items efficiently, e.g. by indexing some properties in a hash table. This is often done by *refining* the parse items or inference rules, increasing the number of items or deduction steps. Second, it is desirable to reduce the number of parse items, which can be done by *filtering* out redundant parts of an algorithm.

The algorithms presented in this paper can all be seen as refinements and filterings of the basic algorithm of Seki et al. (1991):

The naïve algorithm is a refinement of the basic algorithm, since single items and deduction steps are decomposed into several different items and smaller deduction steps.

The approximative algorithm is both a refinement and a filtering of the naïve algorithm; a refinement since the inference rules Pre-Predict and Pre-Combine are added, and a filtering since there will hopefully be less items for Mark-Predict and Mark-Combine to take care of.

The active algorithm is a refinement of the naïve algorithm, since the Combine rule is divided into the rules Complete, Scan and Combine.

The incremental algorithm is finally a refinement of the active algorithm, since Predict and Complete can select from any possible remaining linearization row, and not just the following.

Furthermore, the different prediction strategies (top-down and bottom-up), become filterings of the algorithms, since they reduce the number of parse items.

7.3 Implementing and testing the algorithms

The algorithms presented in this paper have been implemented in the programming language Haskell, for inclusion in the Grammatical Framework system (Ranta, 2004). These implementations are described by Burden (2005). We have also started to implement a selection of the algorithms in the programming language Prolog.

Preliminary results suggest that the Active algorithm with bottom-up prediction is a good candidate for parsing grammars written in the Grammatical Framework. For a normal sentence in the English resource grammar the speedup is about 20 times when compared to context-free parsing and filtering of the parse trees. In the future we plan to test the different algorithms more extensively.

Acknowledgments

The authors are supported by the EU project TALK (Talk and Look, Tools for Ambient Linguistic Knowledge), IST-507802.

References

Pierre Boullier. 2000. Range concatenation grammars. In *6th International Workshop on Parsing Technologies*, pages 53–64, Trento, Italy.

Håkan Burden. 2005. Implementations of parsing algorithms for linear multiple context-free grammars. Master's thesis, Göteborg University, Gothenburg, Sweden.

Tadao Kasami. 1965. An efficient recognition and syntax algorithm for context-free languages. Technical Report AFCLR-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

[3] Nakanishi et al. (1997) reduce the parsing problem to boolean matrix multiplication, but this can be considered a purely theoretical result.

Peter Ljunglöf. 2004a. *Expressivity and Complexity of the Grammatical Framework*. Ph.D. thesis, Göteborg University and Chalmers University of Technology, Gothenburg, Sweden.

Peter Ljunglöf. 2004b. Grammatical Framework and multiple context-free grammars. In *9th Conference on Formal Grammar*, Nancy, France.

Ryuichi Nakanishi, Keita Takada, and Hiroyuki Seki. 1997. An efficient recognition algorithm for multiple context-free languages. In *MOL5: 5th Meeting on the Mathematics of Language*, pages 119–123, Saarbrücken, Germany.

Aarne Ranta. 2004. Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Hiroyuki Seki, Takashi Matsumara, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Klaas Sikkel and Anton Nijholt. 1997. Parsing of context-free languages. In G. Rozenberg and A. Salomaa, editors, *The Handbook of Formal Languages*, volume II, pages 61–100. Springer-Verlag, Berlin.

K. Vijay-Shanker, David Weir, and Aravind Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Meeting of the Association for Computational Linguistics*.

Mats Wirén. 1992. *Studies in Incremental Natural-Language Analysis*. Ph.D. thesis, Linköping University, Linköping, Sweden.

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

Switch Graphs for Parsing Type Logical Grammars*

Bob Carpenter
Alias-i
181 North 11th Street, #401
Brooklyn, NY, 11211
carp@colloquial.com

Glyn Morrill
Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
E-08034 Barcelona
morrill@lsi.upc.edu

Abstract

Parsing in type logical grammars amounts to theorem proving in a substructural logic. This paper takes the proof net presentation of Lambek's associative calculus as a case study. It introduces switch graphs for online maintenance of the Danos-Regnier acyclicity condition on proof nets. Early detection of Danos-Regnier acyclicity violations supports early failure in shift-reduce parsers. Normalized switch graphs represent the combinatorial potential of a set of analyses derived from lexical and structural ambiguities. Packing these subanalyses and memoizing the results leads directly to a dynamic programming algorithm for Lambek grammars.

1 Introduction

Following Montague (1970), we take the goal of a theory of grammar to be that of assigning semantic terms to linguistic expressions. Type logical grammar is a paradigm for developing grammatical theories based on a strong notion of typing for natural language expressions. Specifically, each linguistic expression is assigned a syntactic type and a semantic term. For instance, the expression "John read the book" of English might be assigned a syntactic type S and the semantic term **read**(**the**(**book**))(**j**), the expression "book that John read" the term **that**(λx.**read**(x)(**j**))(**book**) and type CN, and "person that read the book" the type CN and term **that**(λy.**read**(**the**(**book**))(y))(**person**).

*Supported by CICYT project TIC2002–04019–C03–01.

2 Lambek's Associative Calculus

Lambek's associative calculus **L** (Lambek 1958) contains three connectives: concatenation, left division, and right division. Logically, concatenation is conjunction and the divisions are directed implications. Algebraically, concatenation is a free semigroup product and the divisions its left and right residuals. Viewed as a purely syntactic formalism, **L** assigns syntactic types to linguistic expressions modeled as sequences of tokens. From a stipulated lexical assignment of expressions to syntactic types, further assignments of expressions to types are derived through purely logical inference, with the logic representing a sound and complete axiomatization and inference system over the algebraic structure (Pentus 1995).

L appears near the bottom of a hierarchy of substructural logics obtained by dropping structural rules: Lambek proofs are valid as multiplicative intuitionistic linear proofs (restoring permutation) which are valid as conjuntive and implicative relevance proofs (restoring contraction) which are valid as conjuntive and implicative intuitionistic proofs (restoring weakening). In type logical grammars, lexical entries are associated with syntactic types and intuitionistic (in fact probably relevant) proofs as semantic representations, notated as terms of the simply typed λ-calculus with product, under the Curry-Howard correspondence. The semantics of a

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 18–29,
Vancouver, October 2005.

derived expression is the result of substituting the lexical semantics into the reading of the derivation as an intuitionistic proof.

2.1 Syntactic and Semantic Types

The set of *syntactic types* is defined recursively on the basis of a set SynAtom of *atomic syntactic types*. The full set SynTyp of syntactic types is the least set containing the atomic syntactic types SynAtom and closed under the formation of products (SynTyp$\cdot$SynTyp), left divisions (SynTyp$\backslash$SynTyp), and right divisions (SynTyp/SynTyp). The two division, or "slash", types, A/B, read *A over B*, and $B\backslash A$, read *B under A*, refine the semantic function types by providing a directionality of the argument with respect to the function. A linguistic expression assigned to type A/B combines with an expression of type B on its right side to produce an expression of type A. An expression of type $B\backslash A$ combines with an expression of syntactic type B on its left to produce an expression of type A. The product syntactic type $A\cdot B$ is assigned to the concatenation of an expression of type A to an expression of type B. The distinguishing feature of Lambek calculus with respect to the earlier categorial grammar of Bar-Hillel is that as well as the familar cancelation (modus ponens) rules, it admits also a form of the deduction theorem: if the result of concatenating an expression e to each B results in an expression of type A, then it follows that e is assigned to syntactic type A/B.

Semantic representations in Lambek type logical grammar are simply typed λ-terms with product. We assume a set SemAtom of *atomic semantic types*, which generate the usual *function types* $\sigma \to \tau$ and *product types* $\sigma \times \tau$. Terms are grounded on an infinite set of distinct *variables* Var_σ, along with a set of distinct *contants* Con_σ for each type σ. We assume the usual *λ-terms* consisting of variables, constants, *function applications* $\alpha(\beta)$, *function abstractions* $\lambda x.\alpha$, *pairs* $\langle\alpha,\beta\rangle$ and *projections* from pairs $\pi_1\delta$ and $\pi_2\delta$ onto the first and second element of the pair respectively. We say that a term α is *closed* if and only if it contains no free variables.

A *type map* consists of a mapping $\mathsf{typ} : \mathsf{SynAtom} \to \mathsf{SemTyp}$. That is, each atomic syntactic type $A \in \mathsf{AtomCat}$ is assigned to a (not necessarily atomic) semantic type $\mathsf{typ}(A) \in \mathsf{SemTyp}$. Semantic types are assigned to complex syntactic types as follows:

$$\mathsf{typ}(A \cdot B) = \mathsf{typ}(A) \times \mathsf{typ}(B) \quad \text{[Product]}$$

$$\mathsf{typ}(A/B) = \mathsf{typ}(B) \to \mathsf{typ}(A) \quad \text{[Right Division]}$$

$$\mathsf{typ}(B\backslash A) = \mathsf{typ}(B) \to \mathsf{typ}(A) \quad \text{[Left Division]}$$

We will often write $\alpha : A$ where α is a λ-term of type $\mathsf{typ}(A)$.

2.2 Linguistic Expressions and the Lexicon

In the Lambek calculus, linguistic expressions are modeled by sequences of atomic symbols. These atomic symbols are drawn from a finite set Tok of *tokens*. The full set of linguistic *expressions* Tok^* is the set of sequences of tokens. For the sake of this short version of the paper we admit the empty sequence; we will address its exclusion (as in the original definition of **L**) in a longer version.

The compositional assignment of semantic terms to linguistic expressions is grounded by a finite set of assignments of terms and types to expressions. A *lexicon* is a finite relation $\mathsf{Lex} \subseteq \mathsf{Tok}^* \times \mathsf{Term} \times \mathsf{SynTyp}$, where all $\langle w, \alpha, A\rangle \in \mathsf{Lex}$ are such that the semantic term α is of the appropriate type for the syntactic type A. We assume that the only terms used in the lexicon are *relevant*, in the sense of relevance logic, in not containing vacuous abstractions. Note that the set of atomic semantic types, atomic syntactic types and the semantic type mapping are assumed as part of the definition of a lexicon. Type logical grammar is an example of a fully lexicalized grammar formalism in that the lexicon is the only locus of language-specific information.

2.3 Proof Nets

A *sequent* $\Gamma \Rightarrow \alpha : A$ is formed from an *antecedent* Γ consisting of a (possibly empty) sequence of λ-term and syntactic type pairs, and a *consequent* pair $\alpha : A$, where the terms are of the appropritate type for the types. Following Roorda (1991), we define theoremhood with Girard-style proof nets (Girard 1987), a geometric alternative to Lambek's Gentzen-style calculus (Lambek 1958).

Proof nets form a graph over nodes labeled by polar types, where a *polar type* is the combination of a syntactic type and one of two *polarities*, *input* (negative) and *output* (positive). We write $A^\bullet$ for the *input polar type*, which corresponds to antecedent types and is thus logicaly negative. We write A° for

the *output polar type*, which is logically positive and corresponds to a consequent type. A *literal* is a polar type with an atomic syntactic type. Where A is an atomic syntactic type, the literals $A^\bullet$ and A° are said to be *complementary*.

Each polar type defines an ordered binary tree rooted at that polar type, known as a *polar tree*. For a literal, the polar tree is a single node labeled by that literal. For polar types with complex syntactic types, the polar tree is rooted at the polar type and unfolded upwards based on connective and polarity according to the solid lines in Figure 1, which includes also other annotation. Examples for some linguistically motivated types are shown in Figure 2.

The solid edges of the graphs are the edges of the *logical links*. Each unfolding is labeled with a multiplicative linear logic connective, either *multiplicative conjunction* ($\otimes$) or *multiplicative disjunction* ($\wp$). This derives from the logical interpretation of the polar type trees as formula trees in multiplicative linear logic. Unfolding the Lambek connectives to their linear counterparts, $(A/B)^\bullet$ and $(B\backslash A)^\bullet$ unfold to $A^\bullet \wp B^\circ$; $(A/B)^\circ$ and $(B\backslash A)^\circ$ unfold to $A^\circ \otimes B^\bullet$; $(A \cdot B)^\bullet$ unfolds to $A^\bullet \otimes B^\bullet$; and $(A \cdot B)^\circ$ unfolds to $A^\circ \wp B^\circ$. The type unfoldings correspond to the classical equivalences between $(\phi \rightarrow \psi)$ and $(\neg\phi \vee \psi)$, between $\neg(\phi \rightarrow \psi)$ and $(\phi \wedge \neg\psi)$, and between $\neg(\phi \wedge \psi)$ and $(\neg\phi \vee \neg\phi)$. For atomic syntactic types A, $A^\bullet$ becomes simply A, whereas A° becomes its linear *negation* $A^\perp$; this is the sense in which polar atomic types correspond to logical literals. The non-commutatitive nature of the Lambek calculus is reflected in the ordering of the subtrees in the unfoldings; for commutative logics, the proof trees are not ordered.

The *proof frame* for a syntactic sequent $C_1, \ldots, C_n \Rightarrow C_0$ is the ordered sequence of polar trees rooted at $C_0^\circ, C_1^\bullet, \ldots, C_n^\bullet$. We convert sequents to frames in this order, with the output polar tree first. In general, what follows applies to any cyclic reordering of these polar trees. Note that the antecedent types $C_1, \ldots C_n$ have input (negative) polarity inputs and the consequent type C_0 has output (positive) polarity. All of our proof frames are *intuitionistic* in that they have a single output conclusion, i.e. a unique polar tree rooted at an output type.

A *partial proof structure* consists of a proof frame with a set of *axiom links* linking pairs of complementary literals with at most one link per literal. Axiom links in both directions are shown in Figure 3. A *proof structure* is a proof structure in which all literals are connected to complementary literals by axiom links.

Proof nets are proof structures meeting certain conditions. A proof structure is *planar* if and only if its axiom links can be drawn in the half-plane without crossing lines; this condition enforces the lack of commutativity of the Lambek calculus. The final condition on proof structures involves switching. A *switching* of a proof structure is a subgraph that arises from the proof structure by removing exactly one edge from each disjunctive ($\wp$) link. A proof structure is said to be *Danos-Regnier (DR-) acyclic* if and only if each of its switchings is acyclic (Danos and Regnier 1989).[1] A *proof net* is a planar DR-acyclic proof structure. A *theorem* is any sequent forming the proof frame of a proof net.

Consider the three proof nets in Figure 4. The first example has no logical links, and corresponds to the simplest sequent derivation $S \Rightarrow S$. The second example represents a determiner, noun and intransitive verb sequence. Both of these examples are acyclic, as must be every proof net with no logical $\wp$-links. The third example corresponds to the type-raising sequent $N \Rightarrow S/(N\backslash S)$. Unlike the other examples, this proof net involves a $\wp$-link and is cyclic. But both of its switchings are acyclic, so it satisfies the Danos-Regnier acyclicity condition.

2.4 Essential Nets and Semantic Trips

A term is said to be *pure* if and only if it contains no constants. The *linear terms* are closed, pure λ-terms that bind each variable exactly once. Each proof net in the Lambek calculus corresponds to a linear (i.e. binding each variable exactly once) λ-term via the Curry-Howard correspondence. This term abstracts over variables standing in for the semantics of the inputs in the antecedent of the sequent and has a body that is determined by the consequent of the sequent. For instance, the λ-term $\lambda x.\lambda P.P(x)$ corresponds to the syntactic sequent $x : N$, P :

[1] The full Danos-Regnier condition is that every switching be acyclic and connected. Fadda and Morrill (2005) show that for the intuitionistic case (i.e. single output conclusion, as for **L**), DR-acyclicity entails the connectedness of every switching.

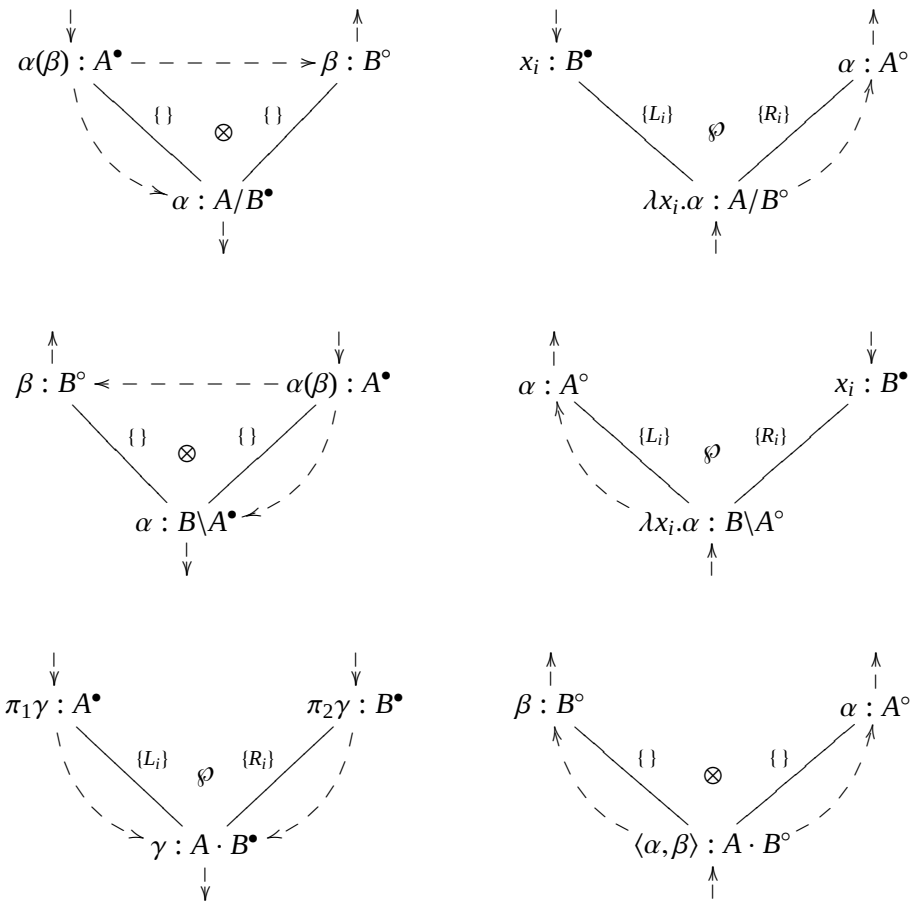


Figure 1: Logical Links with Switch Paths (solid) and Semantic Trip (dashed)

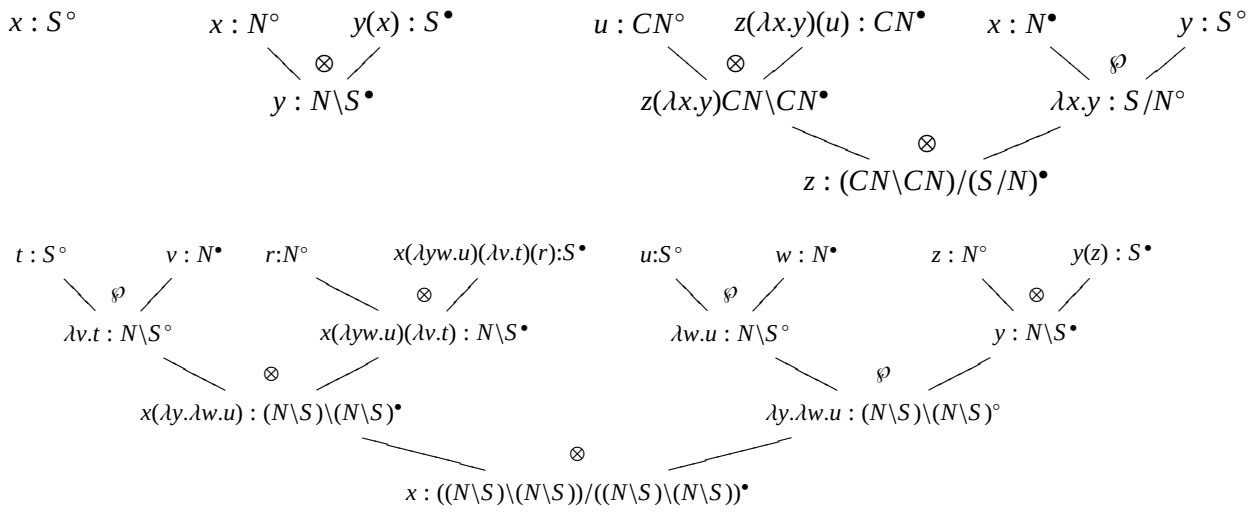


Figure 2: Examples of Polar Type Trees

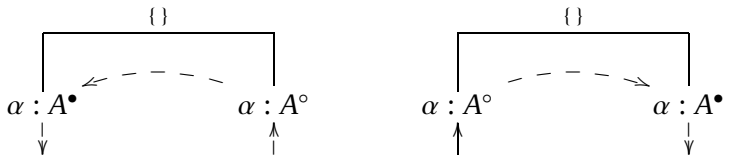


Figure 3: Axiom Links with Switch Paths and Semantic Trip

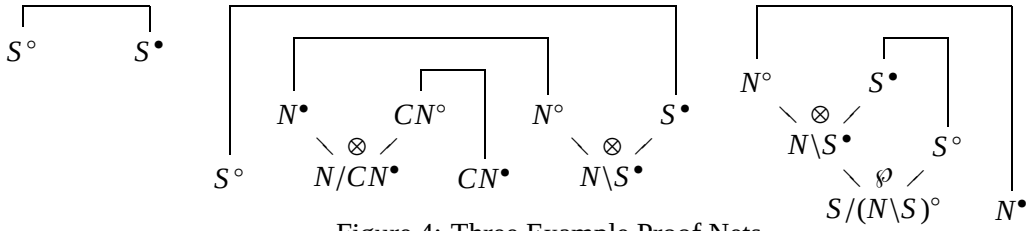


Figure 4: Three Example Proof Nets

$N\backslash S \Rightarrow P(x) : S$ and $\lambda x.\lambda P.P(x)$ corresponds to the sequent $x : N \Rightarrow \lambda P.P(x) : S/(N\backslash S)$. The λ-term induced by the Curry-Howard correspondence can be determined by a unification problem over a proof net (Roorda 1991). Different proof nets for the same theorem correspond to different interpretations through the Curry-Howard correspondence. The *essential net* of a proof structure is the directed graph rooted at the root node of the output polar type tree whose edges are shown as dashed lines in Figures 1 and 3 (LaMarche 1994). Each output division type introduces a fresh variable on its input subtype (its argument), as indicated by the labels x_i in Figure 1. The essential nets for the examples in Figure 4 are shown in Figure 5.

Terms are computed for the polar type trees by assigning terms to the roots of the polar inputs. The tree is then unfolded unifying in substitutions as it goes, as illustrated in the example polar type trees in Figure 2. The direction of axiom links in the essential net provide the substitutions necessary to solve the unification problem of λ-terms in the proof net established by equating the two halves of each axiom linked complementary pair of literals. A traversal of an essential net carrying out the substitutions specified by axiom links constitutes a *semantic trip* the end result of which is the Curry-Howard λ-term for the Lambek calculus theorem derived by the proof net. All λ-terms derived from a semantic trip with variables or constants assigned to input root polar types will be in β-η long form. The essential net directly corresponds to the tree of the semantic term derived by the Curry-Howard correspondence.

The well-formedness of a set of axiom linkings over a polar tree may be expressed in terms of the essential net. Among the conditions are that an essential net must be acyclic and planar. In addition, essential nets must be connected in two ways. First, there must be a path from the root of the single output polar tree to the root of each of the input polar trees. Second, there must be a path from each output daughter of an output division to the input daughter. That is, when A/B° is unfolded to $B^\bullet A^\circ$, there must be a path from A° to $B^\bullet$. These conditions express the definition of linear semantic terms dictated through the logic by the Curry-Howard correspondence. The first condition requires each variable (or term) corresponding to the root of an input polar tree to occur in the output term, whereas the second condition requires that variables only occur within their proper scopes so that they are bound. The essential nets presented in Figure 5 adhere to these conditions and produce well-typed linear λ-terms. The example presented in Figure 6 shows a set of axiom links that does not form a proof net; it violates the condition on variable binding, as is seen from the lack of path from the N° daughter to the $N^\bullet$ daughter of the N/N° node. The major drawback to using these conditions directly in parsing is that they are existential in the sense of requring the existence of a certain kind of path, and thus difficult to refute online during parsing. In comparison, the Danos-

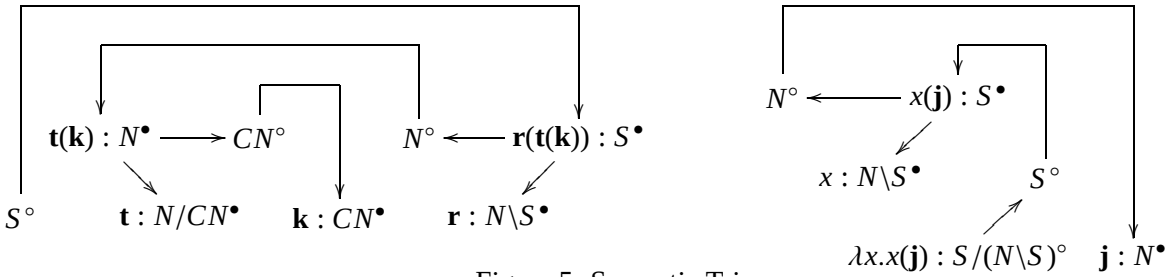


Figure 5: Semantic Trips

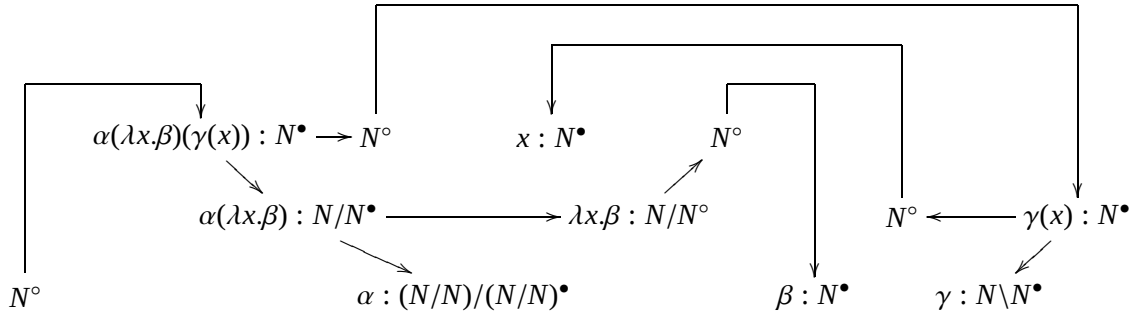


Figure 6: Unbound Variable in Illegal Semantic Trip

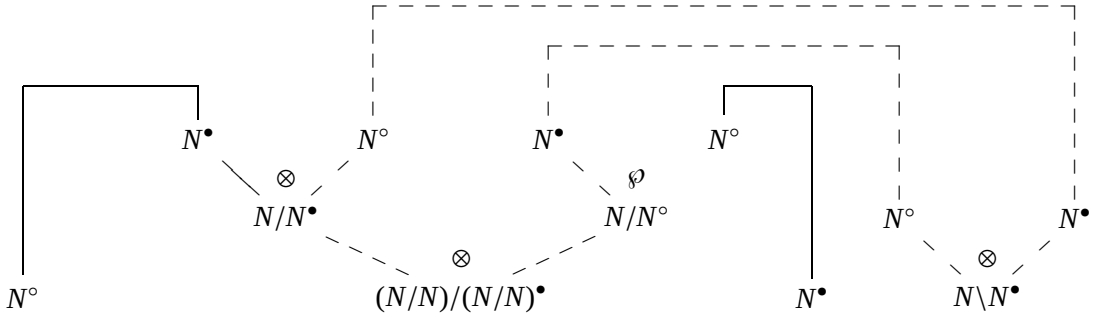


Figure 7: Switching with Path Violating Danos-Regnier Acyclicity

Regnier acyclicity condition is violated by the attempt to close off the binding of the variable. The path vilolating DR acyclicity is shown in Figure 7, with the path given in dashed lines and the switching taking the right daughter of N/N° as the arc to remove.

3 Parsing with Switch Graphs

The planar connection of all literals into a proof structure is straightforward to implement. Axiom links are simply added in such a way that planarity is maintained until a complete linkage is found. In our shift-reduce-style parser, planarity is maintained by a stack in the usual way (Morrill 2000). For dynamic programming, we combine switch graphs in the cells in a Cocke-Kasami-Younger (CKY) parser (Morrill 1996). The main challenge is enforcing DR-acyclicity, and this is the main focus of the rest of the paper. We introduce switch graphs, which not only maintain DR-acyclicity, but also lead the way to a normal form for well-formed subsequence fragments of a partial proof structure. This normal form underlies the packing of ambiguities in subderivations in exactly the same way as usual in dynamic programming parsing.

3.1 Switch Graphs

Switch graphs are based on the observation that a proof structure is DR-acyclic if and only if every cycle contains both edges of a $\wp$-link. If a cycle contains both edges of a $\wp$-link, then any switching removes the cycle. Thus if every cycle in a proof structure contains both edges of a $\wp$-link, every switching is acyclic.

The *(initial) switch graph* of a partial proof structure is defined as the undirected graph underlying the partial proof structure with edges labeled with sets of $\wp$-edge identifiers as indicated in Figures 1 and 3. Each edge in a logical $\wp$-link is labeled with the singleton set containing an identifier of the link itself, either L_i for the left link of $\wp$-link i or R_i for the right link of $\wp$-link i. Edges of axiom links and logical $\otimes$-links are labeled with the empty set.

The *closure* of a switch graph is computed by iterating the following operation: if there is an edge $n_1 - n_2$ labeled with set X_1 and an edge edge $n_2 - n_3$ labeled with set X_2 such that $X_1 \cup X_2$ does not contain both edges of a $\wp$-link, add an edge $n_1 - n_3$ labeled with $X_1 \cup X_2$. An edge $n-m$ labeled by X is *subsumed* by an edge between the same nodes $n-m$ labeled by Y if $Y \subseteq X$. The *normal switch graph* of a partial proof structure is derived by closing its the initial switch graph, removing edges that are subsumed by other edges, and restricting to the literal nodes not connected by an axiom link. These normal switch graphs define a unique representation of the combinatorial possibilities of a span of polar trees and their associated links in a partial proof structure. That is, any partial proof structure substructure that leads to the same normal switch graph may be substituted in any proof net while maintaining well-formedness.

The fundamental insight explored in this paper is that two literals may be connected by an axiom link in a partial proof structure without violating DR-acyclicity if and only if they are not connected in the normal switch graph for the partial proof structure. The normal switch graph arising from the addition of an axiom link is easily computed. It is just the closure generated by adding the new axiom link, with the two literals being linked removed.

3.2 Shift-Reduce Parsing

In this section, we present the search space for a shift-reduce-style parsing algorithm based on switch graphs. The states in the search consist of a *global stack* of literals, a *lexical stack* of literals, the remaining tokens to be processed, and the set of links among nodes on the stacks in the switch graph. The shift-reduce *search space* is characterized by an initial state and state *transitions*. These are shown in schematic form in Figure 8. The *initial state* contains the output type's literals and switch graph. A *lexical* transition is from a state with an empty lexical stack to one containing the lexical literals of the next token; the lexical entry's switch graph merges with the current one. A *shift* transition pops a literal from the lexical stack and pushes it onto the global stack. A *reduce* transition adds an axiom link between the top of the global stack and lexical stack if they are complementary and are not connected in the switch graph; the resulting switch graph results from adding the axiom link and normalizing. The stack discipline insures that all partial proof structures considered are planar.

Figure 10 displays as rows the shift-reduce search

Stack	Lex	Sw-Gr	Op
A°		$\mathsf{gr}(A^\circ)$	start(A)
S		**G**	
S	$A^\bullet$	$\mathbf{G} \oplus \mathsf{gr}(A^\bullet)$	lex(w, A)

Stack	Lex	Sw-Gr	Op
A_i**S**	$\overline{A}_j$**L**	**G**	
S	**L**	$(\mathbf{G} \oplus i{=}j) - \{i, j\}$	reduce(i, j)
A**S**	B**L**	**G**	
BA**S**	**L**	**G**	shift(B)

Figure 8: Shift-Reduce Parsing Schematic

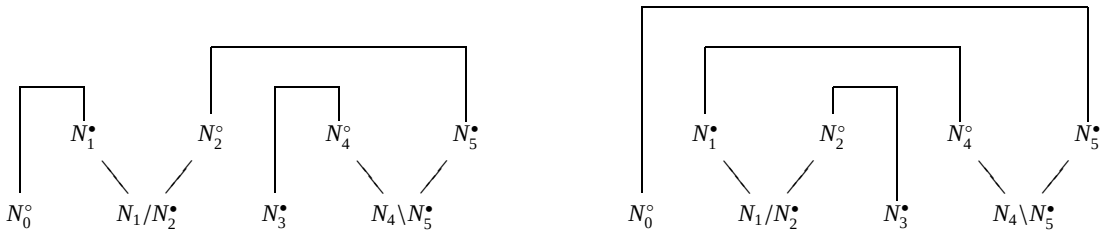


Figure 9: Modifier Attachment Ambiguity Proof Nets

Stack	Lex	Tok	Sw-Gr	Ax	Op
N_0°					start
N_0°	$N_1^\bullet N_2^\circ$	w_1	1-2{}		lex
–	N_2°			0=1	reduce
N_2°					shift
N_2°	$N_3^\bullet$	w_2			lex
$N_3^\bullet N_2^\circ$					shift
$N_3^\bullet N_2^\circ$	$N_4^\circ N_5^\bullet$	w_3	4-5{}		lex
N_2°	$N_5^\bullet$			3=4	reduce
				2=5	reduce

Stack	Lex	Tok	Sw-Gr	Ax	Op
N_0°					start
N_0°	$N_1^\bullet N_2^\circ$	w_1	1-2{}		lex
$N_1^\bullet N_0^\circ$	N_2°		1-2{}		shift
$N_2^\circ N_1^\bullet N_0^\circ$			1-2{}		shift
$N_2^\circ N_1^\bullet N_0^\circ$	$N_3^\bullet$	w_2	1-2{}		lex
$N_1^\bullet N_0^\circ$				2=3	reduce
$N_1^\bullet N_0^\circ$	$N_4^\circ N_5^\bullet$	w_3	4-5{}		lex
N_0°	$N_5^\bullet$			1=4	reduce
				0=5	reduce

Figure 10: Modifier Attachment Ambiguity Shift-Reduce Search States

states corresponding to the two valid proof nets shown in Figure 9. The subscripts on syntactic types in the diagram is only so that they can be indexed in the rows of the table describing the search states. The initial state in both searches is created from the output type's literal. The third column of the diagrams indicate the token consumed at each lexical entry. The switch graphs are shown for the rows for which they're active. Because there are no $\wp$-links, all sets of edges are empty. The fifth column shows the axiom linking made at each reduce step. The history of these decisions and lexical insertion choices determines the final proof net. Finally, the sixth column shows the operation used to derive the result. Note that reduction is from the top of the lexical stack to the top of the global stack and is only allowed if the nodes to be linked are not connected in the switch graph. This is why $N_1^\bullet$ cannot reduce with N_2° in the second diagram in Figure 10; the second shift is mandatory at this point. Note that as active nodes are removed, as in the first diagram reduction step linking 0=2, the switch graph contracts to just the unlinked nodes. After the reduction, only $N_2^\bullet$ is unlinked, so there can be no switch graph links. The link between node 4 and 5 is similarly removed almost as soon as it's introduced in the second reduction step. In the second diagram, the switch graph links persist as lexical literals are pushed onto the stack.

Shift-reduce parses stand in one-to-one correspondence with proof nets. The shift and reduce operations may be read directly from a proof net by working left to right through the literals. Between literals, the horizontal axiom links represent literals on the stack. Literals in the current lexical syntactic type represent the lexical stack. Literals that are shifted to the global stack eventually reduce by axiom linking with a literal to the right; literals that are reduced from the lexical stack axiom link to their left with a literal on the global stack.

3.3 Memoized Parsing

Using switch graphs, we reduce associative Lambek calculus parsing to an infinite binary phrase-structure grammar, where the non-terminals are normalized switch graphs. The phrase structure schemes are shown in Steedman notation in Figure 11. Lexical entries for syntactic type A are derived from the input polar tree rooted at $A^\bullet$. This polar tree yields a switch graph, which is always a valid lexical entry in the phrase structure grammar. Any result of axiom linking adjacent complementary pairs of literals in the polar tree that maintains switch-graph acyclicity is also permitted. For instance, allowing empty left-hand sides of sequents, the input type $A/(B/B)^\bullet$ would produce the literals $A_1^\bullet B_2^\bullet B_3^\circ$ with links 1-2 : $\{L_3\}$, 1-3 : $\{R_3\}$. This could be reduced by taking the axiom link 2=3, to produce the single node switch graph $A_1^\bullet$. In contrast, $(B/B)/A^\bullet$ produces the switch graph $B_1^\bullet B_2^\circ A_3^\circ$ with links 1-2, 1-3, and 2-3. Thus the complementary B literals may not be linked.

Given a pair of normal switch graphs, the binary rule scheme provides a finite set of derived switch graphs. One or more complementary literals may be axiom linked in a nested fashion at the borders of both switch graphs. These sequences are marked as Δ and $\overline{\Delta}$ and their positions are given relative to the other literals in the switch graph in Figure 11. Unlinked combinations are not necessary because the graph must eventually be connected. This scheme is non-deterministic in choice of Δ. For instance, an adverb input $(N_1\backslash S_2)/(N_4\backslash S_3)^\bullet$ produces the literals $N_1^\circ S_2^\bullet S_3^\circ N_4^\bullet$ and connections 1-2, 1-3:$\{L_4\}$, 1-4:$\{R_4\}$, 2-3:$\{L_4\}$, and 2-4:$\{R_4\}$. When it combines with a verb phrase input $N_5\backslash S_6^\bullet$ with literals $N_5^\circ S_6^\bullet$ and connections 5-6, then either the nominals may be linked (4=5), or the nominals and sentential literals may be linked (4=5, 3=6). The result of the single linking is $N_1^\circ S_2^\bullet S_3^\circ S_6^\bullet$ with connections 1-2, 1-3:$\{L_4\}$, 1-6:$\{R_4\}$, 2-3:$\{L_4\}$, and 2-6:$\{R_4\}$. The result of the double linking is simply $N_1^\circ S_6^\bullet$ with connection 1-6, or in other words, a verb phrase.

The dynamic programming equality condition is that two analyses are considered equal if they lead to the same normalized switch graphs. This equality is only considered up to the renaming of nodes and edges. Backpointers to derivations allow semantic readings to be packed in the form of lexical choices and axiom linkings. For instance, consider the two parses in Figure 12.

With a finite set of lexical entries, bottom-up memoized parsing schemes will terminate. We illustrate two derivations of a simple subject-verb-object construction in Figure 13. This is a so-called *spurious ambiguity* because the two derivations produce

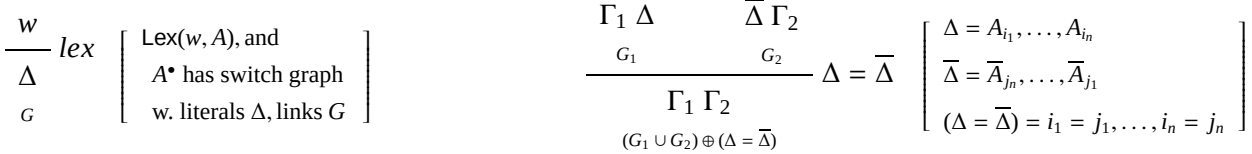


Figure 11: Phrase-Structure Schemes over Switch Graphs

Figure 12: Modifier Attachment Ambiguity Packing

the same semantic term. They are not spurious globally because the alternative linkings are required for adverbial modification and object relativization respectively. The ambiguity in the phrase structure grammar results from the associativity of the combination of axiom linkings. The two derivations do not propagate their ambiguity under the dynamic programming scheme precisely because they produce equivalent results. Nevertheless, a worthwhile optimization is to restrict the structure of combinations of linkings in the phrase-structure schemes to correspond to an unambiguous left-most linking strategy; this corresponds to the way in which other associative operators are parsed in programming language. For instance, $x+y+z$ will be assumed to be $x+(y+z)$ if + is defined to be right associative.

An unambiguous right-associative context-free grammar for linkings M over literals A and their complements $\overline{A}$ is:

$$M \rightarrow \; A\,\overline{A} \;\mid\; A\,M\,\overline{A} \;\mid\; A\,\overline{A}\,M \;\mid\; A\,M\,\overline{A}\,M$$

An example of packing for subject/object scope ambiguities is shown in Figure 14. The derivations in Figure 14 produce different semantic interpretations; one of these is subject-wide scope and the other object-wide scope. Unsurprisingly, the memoizing parser does not solve P = NP in the affirmitive (Pentus 2003). The size of the switch graphs on the intermediate structures is not bounded, nor is the number of alternative switch-paths between literals. It remains an open question as to whether the switch graph parser could be bounded for a fixed lexicon (Pentus 1997).

3.4 Empty Antecedents and Subtending

Lambek's calculus required the antecedent Γ in a sequent $\Gamma \Rightarrow \alpha : A$ to be non-empty. Proof nets derive theorems $(\Rightarrow CN/CN)$ and $((CN/CN)/(CN/CN) \Rightarrow CN/CN)$, as shown in Figure 15. These derivations both allow the construction of an output, namely the identity term $\lambda x.x$ and modifier syntactic type CN/CN, out of no input.

A literal A is said to *subtend* a complementary literal $\overline{A}$ if they are the leftmost and rightmost descendants of a $\wp$-link. In both of the examples in Figure 15, the output adjective CN/CN° unfolds to the sequence of literals $CN^{\bullet}CN^{\circ}$ in which the input $CN^{\bullet}$ subtends the output CN°. If literals that stand in a subtending relation are never linked, the set of theorems is restricted to those derivable in Lambek's original calculus.

Consider the proof net in Figure 16. An analysis in which S°_{8} linked to $S^{\bullet}_{11}$ and $N^{\bullet}_{9}$ linked to N°_{10} is not ruled out by Danos-Regnier acyclicity. It is ruled out by the subtending condition because S°_{8} subtends $S^{\bullet}_{11}$, being the leftmost and right most daughters of the $\wp$-node $(N_{10}\backslash S_{11})\backslash(N_9\backslash S_8)^{\circ}$. Further note that there are no cycles violating DR acyclicity; each of the sixteen switchings is acyclic.

4 Conclusion

We have introduced switch graphs for shift-reduce and CKY-style parsing of grammars in the associative Lambek calculus. Switch graphs encode

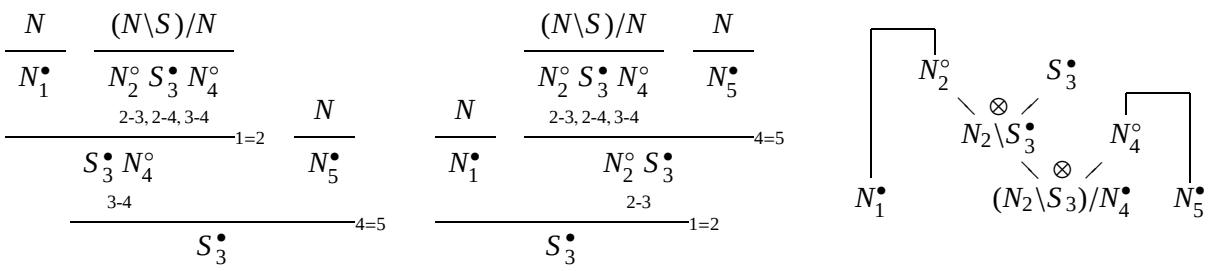


Figure 13: Left vs. Right Attachment: Packing Locally Spurious Attachment Ambiguity

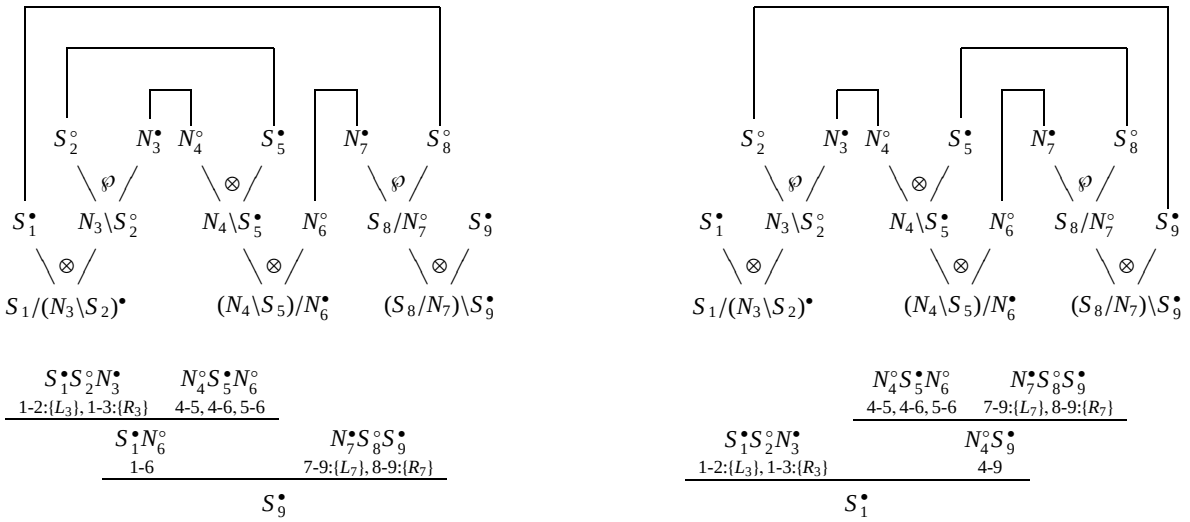


Figure 14: Scope Ambiguity: Partial Proof Structure Fragments with Phrase Structure

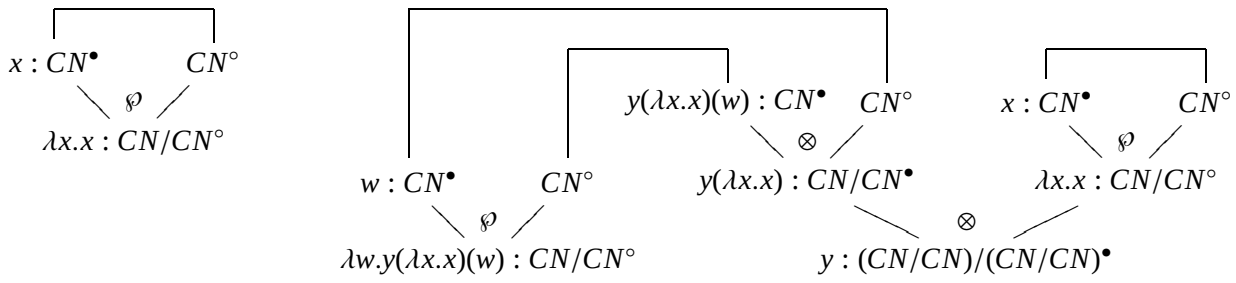


Figure 15: Subtending Examples

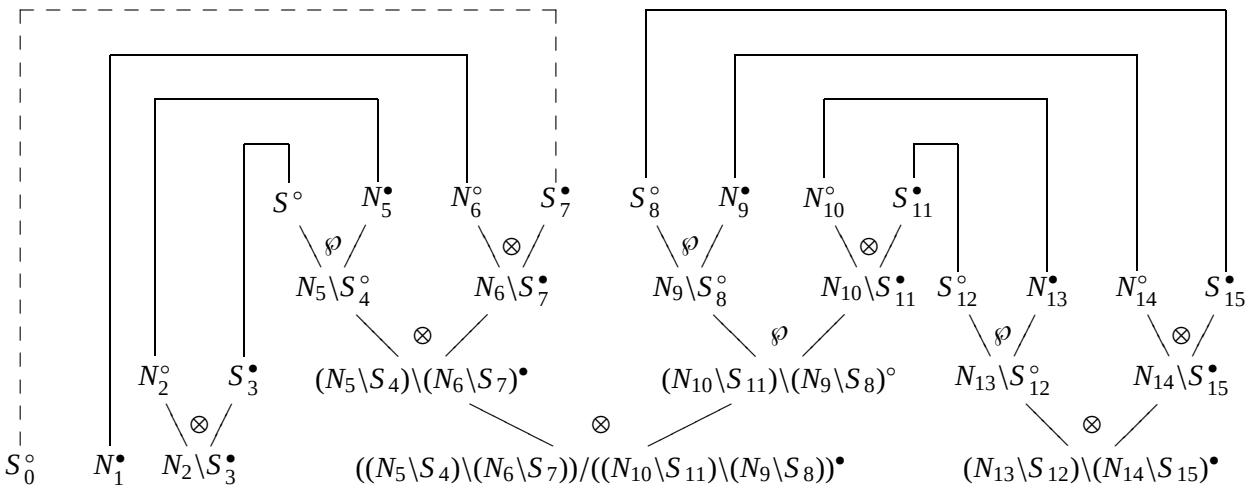


Figure 16: Higher-Order Example: Adverbial Intensifier

the axiom-linking possibilities of a sequence of unlinked literals deriving from underlying polar trees. We introduced two parsers based on switch graphs. The shift-reduce parsers are memory efficient and parses correspond uniquely to (cut free) proof nets. They can be made more efficient by bounding stack size. The memoizing parsers are able to pack attachment and scope distinctions that lead to different λ-terms but have the same combinatory possibilities.

References

D. Bechet. 2003. Incremental parsing of lambek calculus using proof-net interfaces. In *Proc. of the 8th International Workshop on Parsing Technologies*.

V. Danos and L. Regnier. 1989. The structure of multiplicatives. *Arch. Math. Logic*, 28:181–203.

P. de Groote and C. Retoré. 1996. On the semantic readings of proof nets. In *Proc. of Formal Grammar*, pages 57–70.

M. Faddo and G. Morrill. 2005. The Lambek calculus with brackets. In P. Scott, C. Casadio, and R. Seely, editors, *Language and Grammar: Studies in Math. Ling. and Nat. Lang.* CSLI Press, Stanford.

J.-Y. Girard. 1987. Linear logic. *Theoret. Comput. Sci.*, 50:1–102.

F. Lamarche. 1994. Proof nets for intuitionistic linear logic I: Essential nets. Technical report, Imperial College, London.

J. Lambek. 1958. The mathematics of sentence structure. *Amer. Math. Mon.*, 65:154–170.

R. Montague. 1970. Universal grammar. *Theoria*, 36:373–398.

G. Morrill. 1996. Memoisation of categorial proof nets: parallelism in categorial processing. Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

G. Morrill. 2000. Incremental processing and acceptability. *Comput. Ling.*, 26(3):319–338.

M. Pentus. 1995. Models for the Lambek calculus. *Annals of Pure and Applied Logic*, 75(1–2):179–213.

M. Pentus. 1997. Product-free Lambek calculus and context-free grammars. *Journal of Symbolic Logic*, 62(2):648–660.

M. Pentus. 2003. Lambek calculus is NP-complete. Technical Report TR-203005, CUNY Graduate Center.

D. Roorda. 1991. *Resource logics: Proof-theoretical investigations*. Ph.D. thesis, Universiteit van Amsterdam.

Parsing with Soft and Hard Constraints on Dependency Length*

Jason Eisner and **Noah A. Smith**
Department of Computer Science / Center for Language and Speech Processing
Johns Hopkins University, Baltimore, MD 21218 USA
{jason,nasmith}@cs.jhu.edu

Abstract

In lexicalized phrase-structure or dependency parses, a word's modifiers tend to fall near it in the string. We show that a crude way to use dependency length as a parsing feature can substantially improve parsing speed and accuracy in English and Chinese, with more mixed results on German. We then show similar improvements by imposing *hard* bounds on dependency length and (additionally) modeling the resulting sequence of parse fragments. This simple "vine grammar" formalism has only finite-state power, but a context-free parameterization with some extra parameters for stringing fragments together. We exhibit a linear-time chart parsing algorithm with a low grammar constant.

1 Introduction

Many modern parsers identify the head word of each constituent they find. This makes it possible to identify the word-to-word dependencies implicit in a parse.[1] (Some parsers, known as dependency parsers, even return these dependencies as their primary output.)

Why bother to identify these dependencies? The typical reason is to model the fact that some word pairs are more likely than others to engage in a dependency relationship.[2] In this paper, we propose a different reason to identify dependencies in candidate parses: to evaluate not the dependency's word pair but its *length* (i.e., the *string distance* between the two words). Dependency lengths differ from typical parsing features in that they cannot be determined from tree-local information. Though lengths are not usually considered, we will see that bilexical dynamic-programming parsing algorithms can easily consider them as they build the parse.

Soft constraints. Like any other feature of trees, dependency lengths can be explicitly used as features in a probability model that chooses among trees. Such a model will tend to disfavor long dependencies (at least of some kinds), as these are empirically rare. In the first part of the paper, we show that such features improve a simple baseline dependency parser.

Hard constraints. If the bias against long dependencies is strengthened into a hard constraint that absolutely prohibits long dependencies, then the parser turns into a partial parser with only finite-state power. In the second part of the paper, we show how to perform chart parsing in asymptotic linear time with a low grammar constant. Such a partial parser does less work than a full parser in practice, and in many cases recovers a more precise set of dependencies (with little loss in recall).

2 Short Dependencies in Langugage

We assume that correct parses exhibit a "short-dependency preference": a word's dependents tend to be close to it in the string.[3] If the j^{th} word of a sentence depends on the i^{th} word, then $|i-j|$ tends to be

* This work was supported by NSF ITR grant IIS-0313193 to the first author and a fellowship from the Fannie and John Hertz Foundation to the second author. The views expressed are not necessarily endorsed by the sponsors. The authors thank Mark Johnson, Eugene Charniak, Charles Schafer, Keith Hall, and John Hale for helpful discussion and Elliott Drábek and Markus Dreyer for insights on (respectively) Chinese and German parsing. They also thank an anonymous reviewer for suggesting the German experiments.

[1] In a phrase-structure parse, if phrase X headed by word token x is a subconstituent of phrase Y headed by word token $y \neq x$, then x is said to depend on y. In a more powerful compositional formalism like LTAG or CCG, dependencies can be extracted from the derivation tree.

[2] It has recently been questioned whether these "bilexical" features actually contribute much to parsing performance (Klein and Manning, 2003; Bikel, 2004), at least when one has only a million words of training.

[3] In this paper, we consider only a crude notion of "closeness": the number of intervening words. Other distance measures could be substituted or added (following the literature on heavy-shift and sentence comprehension), including the phonological, morphological, syntactic, or referential (given/new) complexity of the intervening material (Gibson, 1998). In parsing, the most relevant previous work is due to Collins (1997), who considered three binary features of the intervening material: did it contain (a) any word tokens at all, (b) any verbs, (c) any commas or colons? Note that (b) is effective because it measures the length of a dependency in terms of the number of alternative attachment sites that the dependent skipped over, a notion that could be generalized. Similarly, McDonald et al. (2005) separately considered each of the intervening POS tags.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 30–41,
Vancouver, October 2005.

small. This implies that neither i nor j is modified by complex phrases that fall between i and j. In terms of phrase structure, it implies that the *phrases* modifying word i from a given side tend to be (1) few in number, (2) ordered so that the longer phrases fall farther from i, and (3) internally structured so that the bulk of each phrase falls on the side of j away from i.

These principles can be blamed for several linguistic phenomena. (1) helps explain the "late closure" or "attach low" heuristic (e.g., Frazier, 1979; Hobbs and Bear, 1990): a modifier such as a PP is more likely to attach to the closest appropriate head. (2) helps account for heavy-shift: when an NP is long and complex, *take NP out*, *put NP on the table*, and *give NP to Mary* are likely to be rephrased as *take out NP*, *put on the table NP*, and *give Mary NP*. (3) explains certain non-canonical word orders: in English, a noun's left modifier must become a right modifier if and only if it is right-heavy (*a taller politician* vs. *a politician taller than all her rivals*[4]), and a verb's left modifier may extrapose its right-heavy portion (*An aardvark walked in who had circumnavigated the globe*[5]).

Why should sentences prefer short dependencies? Such sentences may be easier for humans to produce and comprehend. Each word can quickly "discharge its responsibilities," emitting or finding all its dependents soon after it is uttered or heard; then it can be dropped from working memory (Church, 1980; Gibson, 1998). Such sentences also succumb nicely to disambiguation heuristics that *assume* short dependencies, such as low attachment. Thus, to improve comprehensibility, a speaker can make stylistic choices that shorten dependencies (e.g., heavy-shift), and a language can categorically prohibit some structures that lead to long dependencies (**a taller-than-all-her-rivals politician*; **the sentence that another sentence that had center-embedding was inside was incomprehensible*).

Such functionalist pressures are not all-powerful. For example, many languages use SOV basic word order where SVO (or OVS) would give shorter dependencies. However, where the data exhibit some short-dependency preference, computer parsers as well as human parsers can obtain speed and accuracy benefits by exploiting that fact.

3 Soft Constraints on Dependency Length

We now enhance simple baseline probabilistic parsers for English, Chinese, and German so that they consider dependency lengths. We confine ourselves (throughout the paper) to parsing part-of-speech (POS) tag sequences. This allows us to ignore data sparseness, out-of-vocabulary, smoothing, and pruning issues, but it means that our accuracy measures are not state-of-the-art. Our techniques could be straightforwardly adapted to (bi)lexicalized parsers on actual word sequences, though not necessarily with the same success.

3.1 Grammar Formalism

Throughout this paper we will use split bilexical grammars, or SBGs (Eisner, 2000), a notationally simpler variant of split head-automaton grammars, or SHAGs (Eisner and Satta, 1999). The formalism is context-free. We define here a probabilistic version,[6] which we use for the baseline models in our experiments. They are only baselines because the SBG generative process does *not* take note of dependency length.

An SBG is an tuple $\mathcal{G} = (\Sigma, \$, L, R)$. Σ is an alphabet of words. (In our experiments, we parse only POS tag sequences, so Σ is actually an alphabet of tags.) $\$ \notin \Sigma$ is a distinguished root symbol; let $\bar{\Sigma} = \Sigma \cup \{\$\}$. L and R are functions from $\bar{\Sigma}$ to probabilistic ϵ-free finite-state automata over Σ. Thus, for each $w \in \bar{\Sigma}$, the SBG specifies "left" and "right" probabilistic FSAs, L_w and R_w.

We use $\mathcal{L}_w(\mathcal{G}) : \bar{\Sigma}^* \rightarrow [0, 1]$ to denote the probabilistic context-free language of phrases headed by w. $\mathcal{L}_w(\mathcal{G})$ is defined by the following simple top-down stochastic process for sampling from it:

[4] Whereas **a politician taller* and **a taller-than-all-her-rivals politician* are not allowed. The phenomenon is pervasive.

[5] This actually splits the heavy left dependent *[an aardvark who ...]* into two non-adjacent pieces, moving the heavy second piece. By slightly stretching the *aardvark-who* dependency in this way, it greatly shortens *aardvark-walked*. The same is possible for heavy, non-final right dependents: *I met an aardvark yesterday who had circumnavigated the globe* again stretches *aardvark-who*, which greatly shortens *met-yesterday*. These examples illustrate (3) and (2) respectively. However, the resulting non-contiguous constituents lead to non-projective parses that are beyond the scope of this paper.

[6] There is a straightforward generalization to *weighted* SBGs, which need not have a stochastic generative model.

1. Sample from the finite-state language $\mathcal{L}(L_w)$ a sequence $\lambda = w_{-1}w_{-2}\ldots w_{-\ell} \in \Sigma^*$ of left children, and from $\mathcal{L}(R_w)$ a sequence $\rho = w_1 w_2 \ldots w_r \in \Sigma^*$ of right children. Each sequence is found by a random walk on its probabilistic FSA. We say the children *depend* on w.
2. For each i from $-\ell$ to r with $i \neq 0$, recursively sample $\alpha_i \in \Sigma^*$ from the context-free language $\mathcal{L}_{w_i}(\mathcal{G})$. It is this step that indirectly determines dependency lengths.
3. Return $\alpha_{-\ell}\ldots\alpha_{-2}\alpha_{-1}w\alpha_1\alpha_2\ldots\alpha_r \in \bar{\Sigma}^*$, a concatenation of strings.

Notice that w's left children λ were generated in reverse order, so w_{-1} and w_1 are its closest children while $w_{-\ell}$ and w_r are the farthest.

Given an input sentence $\omega = w_1 w_2 \ldots w_n \in \Sigma^*$, a parser attempts to recover the highest-probability derivation by which $\$\omega$ could have been generated from $\mathcal{L}_{\$}(\mathcal{G})$. Thus, \$ plays the role of w_0. A sample derivation is shown in Fig. 1a. Typically, $L_{\$}$ and $R_{\$}$ are defined so that \$ must have no left children ($\ell = 0$) and at most one right child ($r \leq 1$), the latter serving as the conventional root of the parse.

3.2 Baseline Models

In the experiments reported here, we defined only very simple automata for L_w and R_w ($w \in \Sigma$). However, we tried three automaton types, of varying quality, so as to evaluate the benefit of adding length-sensitivity at three different levels of baseline performance.

In model A (the worst), each automaton has topology $\circledcirc\!\!\rightleftarrows$, with a single state q_1, so token w's left dependents are conditionally independent of one another given w. In model C (the best), each automaton $\circledcirc\!\!\longrightarrow\!\circledcirc\!\!\rightleftarrows$ has an extra state q_0 that allows the first (closest) dependent to be chosen differently from the rest. Model B is a compromise:[7] it is like model A, but each type $w \in \Sigma$ may have an elevated or reduced probability of having no dependents at all. This is accomplished by using automata $\circledcirc\!\!\longrightarrow\!\circledcirc\!\!\rightleftarrows$ as in model C, which allows the stopping probabilities $p(\text{STOP} \mid q_0)$ and $p(\text{STOP} \mid q_1)$ to differ, but tying the conditional distributions $p(q_0 \xrightarrow{w} q_1 \mid q_0, \neg\text{STOP})$ and $p(q_1 \xrightarrow{w} q_1 \mid q_1, \neg\text{STOP})$.

Finally, in §3, $L_{\$}$ and $R_{\$}$ are restricted as above, so $R_{\$}$ gives a probability distribution over Σ only.

3.3 Length-Sensitive Models

None of the baseline models A–C *explicitly* model the distance between a head and child. We enhanced them by multiplying in some extra length-sensitive factors when computing a tree's probability. For each dependency, an extra factor $p(\Delta \mid \ldots)$ is multiplied in for the probability of the dependency's length $\Delta = |i - j|$, where i and j are the positions of the head and child in the *surface* string.[8]

Again we tried three variants. In one version, this new probability $p(\Delta| \ldots)$ is conditioned only on the direction $d = \text{sign}(i - j)$ of the dependency. In another version, it is conditioned only on the POS tag h of the head. In a third version, it is conditioned on d, h, and the POS tag c of the child.

3.4 Parsing Algorithm

Fig. 2a gives a variant of Eisner and Satta's (1999) SHAG parsing algorithm, adapted to SBGs, which are easier to understand.[9] (We will modify this algorithm later in §4.) The algorithm obtains $O(n^3)$ runtime, despite the need to track the position of head words, by exploiting the conditional independence between a head's left children and right children. It builds "half-constituents" denoted by $\triangle$ (a head word together with some modifying phrases on the right, i.e., $w\alpha_1\ldots\alpha_r$) and $\triangle$ (a head word together with some modifying phrases on the left, i.e., $\alpha_{-\ell}\ldots\alpha_{-1}w$). A new dependency is introduced when $\triangle + \triangle$ are combined to get $\square$ or $\square$ (a pair of linked head words with all the intervening phrases, i.e., $w\alpha_1\ldots\alpha_r\alpha'_{-\ell'}\ldots\alpha'_{-1}w'$, where w is respectively the parent or child of w'). One can then combine $\square + \triangle = \triangle$, or

[7] It is equivalent to the "dependency model with valence" of Klein and Manning (2004).

[8] Since the Δ values are fully determined by the tree but every $p(\Delta \mid \ldots) \leq 1$, this crude procedure simply reduces the probability mass of every legal tree. The resulting model is *deficient* (does not sum to 1); the remaining probability mass goes to impossible trees whose putative dependency lengths Δ are inconsistent with the tree structure. We intend in future work to explore non-deficient models (log-linear or generative), but even the present crude approach helps.

[9] The SHAG notation was designed to highlight the connection to *non*-split HAGs.

 . Only $O(n^3)$ combinations are possible in total when parsing a length-n sentence.

3.5 A Note on Word Senses

[This section may be skipped by the casual reader.]

A remark is necessary about $:w$ and $:w'$ in Fig. 2a, which represent *senses* of the words at positions h and h'. Like past algorithms for SBGs (Eisner, 2000), Fig. 2a is designed to be a bit more general and integrate sense disambiguation into parsing. It formally runs on an input $\Omega = W_1 \dots W_n \subseteq \Sigma^*$, where each $W_i \subseteq \Sigma$ is a "confusion set" over possible values of the i^{th} word w_i. The algorithm recovers the highest-probability derivation that generates $\$\omega$ for *some* $\omega \in \Omega$ (i.e., $\omega = w_1 \dots w_n$ with $(\forall i) w_i \in W_i$).

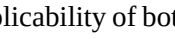
This extra level of generality is not needed for any of our experiments, but it is needed for SBG parsers to be as flexible as SHAG parsers. We include it in this paper to broaden the applicability of both Fig. 2a and our extension of it in §4.

The "senses" can be used in an SBG to pass a finite amount of information between the left and right children of a word, just as SHAGs allow.[10] For example, to model the fronting of a direct object, an SBG might use a special sense of a verb, whose automata tend to generate both one more noun in λ and one fewer noun in ρ.

Senses can also be used to pass information between parents and children. Important uses are to encode lexical senses, or to enrich the dependency parse with constituent labels or dependency labels (Eisner, 2000). For example, the input token $W_i = \{bank_1/N/NP,\ bank_2/N/NP,\ bank_3/V/VP,\ bank_3/V/S\} \subset \Sigma$ allows four "senses" of bank, namely two nominal meanings, and two *syntactically different* versions of the verbal meaning, whose automata require them to expand into VP and S phrases respectively.

The cubic runtime is proportional to the number of ways of instantiating the inference rules in Fig. 2a: $O(n^2(n + t')tg^2)$, where $n = |\Omega|$ is the input length, $g = \max_{i=1}^{n} |W_i|$ bounds the size of a confusion set, t bounds the number of states per automaton, and $t' \leq t$ bounds the number of automaton transitions from a state that emit the same word. For deterministic automata, $t' = 1$.[11]

3.6 Probabilistic Parsing

It is easy to make the algorithm of Fig. 2a length-sensitive. When a new dependency is added by an ATTACH rule that combines  , the annotations on and suffice to determine the dependency's length $\Delta = |h - h'|$, direction $d = \text{sign}(h - h')$, head word w, and child word w'.[12] So the additional cost of such a dependency, e.g. $p(\Delta \mid d, w, w')$, can be included as the weight of an extra antecedent to the rule, and so included in the weight of the resulting  or .

To execute the inference rules in Fig. 2a, we use a prioritized agenda. Derived items such as  , , , and are prioritized by their Viterbi-inside probabilities. This is known as *uniform-cost search* or *shortest-hyperpath search* (Nederhof, 2003). We halt as soon as a full parse (the *accept* item) pops from the agenda, since uniform-cost search (as a special case of the A* algorithm) guarantees this to be the maximum-probability parse. No other pruning is done.

[10] Fig. 2a enhances the Eisner-Satta version with explicit senses while matching its asymptotic performance. On this point, see (Eisner and Satta, 1999, §8 and footnote 6). However, it does have a practical slowdown, in that START-LEFT nondeterministically guesses every possible sense of W_i, and these senses are pursued separately. To match the Eisner-Satta algorithm, we should not need to commit to a word's sense until we have seen all its left children. That is, left triangles and left trapezoids should not carry a sense $:w$ at all, except for the completed left triangle (marked F) that is produced by FINISH-LEFT. FINISH-LEFT should choose a sense w of W_h according to the final state q, which reflects knowledge of W_h's left children. For this strategy to work, the transitions in L_w (used by ATTACH-LEFT) must not depend on the particular sense w but only on W. In other words, all $L_w : w \in W_h$ are really copies of a shared L_{W_h}, except that they may have different final states. This requirement involves no loss of generality, since the nondeterministic shared L_{W_h} is free to branch as soon as it likes onto paths that commit to the various senses w.

[11] Confusion-set parsing may be regarded as parsing a particular lattice with n states and ng arcs. The algorithm can be generalized to lattice parsing, in which case it has runtime $O(m^2(n + t')t)$ for a lattice of n states and m arcs. Roughly, $h : w$ is replaced by an arc, while i is replaced by a state and $i - 1$ is replaced by the same state.

[12] For general lattice parsing, it is not possible to determine Δ while applying this rule. There h and h' are arcs in the lattice, not integers, and different paths from h to h' might cover different numbers of words. Thus, if one still wanted to measure dependency length in words (rather than in, say, milliseconds of speech), each item would have to record its width explicitly, leading in general to more items and increased runtime.

With a prioritized agenda, a probability model that more sharply discriminates among parses will typically lead to a faster parser. (Low-probability constituents languish at the back of the agenda and are never pursued.) We will see that the length-sensitive models do run faster for this reason.

3.7 Experiments with Soft Constraints

We trained models A–C, using unsmoothed maximum likelihood estimation, on three treebanks: the Penn (English) Treebank (split in the standard way, §2–21 train/§23 test, or 950K/57K words), the Penn Chinese Treebank (80% train/10% test or 508K/55K words), and the German TIGER corpus (80%/10% or 539K/68K words).[13] Estimation was a simple matter of counting automaton events and normalizing counts into probabilities. For each model, we also trained the three length-sensitive versions described in §3.3.

The German corpus contains non-projective trees. None of our parsers can recover non-projective dependencies (nor can our models produce them). This fact was ignored when counting events for maximum likelihood estimation: in particular, we always trained L_w and R_w on the sequence of w's immediate children, even in non-projective trees.

Our results (Tab. 1) show that sharpening the probabilities with the most sophisticated distance factors $p(\Delta \mid d, h, c)$, consistently improved the *speed* of all parsers.[14] The change to the code is trivial. The only overhead is the cost of looking up and multiplying in the extra distance factors.

Accuracy also improved over the baseline models of English and Chinese, as well as the simpler baseline models of German. Again, the most sophisticated distance factors helped most, but even the simplest distance factor usually obtained most of the accuracy benefit.

German model C fell slightly in accuracy. The speedup here suggests that the probabilities were sharpened, but often in favor of the wrong parses. We did not analyze the errors on German; it may be relevant that 25% of the German sentences contained a non-projective dependency between non-punctuation tokens.

Studying the parser output for English, we found that the length-sensitive models preferred closer attachments, with 19.7% of tags having a nearer parent in the best parse under model C with $p(\Delta \mid d, h, c)$ than in the original model C, 77.7% having a parent at the same distance, and only 2.5% having a farther parent. The surviving long dependencies (at any length > 1) tended to be much more accurate, while the (now more numerous) length-1 dependencies were slightly less accurate than before.

We caution that length sensitivity's most dramatic improvements to accuracy were on the worse baseline models, which had more room to improve. The better baseline models (B and C) were already able to indirectly capture some preference for short dependencies, by learning that some parts of speech were unlikely to have multiple left or multiple right dependents. Enhancing B and C therefore contributed less, and indeed may have had some harmful effect by over-penalizing some structures that were already appropriately penalized.[15] It remains to be seen, therefore, whether distance features would help state-of-the art parsers that are already much better than model C. Such parsers may already incorporate features that indirectly impose a good model of distance, though perhaps not as cheaply.

4 Hard Dependency-Length Constraints

We have seen how an explicit model of distance can improve the speed and accuracy of a simple probabilistic dependency parser. Another way to capitalize on the fact that most dependencies are local is to impose a *hard constraint* that simply forbids long dependencies.

The dependency trees that satisfy this constraint yield a regular string language.[16] The constraint prevents arbitrarily deep center-embedding, as well as arbitrarily many direct dependents on a given head,

[13] Heads were extracted for English using Michael Collins' rules and Chinese using Fei Xia's rules (defaulting in both cases to right-most heads where the rules fail). German heads were extracted using the TIGER Java API; we discarded all resulting dependency structures that were cyclic or unconnected (6%).

[14] We measure speed abstractly by the number of items built and pushed on the agenda.

[15] Owing to our deficient model. A log-linear or discriminative model would be trained to correct for overlapping penalties and would avoid this risk. Non-deficient generative models are also possible to design, along lines similar to footnote 16.

[16] One proof is to construct a strongly equivalent CFG without center-embedding (Nederhof, 2000). Each nonterminal has the form $\langle w, q, i, j \rangle$, where $w \in \Sigma$, q is a state of L_w or R_w, and $i, j \in \{0, 1, \ldots k-1, \geq k\}$. We leave the details as an exercise.

	English (Penn Treebank)				Chinese (Chinese Treebank)				German (TIGER Corpus)			
	recall (%)		runtime	model	recall (%)		runtime	model	recall (%)		runtime	model
model	train	test	test	size	train	test	test	size	train	test	test	size
A (1 state)	62.0	62.2	93.6	1,878	50.7	49.3	146.7	782	70.9	72.0	53.4	1,598
+ $p(\Delta \mid d)$	70.1	70.6	97.0	2,032	59.0	58.0	161.9	1,037	72.3	73.0	53.2	1,763
+ $p(\Delta \mid h)$	70.5	71.0	94.7	3,091	60.5	59.1	148.3	1,759	73.1	74.0	48.3	2,575
+ $p(\Delta \mid d, h, c)$	72.8	73.1	70.4	16,305	62.2	60.6	106.7	7,828	75.0	75.1	31.6	12,325
B (2 states, tied arcs)	69.7	70.4	93.5	2,106	56.7	56.2	151.4	928	73.7	75.1	52.9	1,845
+ $p(\Delta \mid d)$	72.6	73.2	95.3	2,260	60.2	59.5	156.9	1,183	72.9	73.9	52.6	2,010
+ $p(\Delta \mid h)$	73.1	73.7	92.1	3,319	61.6	60.7	144.2	1,905	74.1	75.3	47.6	2,822
+ $p(\Delta \mid d, h, c)$	75.3	75.6	67.7	16,533	62.9	61.6	104.0	7,974	75.2	75.5	31.5	12,572
C (2 states)	72.7	73.1	90.3	3,233	61.8	61.0	148.3	1,314	75.6	76.9	48.5	2,638
+ $p(\Delta \mid d)$	73.9	74.5	91.7	3,387	61.5	60.6	154.7	1,569	74.3	75.0	48.9	2,803
+ $p(\Delta \mid h)$	74.3	75.0	88.6	4,446	63.1	61.9	141.9	2,291	75.2	76.3	44.3	3,615
+ $p(\Delta \mid d, h, c)$	75.3	75.5	66.6	17,660	63.4	61.8	103.4	8,360	75.1	75.2	31.0	13,365

Table 1: Dependency parsing of POS tag sequences with simple probabilistic split bilexical grammars. The models differ only in how they weight the same candidate parse trees. Length-sensitive models are larger but can improve dependency accuracy and speed. (**Recall** is measured as the fraction of non-punctuation tags whose correct parent (if not the **$** symbol) was correctly recovered by the parser; it equals precision, unless the parser left some sentences unparsed (or incompletely parsed, as in §4), in which case precision is higher. **Runtime** is measured abstractly as the average number of items (i.e.,  ,  ,  , ) built per word. **Model size** is measured as the number of nonzero parameters.)

either of which would allow the non-regular language $\{a^n bc^n : 0 < n < \infty\}$. It *does* allow arbitrarily deep right- or left-branching structures.

4.1 Vine Grammars

The tighter the bound on dependency length, the fewer parse trees we allow and the faster we can find them using the algorithm of Fig. 2a. If the bound is too tight to allow the correct parse of some sentence, we would still like to allow an accurate partial parse: a sequence of accurate parse fragments (Hindle, 1990; Abney, 1991; Appelt et al., 1993; Chen, 1995; Grefenstette, 1996). Furthermore, we would like to use the fact that some fragment sequences are presumably more likely than others.

Our partial parses will look like the one in Fig. 1b. where 4 subtrees rather than 1 are dependent on **$**. This is easy to arrange in the SBG formalism. We merely need to construct our SBG so that the automaton $R_{\$}$ is now permitted to generate multiple children—the roots of parse fragments.

This $R_{\$}$ is a probabilistic finite-state automaton that describes legal or likely root sequences in Σ^*. In our experiments in this section, we will train it to be a first-order (bigram) Markov model. (Thus we construct $R_{\$}$ in the usual way to have $|\Sigma| + 1$ states, and train it on data like the other left and right automata. During generation, its state remembers the previously generated root, if any. Recall that we are working with POS tag sequences, so the roots, like all other words, are tags in Σ.)

The 4 subtrees in Fig. 1b appear as so many bunches of grapes hanging off a vine. We refer to the dotted dependencies upon **$** as *vine dependencies*, and the remaining, bilexical dependencies as *tree dependencies*.

One might informally use the term "vine grammar" (VG) for any generative formalism, intended for partial parsing, in which a parse is a constrained sequence of trees that cover the sentence. In general, a VG might use a two-part generative process: first generate a finite-state sequence of roots, then expand the roots according to some more powerful formalism. Conveniently, however, SBGs and other dependency grammars can integrate these two steps into a single formalism.

4.2 Feasible Parsing

Now, for both speed and accuracy, we will restrict the trees that may hang from the vine. We define a *feasible* parse under our SBG to be one in which all *tree* dependencies are short, i.e., their length never exceeds some hard bound k. The vine dependencies may have unbounded length, of course, as in Fig. 1b.

Sentences with feasible parses form a regular language. This would also be true under other definitions of feasibility, e.g., we could have limited the depth or width of each tree on the vine. However, that would have ruled out deeply right-branching trees, which are very common in language, and

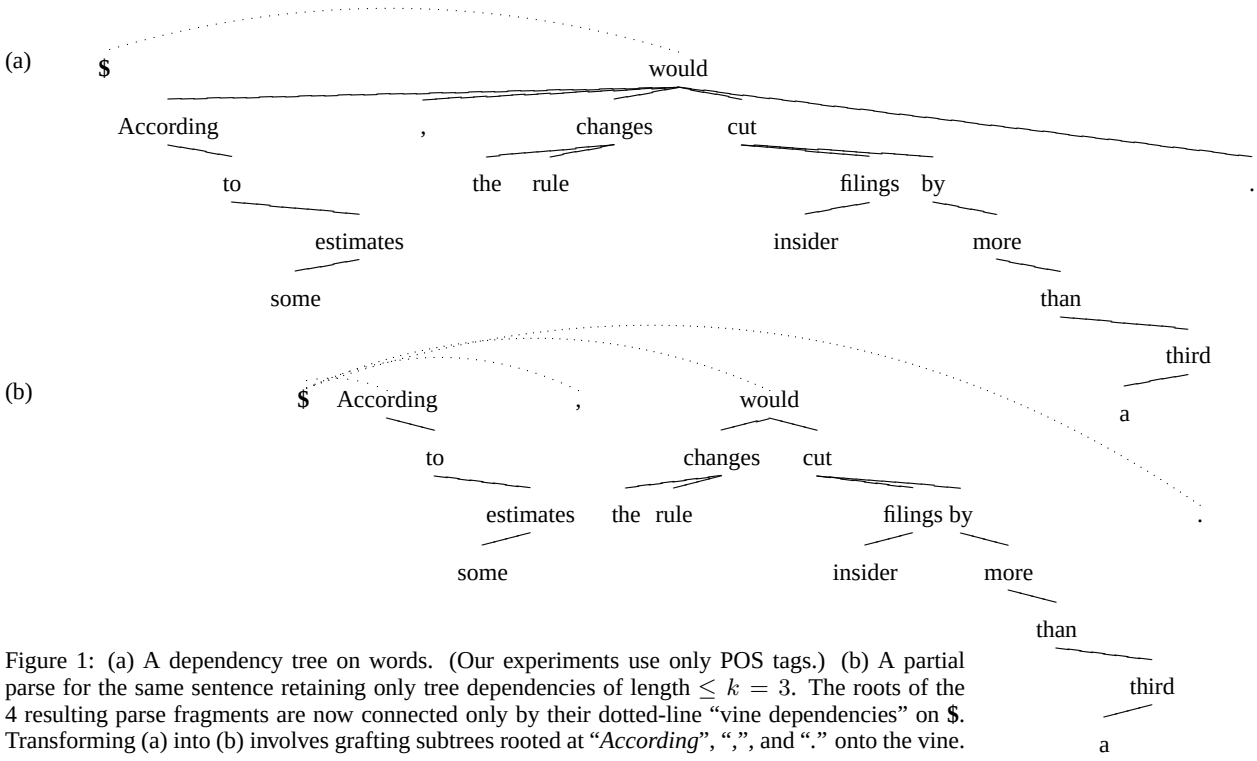


Figure 1: (a) A dependency tree on words. (Our experiments use only POS tags.) (b) A partial parse for the same sentence retaining only tree dependencies of length $\leq k = 3$. The roots of the 4 resulting parse fragments are now connected only by their dotted-line "vine dependencies" on **$**. Transforming (a) into (b) involves grafting subtrees rooted at "*According*", ",", and "." onto the vine.

are also the traditional way to describe finite-state sublanguages within a context-free grammar. By contrast, our limitation on dependency length ensures regularity while still allowing (for any bound $k \geq 1$) arbitrarily wide and deep trees, such as $a \rightarrow b \rightarrow \ldots \rightarrow$ *root* $\leftarrow \ldots \leftarrow y \leftarrow z$.

Our goal is to find the *best feasible* parse (if any). Rather than transform the grammar as in footnote 16, our strategy is to modify the parser so that it only considers feasible parses. The interesting problem is to achieve linear-time parsing with a grammar constant that is as small as for ordinary parsing.

We also correspondingly modify the training data so that we only train on feasible parses. That is, we break any long dependencies and thereby fragment each training parse (a single tree) into a vine of one or more restricted trees. When we break a child-to-parent dependency, we reattach the child to **$**.[17] This process, *grafting*, is illustrated in Fig. 1. Although this new parse may score less than 100% recall of the original dependencies, it is the best feasible parse, so we would like to train the parser to find it.[18] By training on the modified data, we learn more appropriate statistics for both $R_{\$}$ and the other automata. If we trained on the original trees, we would inaptly learn that $R_{\$}$ always generates a single root rather than a certain kind of sequence of roots.

For evaluation, we score tree dependencies in our feasible parses against the tree dependencies in the *unmodified* gold standard parses, which are not necessarily feasible. We also show oracle performance.

4.3 Approach #1: FSA Parsing

Since we are now dealing with a regular language, it is possible in principle to use a weighted finite-state automaton (FSA) to search for the best feasible parse. The idea is to find the highest-weighted path that accepts the input string $\omega = w_1 w_2 \ldots w_n$. Using the Viterbi algorithm, this takes time $O(n)$.

The trouble is that this linear runtime hides a constant factor, which depends on the size of the relevant part of the FSA and may be enormous for any correct FSA.[19]

Consider an example from Fig 1b. After nondeterministically reading $w_1 \ldots w_{11}$ = *According... insider* along the *correct* path, the FSA state must record (at least) that *insider* has no parent yet and that $R_{\$}$ and R_{cut} are in particular states that

[17] Any dependency *covering* the child must also be broken to preserve projectivity. This case arises later; see footnote 25.

[18] Although the parser will still not be able to find it if it is non-projective (possible in German). Arguably we should have defined "feasible" to also require projectivity, but we did not.

[19] The full runtime is $O(nE)$, where E is the number of FSA edges, or for a tighter estimate, the number of FSA edges that can be traversed by reading ω.

may still accept more children. Else the FSA cannot know whether to accept a continuation $w_{12} \dots w_n$.

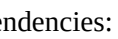
In general, after parsing a prefix $w_1 \dots w_j$, the FSA state must somehow record information about all incompletely linked words in the past. It must record the sequence of past words w_i ($i \leq j$) that still need a parent or child in the future; if w_i still needs a child, it must also record the state of R_{w_i}.

Our restriction to dependency length $\leq k$ is what allows us to build a *finite*-state machine (as opposed to some kind of pushdown automaton with an unbounded number of configurations). We need only build the *finitely* many states where the incompletely linked words are limited to at most $w_0 =$ **$** and the k most recent words, $w_{j-k+1} \dots w_j$. Other states cannot extend into a feasible parse, and can be pruned.

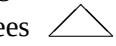
However, this still allows the FSA to be in $O(2^k t^{k+1})$ different states after reading $w_1 \dots w_j$. Then the runtime of the Viterbi algorithm, though linear in n, is exponential in k.

4.4 Approach #2: Ordinary Chart Parsing

A much better idea for most purposes is to use a chart parser. This allows the usual dynamic programming techniques for reusing computation. (The FSA in the previous section failed to exploit many such opportunities: exponentially many states would have proceeded redundantly by building the same $w_{j+1}w_{j+2}w_{j+3}$ constituent.)

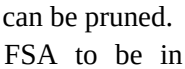
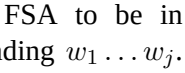
It is simple to restrict our algorithm of Fig. 2a to find only feasible parses. It is the ATTACH rules  that add dependencies: simply use a side condition to block them from applying unless $|h - h'| \leq k$ (short tree dependency) or $h = 0$ (vine dependency). This ensures that all  and  will have width $\leq k$ or have their left edge at 0.

One might now incorrectly expect runtime linear in n: the number of possible ATTACH combinations is reduced from $O(n^3)$ to $O(nk^2)$, because i and h' are now restricted to a narrow range given h.

Unfortunately, the half-constituents  and  may still be arbitrarily wide, thanks to arbitrary right- and left-branching: a feasible vine parse may be a sequence of wide trees  . Thus there are $O(n^2k)$ possible COMPLETE combinations, not to mention $O(n^2)$ ATTACH-RIGHT combinations for which $h = 0$. So the runtime remains quadratic.

4.5 Approach #3: Specialized Chart Parsing

How, then, do we get linear runtime *and* a reasonable grammar constant? We give two ways to achieve runtime of $O(nk^2)$.

First, we observe without details that we can easily achieve this by starting instead with the algorithm of Eisner (2000),[20] rather than Eisner and Satta (1999), and again refusing to add long tree dependencies. That algorithm effectively concatenates only trapezoids, not triangles. Each is spanned by a single dependency and so has width $\leq k$. The vine dependencies do lead to wide trapezoids, but these are constrained to start at 0, where **$** is. So the algorithm tries at most $O(nk^2)$ combinations of the form  (like the ATTACH combinations above) and $O(nk)$ combinations of the form  , where $i - h \leq k, j - i \leq k$. The precise runtime is $O(nk(k + t')tg^3)$.

We now propose a hybrid linear-time algorithm that further improves runtime to $O(nk(k + t')tg^2)$, saving a factor of g in the grammar constant.[21] We observe that since within-tree dependencies must have length $\leq k$, they can all be captured within Eisner-Satta trapezoids of width $\leq k$. So our VG parse  can be assembled by simply *concatenating* a sequence  of these narrow trapezoids interspersed with width-0 triangles. As this is a *regular* sequence, we can assemble it in linear time from left to right (rather than in the order of Eisner and Satta (1999)), multiplying the items' probabilities together. Whenever we start adding the right half of a tree along the vine, we have discovered that tree's root, so we multiply in the probability of a **$** $\leftarrow$ root dependency.

Formally, our hybrid parsing algorithm restricts the original rules of Fig. 2a to build only trapezoids of width $\leq k$ and triangles of width $< k$.[22] The additional inference rules in Fig. 2b then assemble the final VG parse as just described.

[20] With a small change that when two items are combined, the *right* item (rather than the left) must be simple.

[21] This savings comes from building the internal structure of a trapezoid from both ends inward rather than from left to right. The corresponding unrestricted algorithms (Eisner, 2000; Eisner and Satta, 1999, respectively) have exactly the same runtimes with k replaced by n.

[22] For the experiments of §4.7, where k varied by type, we restricted these rules as tightly as possible given h and h'.

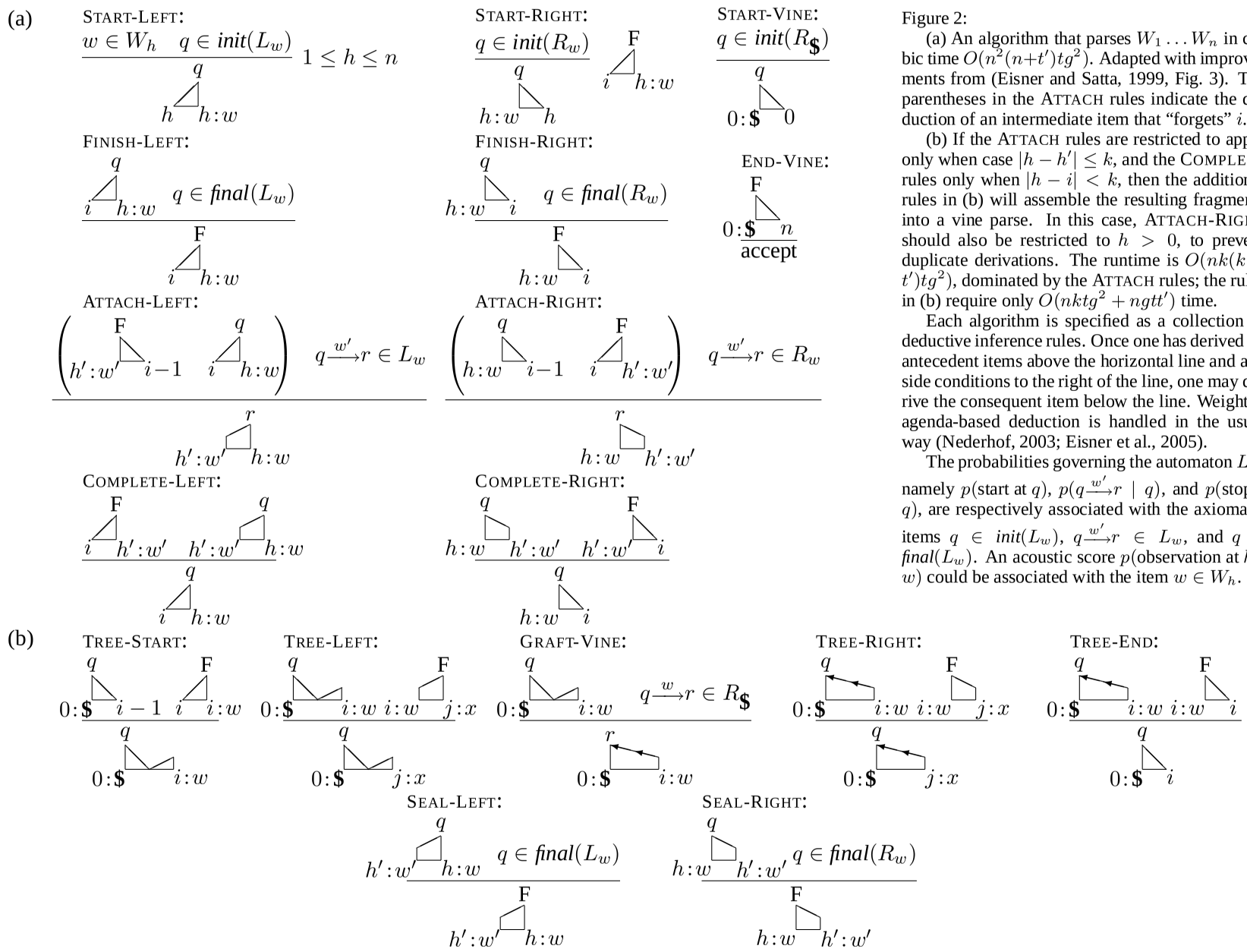


Figure 2:

(a) An algorithm that parses $W_1 \ldots W_n$ in cubic time $O(n^2(n+t')tg^2)$. Adapted with improvements from (Eisner and Satta, 1999, Fig. 3). The parentheses in the ATTACH rules indicate the deduction of an intermediate item that "forgets" i.

(b) If the ATTACH rules are restricted to apply only when case $|h - h'| \le k$, and the COMPLETE rules only when $|h - i| < k$, then the additional rules in (b) will assemble the resulting fragments into a vine parse. In this case, ATTACH-RIGHT should also be restricted to $h > 0$, to prevent duplicate derivations. The runtime is $O(nk(k + t')tg^2)$, dominated by the ATTACH rules; the rules in (b) require only $O(nktg^2 + ngtt')$ time.

Each algorithm is specified as a collection of deductive inference rules. Once one has derived all antecedent items above the horizontal line and any side conditions to the right of the line, one may derive the consequent item below the line. Weighted agenda-based deduction is handled in the usual way (Nederhof, 2003; Eisner et al., 2005).

The probabilities governing the automaton L_w, namely $p(\text{start at } q)$, $p(q \xrightarrow{w'} r \mid q)$, and $p(\text{stop} \mid q)$, are respectively associated with the axiomatic items $q \in \mathit{init}(L_w)$, $q \xrightarrow{w'} r \in L_w$, and $q \in \mathit{final}(L_w)$. An acoustic score $p(\text{observation at } h \mid w)$ could be associated with the item $w \in W_h$.

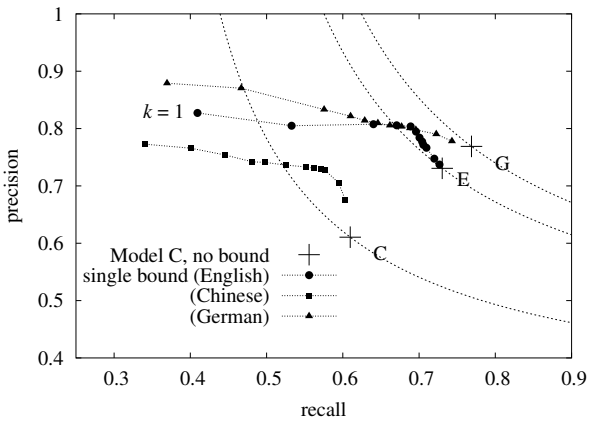


Figure 3: Trading precision and recall: Imposing bounds can improve precision at the expense of recall, for English and Chinese. German performance suffers more. Bounds shown are $k = \{1, 2, ..., 10, 15, 20\}$. The dotted lines show constant F-measure of the unbounded model.

4.6 Experiments with Hard Constraints

Our experiments used the asymptotically fast hybrid parsing algorithm above. We used the same left and right automata as in model C, the best-performing model from §3.2. However, we now define $R_{\$}$ to be a first-order (bigram) Markov model (§4.1). We trained and tested on the same headed treebanks as before (§3.7), except that we modified the *training* trees to make them feasible (§4.2).

Results are shown in Figures 3 (precision/recall tradeoff) and 4 (accuracy/speed tradeoff), for $k \in \{1, 2, ..., 10, 15, 20\}$. Dots correspond to different values of k. On English and Chinese, some values of k actually achieve better F-measure accuracy than the unbounded parser, by eliminating errors.[23]

We observed that changing $R_{\$}$ from a bigram to a unigram model significantly hurt performance, showing that it is in fact useful to empirically model likely *sequences* of parse fragments.

4.7 Finer-Grained Hard Constraints

The dependency length bound k need not be a single value. Substantially better accuracy can be retained if each dependency type—each $(h, c, d) =$ (head tag, child tag, direction) tuple—has its own bound $k(h, c, d)$. We call these *type-specific* bounds: they create a many-dimensional space of possible parsers. We measured speed and accuracy along a sensible path through this space, gradually tightening the bounds using the following process:

1. Initialize each bound $k(h, c, d)$ to the maximum distance observed in training (or 1 for unseen triples).[24]
2. Greedily choose a bound $k(h, c, d)$ such that, if its value is decremented and trees that violate the new bound are accordingly broken, the *fewest* dependencies will be broken.[25]
3. Decrement the bound $k(h, c, d)$ and modify the training data to respect the bound by breaking dependencies that violate the bound and "grafting" the loose portion onto the vine. Retrain the parser on the training data.
4. If all bounds are not equal to 1, go to step 2.

The performance of every 200th model along the trajectory of this search is plotted in Fig. 4.[26] The graph shows that type-specific bounds can speed up the parser to a given level with less loss in accuracy.

5 Related Work

As discussed in footnote 3, Collins (1997) and McDonald et al. (2005) considered the POS tags intervening between a head and child. These soft constraints were very helpful, perhaps in part because they helped capture the short dependency preference (§2). Collins used them as conditioning variables and McDonald et al. as log-linear features, whereas our §3 predicted them directly in a deficient model.

As for hard constraints (§4), our limitation on dependency length can be regarded as approximating a context-free language by a subset that is a regular

[23] Because our prototype implementation of each kind of parser (baseline, soft constraints, single-bound, and type-specific bounds) is known to suffer from different inefficiencies, runtimes in milliseconds are not comparable across parsers. To give a general idea, 60-word English sentences parsed in around 300ms with no bounds, but at around 200ms with either a distance model $p(\Delta|d, h, c)$ or a generous hard bound of $k = 10$.

[24] In the case of the German TIGER corpus, which contains non-projective dependencies, we first make the training trees into projective vines by raising all non-projective child nodes to become heads on the vine.

[25] Not counting dependencies that must be broken indirectly in order to maintain projectivity. (If word 4 depends on word 7 which depends on word 2, and the $4 \rightarrow 7$ dependency is broken, making 4 a root, then we must also break the $2 \rightarrow 7$ dependency.)

[26] Note that $k(h, c, \text{right}) = 7$ bounds the width of  +  = . For a finer-grained approach, we could instead separately bound the widths of and , say by $k_r(h, c, \text{right}) = 4$ and $k_l(h, c, \text{right}) = 2$.

language. Our "vines" then let us concatenate several strings in this subset, which typically yields a superset of the original context-free language. Subset and superset approximations of (weighted) CFLs by (weighted) regular languages, usually by preventing center-embedding, have been widely explored; Nederhof (2000) gives a thorough review. We limit *all* dependency lengths (not just center-embedding).[27] Further, we derive weights from a modified treebank rather than by approximating the true weights. And though regular grammar approximations are useful for other purposes, we argue that for parsing it is more efficient to perform the approximation in the parser, not in the grammar.

Brants (1999) described a parser that encoded the grammar as a set of cascaded Markov models. The decoder was applied iteratively, with each iteration transforming the best (or n-best) output from the previous one until only the root symbol remained. This is a greedy variant of CFG parsing where the grammar is in Backus-Naur form.

Bertsch and Nederhof (1999) gave a linear-time recognition algorithm for the recognition of the regular closure of deterministic context-free languages. Our result is related; instead of a closure of *deterministic* CFLs, we deal in a closure of CFLs that are assumed (by the parser) to obey some constraint on trees (like a maximum dependency length).

6 Future Work

The simple POS-sequence models we used as an experimental baseline are certainly not among the best parsers available today. They were chosen to illustrate how modeling and exploiting distance in syntax can affect various performance measures. Our approach may be helpful for other kinds of parsers as well. First, we hope that our results will generalize to more expressive grammar formalisms such as lexicalized CFG, CCG, and TAG, and to more expressively weighted grammars, such as log-linear models that can include head-child distance among other rich features. The parsing algorithms we presented also admit *inside-outside* variants, allowing iterative estimation methods for log-linear models (see, e.g., Miyao and Tsujii, 2002).

[27] Of course, this still allows right-branching or left-branching to unbounded depth.

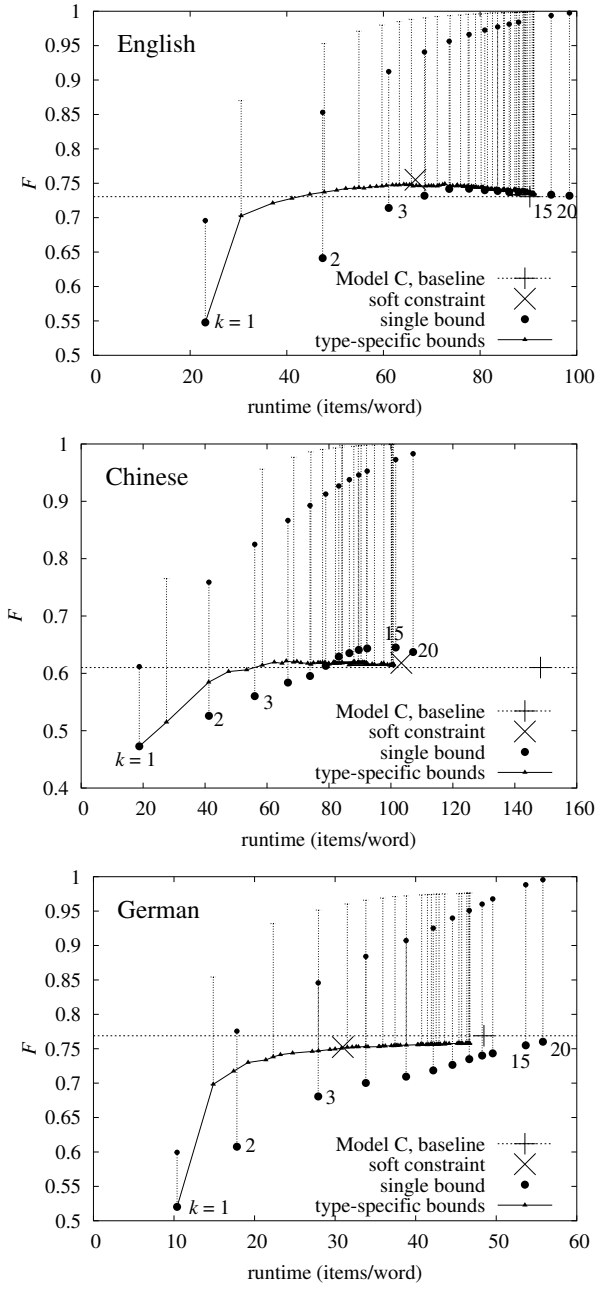


Figure 4: Trading off speed and accuracy by varying the set of feasible parses: The baseline (no length bound) is shown as $+$. Tighter bounds always improve speed, except for the most lax bounds, for which vine construction overhead incurs a slowdown. Type-specific bounds tend to maintain good F-measure at higher speeds than the single-bound approach. The vertical error bars show the "oracle" accuracy for each experiment (i.e., the F-measure if we had recovered the best feasible parse, as constructed from the gold-standard parse by grafting: see §4.2). Runtime is measured as the number of items per word (i.e., ) built by the agenda parser. The "soft constraint" point marked with $\times$ represents the $p(\Delta \mid d, h, c)$-augmented model from §3.

Second, fast approximate parsing may play a role in more accurate parsing. It might be used to rapidly compute approximate outside-probability estimates to prioritize best-first search (e.g., Caraballo and Charniak, 1998). It might also be used to speed up the early iterations of training a weighted parsing model, which for modern training methods tends to require repeated parsing (either for the best parse, as by Taskar et al., 2004, or all parses, as by Miyao and Tsujii, 2002).

Third, it would be useful to investigate algorithmic techniques and empirical benefits for limiting dependency length in more powerful grammar formalisms. Our runtime reduction from $O(n^3) \rightarrow O(nk^2)$ for a length-k bound applies only to a "split" bilexical grammar.[28] Various kinds of *synchronous* grammars, in particular, are becoming important in statistical machine translation. Their high runtime complexity might be reduced by limiting monolingual dependency length (for a related idea see Schafer and Yarowsky, 2003).

Finally, consider the possibility of limiting dependency length during grammar induction. We reason that a learner might start with simple structures that focus on local relationships, and gradually relax this restriction to allow more complex models.

7 Conclusion

We have described a novel reason for identifying headword-to-headword dependencies while parsing: to consider their length. We have demonstrated that simple bilexical parsers of English, Chinese, and German can exploit a "short-dependency preference." Notably, *soft* constraints on dependency length can improve both speed and accuracy, and *hard* constraints allow improved precision and speed with some loss in recall (on English and Chinese, remarkably little loss). Further, for the hard constraint "length $\leq k$," we have given an $O(nk^2)$ partial parsing algorithm for split bilexical grammars; the grammar constant is no worse than for state-of-the-art $O(n^3)$ algorithms. This algorithm strings together the partial trees' roots along a "vine."

[28] The obvious reduction for unsplit head automaton grammars, say, is only $O(n^4) \rightarrow O(n^3k)$, following (Eisner and Satta, 1999). Alternatively, one can convert the unsplit HAG to a split one that preserves the set of feasible (length $\leq k$) parses, but then g becomes prohibitively large in the worst case.

Our approach might be adapted to richer parsing formalisms, including synchronous ones, and should be helpful as an approximation to full parsing when fast, high-precision recovery of syntactic information is needed.

References

S. P. Abney. Parsing by chunks. In *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer, 1991.

D. E. Appelt, J. R. Hobbs, J. Bear, D. Israel, and M. Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *Proc. of IJCAI*, 1993.

E. Bertsch and M.-J. Nederhof. Regular closure of deterministic languages. *SIAM J. on Computing*, 29(1):81–102, 1999.

D. Bikel. A distributional analysis of a lexicalized statistical parsing model. In *Proc. of EMNLP*, 2004.

T. Brants. Cascaded Markov models. In *Proc. of EACL*, 1999.

S. A. Caraballo and E. Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–98, 1998.

S. Chen. Bayesian grammar induction for language modeling. In *Proc. of ACL*, 1995.

K. W. Church. On memory limitations in natural language processing. Master's thesis, MIT, 1980.

M. Collins. Three generative, lexicalised models for statistical parsing. In *Proc. of ACL*, 1997.

J. Eisner. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*. Kluwer, 2000.

J. Eisner, E. Goldlust, and N. A. Smith. Compiling Comp Ling: Practical weighted dynamic programming and the Dyna language. In *Proc. of HLT-EMNLP*, 2005.

J. Eisner and G. Satta. Efficient parsing for bilexical cfgs and head automaton grammars. In *Proc. of ACL*, 1999.

L. Frazier. *On Comprehending Sentences: Syntactic Parsing Strategies*. PhD thesis, University of Massachusetts, 1979.

E. Gibson. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68:1–76, 1998.

G. Grefenstette. Light parsing as finite-state filtering. In *Proc. of Workshop on Extended FS Models of Language*, 1996.

D. Hindle. Noun classification from predicate-argument structure. In *Proc. of ACL*, 1990.

J. R. Hobbs and J. Bear. Two principles of parse preference. In *Proc. of COLING*, 1990.

D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proc. of ACL*, 2003.

D. Klein and C. D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. of ACL*, 2004.

R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proc. of ACL*, 2005.

Y. Miyao and J. Tsujii. Maximum entropy estimation for feature forests. In *Proc. of HLT*, 2002.

M.-J. Nederhof. Practical experiments with regular approximation of context-free languages. *CL*, 26(1):17–44, 2000.

M.-J. Nederhof. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143, 2003.

C. Schafer and D. Yarowsky. A two-level syntax-based approach to Arabic-English statistical machine translation. In *Proc. of Workshop on MT for Semitic Languages*, 2003.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proc. of EMNLP*, 2004.

Corrective Modeling for Non-Projective Dependency Parsing

Keith Hall
Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218
keith_hall@jhu.edu

Václav Novák
Institute of Formal and Applied Linguistics
Charles University
Prague, Czech Republic
novak@ufal.mff.cuni.cz

Abstract

We present a corrective model for recovering non-projective dependency structures from trees generated by state-of-the-art constituency-based parsers. The continuity constraint of these constituency-based parsers makes it impossible for them to posit non-projective dependency trees. Analysis of the types of dependency errors made by these parsers on a Czech corpus show that the correct governor is likely to be found within a local neighborhood of the governor proposed by the parser. Our model, based on a MaxEnt classifier, improves overall dependency accuracy by .7% (a 4.5% reduction in error) with over 50% accuracy for non-projective structures.

1 Introduction

Statistical parsing models have been shown to be successful in recovering labeled constituencies (Collins, 2003; Charniak and Johnson, 2005; Roark and Collins, 2004) and have also been shown to be adequate in recovering dependency relationships (Collins et al., 1999; Levy and Manning, 2004; Dubey and Keller, 2003). The most successful models are based on lexicalized probabilistic context free grammars (PCFGs) induced from constituency-based treebanks. The linear-precedence constraint of these grammars restricts the types of dependency structures that can be encoded in such trees.[1] A shortcoming of the constituency-based paradigm for parsing is that it is inherently incapable of representing non-projective dependencies trees (we define non-projectivity in the following section). This is particularly problematic when parsing free word-order languages, such as Czech, due to the frequency of sentences with non-projective constructions.

In this work, we explore a corrective model which recovers non-projective dependency structures by training a classifier to select correct dependency pairs from a set of candidates based on parses generated by a constituency-based parser. We chose to use this model due to the observations that the dependency errors made by the parsers are generally local errors. For the nodes with incorrect dependency links in the parser output, the correct governor of a node is often found within a local context of the proposed governor. By considering alternative dependencies based on local deviations of the parser output we constrain the set of candidate governors for each node during the corrective procedure. We examine two state-of-the-art constituency-based parsers in this work: the Collins Czech parser (1999) and a version of the Charniak parser (2001) that was modified to parse Czech.

Alternative efforts to recover dependency structure from English are based on reconstructing the movement *traces* encoded in constituency trees (Collins, 2003; Levy and Manning, 2004; Johnson, 2002; Dubey and Keller, 2003). In fact, the fea-

[1] In order to correctly capture the dependency structure, co-indexed movement *traces* are used in a form similar to government and Binding theory, GPSG, etc.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 42–52,
Vancouver, October 2005.

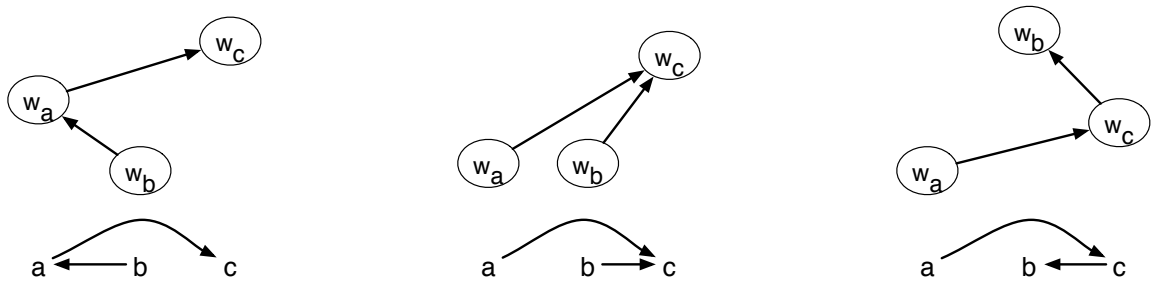


Figure 1: Examples of projective and non-projective trees. The trees on the left and center are both projective. The tree on the right is non-projective.

tures we use in the current model are similar to those proposed by Levy and Manning (2004). However, the approach we propose discards the constituency structure prior to the modeling phase; we model corrective transformations of dependency trees.

The technique proposed in this paper is similar to that of recent parser reranking approaches (Collins, 2000; Charniak and Johnson, 2005); however, while reranking approaches allow a parser to generate a likely candidate set according to a generative model, we consider a set of candidates based on local perturbations of the single most likely tree generated. The primary reason for such an approach is that we allow dependency structures which would never be hypothesized by the parser. Specifically, we allow for non-projective dependencies.

The corrective algorithm proposed in this paper shares the motivation of the transformation-based learning work (Brill, 1995). We do consider local transformations of the dependency trees; however, the technique presented here is based on a generative model that maximizes the likelihood of good dependents. We consider a finite set of local perturbations of the tree and use a fixed model to select the best tree by independently choosing optimal dependency links.

In the remainder of the paper we provide a definition of a dependency tree and the motivation for using such trees as well as a description of the particular dataset that we use in our experiments, the Prague Dependency Treebank (PDT). In Section 3 we describe the techniques used to adapt constituency-based parsers to train from and generate dependency trees. Section 4 describes corrective modeling as used in this work and Section 4.2 describes the particular features with which we have experimented. Section 5 presents the results of a set of experiments we performed on data from the PDT.

2 Syntactic Dependency Trees and the Prague Dependency Treebank

A dependency tree is a set of nodes $\Omega = \{w_0, w_1, \ldots, w_k\}$ where w_0 is the imaginary root node[2] and a set of dependency links $G = \{g_1, \ldots, g_k\}$ where g_i is an index into Ω representing the governor of w_i. In other words $g_3 = 1$ indicates that the governor of w_3 is w_1. Finally, every node has exactly one governor except for w_0, which has no governor (the tree constraints).[3] The index of the nodes represents the surface order of the nodes in the sequence (i.e., w_i precedes w_j in the sentence if $i < j$).

A tree is *projective* if for every three nodes: w_a, w_b, and w_c where $a < b < c$; if w_a is governed by w_c then w_b is transitively governed by w_c or if w_c is governed by w_a then w_b is transitively governed by w_a.[4] Figure 1 shows examples of projective and non-projective trees. The rightmost tree, which is non-projective, contains a subtree consisting of w_a and w_c but not w_b; however, w_b occurs between w_a and w_c in the linear ordering of the nodes. Projectivity in a dependency tree is akin to the continuity constraint in a constituency tree; such a constraint is

[2]The imaginary root node simplifies notation.

[3]The dependency structures here are very similar to those described by Mel'čuk (1988); however the nodes of the dependency trees discussed in this paper are limited to the words of the sentence and are always ordered according to the surface word-order.

[4]Node w_a is said to transitively govern node w_b if w_b is a descendant of w_a in the dependency tree.

implicitly imposed by trees generated from context free grammars (CFGs).

Strict word-order languages, such as English, exhibit non-projective dependency structures in a relatively constrained set of syntactic configurations (e.g., right-node raising). Traditionally, these movements are encoded in syntactic analyses as *traces*. In languages with free word-order, such as Czech, constituency-based representations are overly constrained (Sgall et al., 1986). Syntactic dependency trees encode syntactic subordination relationships allowing the structure to be non-specific about the *underlying deep representation*. The relationship between a node and its subordinates expresses a sense of syntactic (functional) entailment.

In this work we explore the dependency structures encoded in the Prague Dependency Treebank (Hajič, 1998; Böhmová et al., 2002). The PDT 1.0 analytical layer is a set of Czech syntactic dependency trees; the nodes of which contain the word forms, morphological features, and syntactic annotations. These trees were annotated by hand and are intended as an intermediate stage in the annotation of the Tectogrammatical Representation (TR), a *deep-syntactic* or syntacto-semantic theory of language (Sgall et al., 1986). All current automatic techniques for generating TR structures are based on syntactic dependency parsing.

When evaluating the correctness of dependency trees, we only consider the structural relationships between the words of the sentence (unlabeled dependencies). However, the model we propose contains features that are considered part of the dependency rather than the nodes in isolation (e.g., agreement features). We do not propose a model for correctly labeling dependency structures in this work.

3 Constituency Parsing for Dependency Trees

A pragmatic justification for using constituency-based parsers in order to predict dependency structures is that currently the best Czech dependency-tree parser is a constituency-based parser (Collins et al., 1999; Zeman, 2004). In fact both Charniak's and Collins' generative probabilistic models contain lexical dependency features.[5] From a generative modeling perspective, we use the constraints imposed by constituents (i.e., projectivity) to enable the encapsulation of syntactic substructures. This directly leads to efficient parsing algorithms such as the CKY algorithm and related agenda-based parsing algorithms (Manning and Schütze, 1999). Additionally, this allows for the efficient computation of the scores for the dynamic-programming state variables (i.e., the inside and outside probabilities) that are used in efficient statistical parsers. The computational complexity advantages of dynamic programming techniques along with efficient search techniques (Caraballo and Charniak, 1998; Klein and Manning, 2003) allow for richer predictive models which include local contextual information.

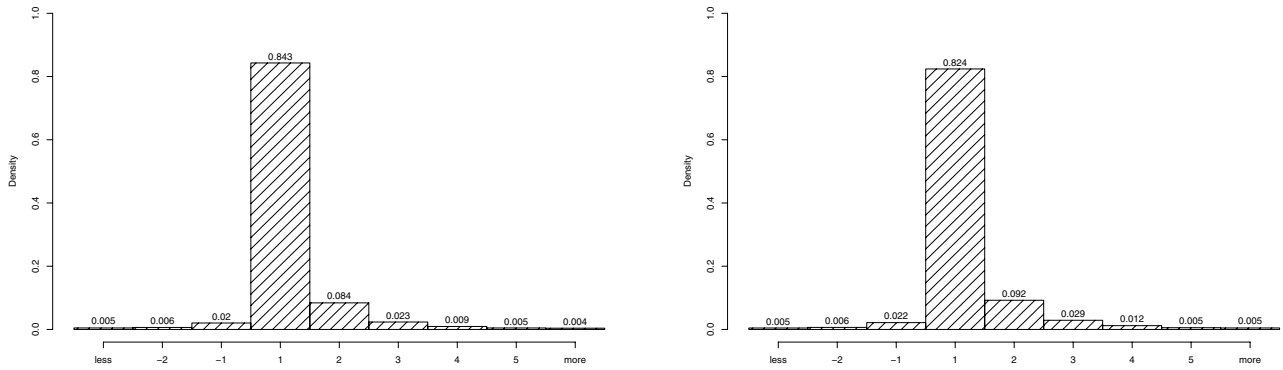
In an attempt to extend a constituency-based parsing model to train on dependency trees, Collins transforms the PDT dependency trees into constituency trees (Collins et al., 1999). In order to accomplish this task, he first normalizes the trees to remove non-projectivities. Then, he creates artificial constituents based on the parts-of-speech of the words associated with each dependency node. The mapping from dependency tree to constituency tree is not one-to-one. Collins describes a heuristic for choosing trees that work well with his parsing model.

3.1 Training a Constituency-based Parser

We consider two approaches to creating projective trees from dependency trees exhibiting non-projectivities. The first is based on word-reordering and is the model that was used with the Collins parser. This algorithm identifies non-projective structures and deterministically reorders the words of the sentence to create projective trees. An alternative method, used by Charniak in the adaptation of his parser for Czech[6] and used by Nivre and Nilsson (2005), alters the dependency links by raising the governor to a higher node in the tree whenever

[5] Bilexical dependencies are components of both the Collins and Charniak parsers and effectively model the types of syntactic subordination that we wish to extract in a dependency tree. (Bilexical models were also proposed by Eisner (Eisner, 1996)). In the absence of lexicalization, both parsers have dependency features that are encoded as head-constituent to sibling features.

[6] This information was provided by Eugene Charniak in a personal communication.



$(a) Charniak$ $(b) Collins$

Figure 2: Statistical distribution of correct governor positions in the Charniak (left) and Collins (right) parser output of parsed PDT development data.

a non-projectivity is observed. The trees are then transformed into Penn Treebank style constituencies using the technique described in (Collins et al., 1999).

Both of these techniques have advantages and disadvantages which we briefly outline here:

Reordering The dependency structure is preserved, but the training procedure will learn statistics for structures over word-strings that may not be part of the language. The parser, however, may be capable of constructing parses for any string of words if a smoothed grammar is being used.

Governor–Raising The dependency structure is corrupted leading the parser to incorporate arbitrary dependency statistics into the model. However, the parser is trained on true sentences, the words of which are in the correct linear order. We expect the parser to predict similar incorrect dependencies when sentences similar to the training data are observed.

Although the results presented in (Collins et al., 1999) used the reordering technique, we have experimented with his parser using the governor–raising technique and observe an increase in dependency accuracy. For the remainder of the paper, we assume the governor–raising technique.

The process of generating dependency trees from parsed constituency trees is relatively straightforward. Both the Collins and Charniak parsers provide head-word annotation on each constituent. This is precisely the information that we encode in an unlabeled dependency tree, so the dependency structure can simply be extracted from the parsed constituency trees. Furthermore, the constituency labels can be used to identify the dependency labels; however, we do not attempt to identify correct dependency labels in this work.

3.2 Constituency-based errors

We now discuss a quantitative measure for the types of dependency errors made by constituency-based parsing techniques. For node w_i and the correct governor $w_{g_i^*}$ the *distance* between the two nodes in the hypothesized dependency tree is:

$$dist(w_i, w_{g_i^*}) = \begin{cases} d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is ancestor of } w_i \\ d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is sibling/cousin of } w_i \\ -d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is descendant of } w_i \end{cases}$$

Ancestor, sibling, cousin, and descendant have the standard interpretation in the context of a tree. The dependency distance $d(w_i, w_{g_i^*})$ is the minimum number of dependency links traversed on the undirected path from w_i to $w_{g_i^*}$ in the hypothesized dependency tree. The definition of the *dist* function makes a distinction between paths through the parent of w_i (positive values) and paths through chil-

CORRECT(W)

1 Parse sentence W using the constituency-based parser
2 Generate a dependency structure from the constituency tree
3 **for** $w_i \in W$
4 **do for** $w_c \in \mathcal{N}(w_{g_i^h})$ *// Local neighborhood of proposed governor*
5 **do** $l(c) \leftarrow P(g_i^* = c | w_i, \mathcal{N}(w_{g_i^h}))$
6 $g_i^{'} \leftarrow \arg\max_c l(c)$ *// Pick the governor in which we are most confident*

Table 1: Corrective Modeling Procedure

dren of w_i (negative values). We found that a vast majority of the correct governors were actually hypothesized as siblings or grandparents (a *dist* values of 2) – an extreme local error.

Figure 2 shows a histogram of the fraction of nodes whose correct governor was within a particular *dist* in the hypothesized tree. A *dist* of 1 indicates the correct governor was selected by the parser; in these graphs, the density at *dist* $= 1$ (on the x axis) shows the baseline dependency accuracy of each parser. Note that if we repaired only the nodes that are within a *dist* of 2 (grandparents and siblings), we can recover more than 50% of the incorrect dependency links (a raw accuracy improvement of up to 9%). We believe this distribution to be indirectly caused by the governor raising projectivization routine. In the cases where non-projective structures can be repaired by raising the node's governor to its parent, the correct governor becomes a sibling of the node.

4 Corrective Modeling

The error analysis of the previous section suggests that by looking only at a local neighborhood of the proposed governor in the hypothesized trees, we can correct many of the incorrect dependencies. This fact motivates the corrective modeling procedure employed here.

Table 1 presents the pseudo-code for the corrective procedure. The set g^h contains the indices of governors as predicted by the parser. The set of governors predicted by the corrective procedure is denoted as $g^{'}$. The procedure independently corrects each node of the parsed trees meaning that there is potential for inconsistent governor relationships to exist in the proposed set; specifically, the resulting dependency graph may have cycles. We employ a greedy search to remove cycles when they are present in the output graph.

The final line of the algorithm picks the governor in which we are most confident. We use the correct-governor classification likelihood, $P(g_i^* = j | w_i, \mathcal{N}(w_{g_i^h}))$, as a measure of the confidence that w_c is the correct governor of w_i where the parser had proposed $w_{g_i^h}$ as the governor. In effect, we create a decision list using the most likely decision if we can (i.e., there are no cycles). If the dependency graph resulting from the most likely decisions does not result in a tree, we use the decision lists to greedily select the tree for which the product of the independent decisions is maximal.

Training the corrective model requires pairs of dependency trees; each pair contains a manually-annotated tree (i.e., the gold standard tree) and a tree generated by the parser. This data is trivially transformed into per-node samples. For each node w_i in the tree, there are $|\mathcal{N}(w_{g_i^h})|$ samples; one for each governor candidate in the local neighborhood.

One advantage to the type of corrective algorithm presented here is that it is completely disconnected from the parser used to generate the tree hypotheses. This means that the original parser need not be statistical or even constituency based. What is critical for this technique to work is that the distribution of dependency errors be relatively local as is the case with the errors made by the Charniak and Collins parsers. This can be determined via data analysis using the *dist* metric. Determining the size of the local neighborhood is data dependent. If subordinate nodes are considered as candidate governors, then a more robust cycle removal technique is be required.

4.1 MaxEnt Estimation

We have chosen a MaxEnt model to estimate the governor distributions, $P(g_i^* = j|w_i, \mathcal{N}(w_{g_i^h}))$. In the next section we outline the feature set with which we have experimented, noting that the features are selected based on linguistic intuition (specifically for Czech). We choose not to factor the feature vector as it is not clear what constitutes a reasonable factorization of these features. For this reason we use the MaxEnt estimator which provides us with the flexibility to incorporate interdependent features independently while still optimizing for likelihood.

The maximum entropy principle states that we wish to find an estimate of $p(y|x) \in \mathcal{C}$ that maximizes the entropy over a sample set X for some set of observations Y, where $x \in X$ is an observation and $y \in Y$ is a outcome label assigned to that observation,

$$H(p) \equiv -\sum_{x \in X, y \in Y} \tilde{p}(x) p(y|x) \log p(y|x)$$

The set $\mathcal{C}$ is the candidate set of distributions from which we wish to select $p(y|x)$. We define this set as the $p(y|x)$ that meets a feature-based expectation constraint. Specifically, we want the expected count of a feature, $f(x, y)$, to be equivalent under the distribution $p(y|x)$ and under the observed distribution $\tilde{p}(y|x)$.

$$\sum_{x \in X, y \in Y} \tilde{p}(x) p(y|x) f_i(x, y) = \sum_{x \in X, y \in Y} \tilde{p}(x) \tilde{p}(y|x) f_i(x, y)$$

$f_i(x, y)$ is a feature of our model with which we capture correlations between observations and outcomes. In the following section, we describe a set of features with which we have experimented to determine when a word is likely to be the correct governor of another word.

We incorporate the expected feature-count constraints into the maximum entropy objective using Lagrange multipliers (additionally, constraints are added to ensure the distributions $p(y|x)$ are consistent probability distributions):

$$\begin{aligned} & H(p) \\ & + \sum_i \alpha_i \sum_{x \in X, y \in Y} \Big(\tilde{p}(x) p(y|x) f_i(x, y) \\ & \qquad - \tilde{p}(x) \tilde{p}(y|x) f_i(x, y) \Big) \\ & + \gamma \sum_{y \in Y} p(y|x) - 1 \end{aligned}$$

Holding the α_i's constant, we compute the unconstrained maximum of the above Lagrangian form:

$$p_\alpha(y|x) = \frac{1}{Z_\alpha(x)} \exp\Big(\sum_i \alpha_i f_i(x, y)\Big)$$
$$Z_\alpha(x) = \sum_{y \in Y} \exp\Big(\sum_i \alpha_i f_i(x, y)\Big)$$

giving us the log-linear form of the distributions $p(y|x)$ in $\mathcal{C}$ (Z is a normalization constant). Finally, we compute the α_i's that maximize the objective function:

$$-\sum_{x \in X} \tilde{p}(x) \log Z_\alpha(x) + \sum_i \alpha_i \tilde{p}(x, y) f_i(x, y)$$

A number of algorithms have been proposed to efficiently compute the optimization described in this derivation. For a more detailed introduction to maximum entropy estimation see (Berger et al., 1996).

4.2 Proposed Model

Given the above formulation of the MaxEnt estimation procedure, we define features over pairs of observations and outcomes. In our case, the observations are simply w_i, w_c, and $\mathcal{N}(w_{g_i^h})$ and the outcome is a binary variable indicating whether $c = g_i^*$ (i.e., w_c is the correct governor). In order to limit the dimensionality of the feature space, we consider feature functions over the outcome, the current node w_i, the candidate governor node w_c and the node proposed as the governor by the parser $w_{g_i^h}$.

Table 2 describes the general classes of features used. We write F_i to indicate the form of the current child node, F_c for the form of the candidate, and F_g as the form of the governor proposed by the parser. A combined feature is denoted as $L_i T_c$ and indicates we observed a particular lemma for the current node with a particular tag of the candidate.

Feature Type	Id	Description
Form	F	the fully inflected word form as it appears in the data
Lemma	L	the morphologically reduced lemma
MTag	T	a subset of the morphological tag as described in (Collins et al., 1999)
POS	P	major part-of-speech tag (first field of the morphological tag)
ParserGov	G	true if candidate was proposed as governor by parser
ChildCount	C	the number of children
Agreement	$A(x, y)$	check for case/number agreement between word x and y

Table 2: Description of the classes of features used

In all models, we include features containing the form, the lemma, the morphological tag, and the ParserGov feature. We have experimented with different sets of feature combinations. Each combination set is intended to capture some intuitive linguistic correlation. For example, the feature component L_iT_c will fire if a particular child's lemma L_i is observed with a particular candidate's morphological tag T_c. This feature is intended to capture phenomena surrounding particles; for example, in Czech, the governor of the reflexive particle *se* will likely be a verb.

4.3 Related Work

Recent work by Nivre and Nilsson introduces a technique where the projectivization transformation is encoded in the non-terminals of constituents during parsing (Nivre and Nilsson, 2005). This allows for a deterministic procedure that undoes the projectivization in the generated parse trees, creating non-projective structures. This technique could be incorporated into a statistical parsing framework, however we believe the sparsity of such non-projective configurations may be problematic when using smoothed backed-off grammars. We suspect that the deterministic procedure employed by Nivre and Nilsson enables their parser to greedily consider non-projective constructions when possible. This may also explain the relatively low overall performance of their parser.

A primary difference between the Nivre and Nilsson approach and what we propose in this paper is that of determining the projectivization procedure. While we exploit particular side-effects of the projectivization procedure, we do not assume any particular algorithm. Additionally, we consider transformations for all dependency errors where their technique explicitly addresses non-projectivity errors.

We mentioned above that our approach appears to be similar to that of reranking for statistical parsing (Collins, 2000; Charniak and Johnson, 2005). While it is true that we are improving upon the output of the automatic parser, we are not considering multiple alternate parses. Instead, we consider a complete set of alternate trees that are minimal perturbations of the best tree generated by the parser. In the context of dependency parsing, we do this in order to generate structures that constituency-based parsers are incapable of generating (i.e., non-projectivities).

Recent work by Smith and Eisner (2005) on *contrastive estimation* suggests similar techniques to generate local neighborhoods of a parse; however, the purpose in their work is to define an approximation to the partition function for log-linear estimation (i.e., the normalization factor in a MaxEnt model).

5 Empirical Results

In this section we report results from experiments on the PDT Czech dataset. Approximately 1.9% of the words' dependencies are non-projective in version 1.0 of this corpus and these occur in 23.2% of the sentences (Hajičová et al., 2004). We used the standard training, development, and evaluation datasets defined in the PDT documentation for all experiments.[7] We use Zhang Lee's implementation of the

[7]We have used PDT 1.0 (2002) data for the Charniak experiments and PDT 2.0 (2005) data for the Collins experiments. We use the most recent version of each parser; however we do not have a training program for the Charniak parser and have used the pretrained parser provided by Charniak; this was trained on the training section of the PDT 1.0. We train our model on the

Model	Features	Description
Count	ChildCount	count of children for the three nodes
MTagL	$T_iT_c, L_iL_c, L_iT_c, T_iL_c, T_iP_g$	conjunctions of MTag and Lemmas
MTagF	$T_iT_c, F_iF_c, F_iT_c, T_iF_c, T_iP_g$	conjunctions of MTag and Forms
POSL	$P_i, P_c, P_g, P_iP_cP_g, P_iP_g, P_cL_c$	conjunctions of POS and Lemma
TTT	$T_iT_cT_g$	conjunction of tags for each of the three nodes
Agr	$A(T_i, T_c), A(T_i, T_g)$	binary feature if case/number agree
Trig	$L_iL_gT_c, T_iL_gT_c, L_iL_gL_c$	trigrams of Lemma/Tag

Table 3: Model feature descriptions.

Model	Charniak Parse Trees		Collins Parse Trees	
	Devel. Accuracy	NonP Accuracy	Devel. Accuracy	NonP Accuracy
Baseline	84.3%	15.9%	82.4%	12.0%
Simple	84.3%	16.0%	82.5%	12.2%
Simple + Count	84.3%	16.7%	82.5%	13.8%
Simple + MtagL	84.8%	43.5%	83.2%	44.1%
Simple + MtagF	84.8%	42.2%	83.2%	43.2%
Simple + POS	84.3%	16.0%	82.4%	12.1%
Simple + TTT	84.3%	16.0%	82.5%	12.2%
Simple + Agr	84.3%	16.2%	82.5%	12.2%
Simple + Trig	84.9%	47.9%	83.1%	47.7%
All Features	**85.0%**	**51.9%**	**83.5%**	**57.5%**

Table 4: Comparative results for different versions of our model on the Charniak and Collins parse trees for the PDT development data.

MaxEnt estimator using the L-BFGS optimization algorithms and Gaussian smoothing.[8]

Table 4 presents results on development data for the correction model with different feature sets. The features of the **Simple** model are the form (F), lemma (L), and morphological tag (M) for the each node, the parser-proposed governor node, and the candidate node; this model also contains the Parser-Gov feature. In the table's following rows, we show the results for the simple model augmented with feature sets of the categories described in Table 2. Table 3 provides a short description of each of the models. As we believe the **Simple** model provides the minimum information needed to perform this task, we experimented with the feature-classes as additions to it. The final row of Table 4 contains results for the model which includes all features from all other models.

We define **NonP Accuracy** as the accuracy for the nodes which were non-projective in the original trees. Although both the Charniak and the Collins parser can never produce non-projective trees, the baseline NonP accuracy is greater than zero. This is due to the parser making mistakes in the tree such that the originally non-projective node's dependency is projective.

Alternatively, we report the Non-Projective Precision and Recall for our experiment suite in Table 5. Here the numerator of the precision is the number of nodes that are non-projective in the correct tree and end up in a non-projective configuration; however, this new configuration may be based on incorrect dependencies. Recall is the obvious counterpart to precision. These values correspond to the NonP

Collins trees via a 20-fold Jackknife training procedure.

[8]Using held-out development data, we determined a Gaussian prior parameter setting of 4 worked best. The optimal number of training iterations was chosen on held-out data for each experiment. This was generally in the order of a couple hundred iterations of L-BFGS. The MaxEnt modeling implementation can be found at `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`.

Model	Charniak Parse Trees			Collins Parse Trees		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Baseline	N/A	0.0%	0.000	N/A	0.0%	0.000
Simple	22.6%	0.3%	0.592	5.0%	0.2%	0.385
Simple + Count	37.3%	1.1%	2.137	16.8%	2.0%	3.574
Simple + MtagL	78.0%	29.7%	43.020	62.4%	35.0%	44.846
Simple + MtagF	78.7%	28.6%	41.953	62.0%	34.3%	44.166
Simple + POS	23.3%	0.3%	0.592	2.5%	0.1%	0.192
Simple + TTT	20.7%	0.3%	0.591	6.1%	0.2%	0.387
Simple + Agr	40.0%	0.5%	0.988	5.7%	0.2%	0.386
Simple + Trig	74.6%	35.0%	47.646	52.3%	40.2%	45.459
All Features	75.7%	39.0%	51.479	48.1%	51.6%	49.789

Table 5: Alternative non-projectivity scores for different versions of our model on the Charniak and Collins parse trees.

accuracy results reported in Table 4. From these tables, we see that the most effective features (when used in isolation) are the conjunctive MTag/Lemma, MTag/Form, and Trigram MTag/Lemma features.

Model	Dependency Accuracy	NonP Accuracy
Collins	81.6%	N/A
Collins + Corrective	**82.8%**	**53.1**%
Charniak	84.4%	N/A
Charniak + Corrective	**85.1%**	**53.9**%

Table 6: Final results on PDT evaluation datasets for Collins' and Charniak's trees with and without the corrective model

Finally, Table 6 shows the results of the full model run on the evaluation data for the Collins and Charniak parse trees. It appears that the Charniak parser fares better on the evaluation data than does the Collins parser. However, the corrective model is still successful at recovering non-projective structures. Overall, we see a significant improvement in the dependency accuracy.

We have performed a review of the errors that the corrective process makes and observed that the model does a poor job dealing with punctuation. This is shown in Table 7 along with other types of nodes on which we performed well and poorly, respectively. Collins (1999) explicitly added features to his parser to improve punctuation dependency parsing accuracy. The PARSEVAL evaluation metric for constituency-based parsing explicitly ignores punctuation in determining the correct boundaries of constituents (Harrison et al., 1991) and so should the dependency evaluation. However, the reported results include punctuation for comparative purposes. Finally, we show in Table 8 a coarse analysis of the corrective performance of our model. We are repair-

Top Five Good/Bad Repairs	
Well repaired child	se i si až jen
Well repaired false governor	v však li na o
Well repaired real governor	a je stát ba ,
Poorly repaired child	, se na že -
Poorly repaired false governor	a , však musí li
Poorly repaired real governor	*root* sklo , je -

Table 7: Categorization of corrections and errors made by our model on trees from the Charniak parser. *root* is the artificial root node of the PDT tree. For each node position (child, proposed parent, and correct parent), the top five words are reported (based on absolute count of occurrences). The particle 'se' occurs frequently explaining why it occurs in the top five good and top five bad repairs.

	Charniak	Collins
Correct to incorrect	13.0%	20.0%
Incorrect to incorrect	21.6%	25.8%
Incorrect to correct	65.5%	54.1%

Table 8: Categorization of corrections made by our model on Charniak and Collins trees.

ing more dependencies than we are corrupting.

6 Conclusion

We have presented a Maximum Entropy-based corrective model for dependency parsing. The goal is to recover non-projective dependency structures that are lost when using state-of-the-art constituency-based parsers; we show that our technique recovers over 50% of these dependencies. Our algorithm provides a simple framework for corrective modeling of dependency trees, making no prior assumptions about the trees. However, in the current model, we focus on trees with local errors. Overall, our technique improves dependency parsing and provides the necessary mechanism to recover non-projective structures.

References

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Vidová Hladká. 2002. The prague dependency treebank: Three-level annotation scenario. In Anne Abeille, editor, *In Treebanks: Building and Using Syntactically Annotated Corpora*. Dordrecht, Kluwer Academic Publishers, The Neterlands.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, December.

Sharon Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, June.

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*.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*.

Michael Collins, Lance Ramshaw, Jan Hajič, and Christoph Tillmann. 1999. A statistical parser for czech. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, pages 505–512.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning 2000*.

Michael Collins. 2003. Head-driven statistical models for natural language processing. *Computational Linguistics*, 29(4):589–637.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

Jan Hajič, Eva Hajičová, Petr Pajas, Jarmila Panevová, Petr Sgall, and Barbora Vidová Hladká. 2005. The prague dependency treebank 2.0. http://ufal.mff.cuni.cz/pdt2.0.

Jan Hajič. 1998. Building a syntactically annotated corpus: The prague dependency treebank. In *Issues of Valency and Meaning*, pages 106–132. Karolinum, Praha.

Eva Hajičová, Jiří Havelka, Petr Sgall, Kateřina Veselá, and Daniel Zeman. 2004. Issues of projectivity in the prague dependency treebank. *Prague Bulletin of Mathematical Linguistics*, 81:5–22.

P. Harrison, S. Abney, D. Fleckenger, C. Gdaniec, R. Grishman, D. Hindle, B. Ingria, M. Marcus, B. Santorini, , and T. Strzalkowski. 1991. Evaluating syntax performance of parser/grammars of english. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems, ACL*.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

Dan Klein and Christopher D. Manning. 2003. Factored A* search for models over sequences and trees. In *Proceedings of IJCAI 2003*.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 327–334, Barcelona, Spain.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT Press.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. SUNY Press, Albany, NY.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 99–106, Ann Arbor.

Brian Roark and Michael Collins. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Kluwer Academic, Boston.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the Association for Computational Linguistics (ACL 2005)*, Ann Arbor, Michigan.

Daniel Zeman. 2004. *Parsing with a statistical dependency model*. Ph.D. thesis, Charles University in Prague.

Better k-best Parsing

Liang Huang
Dept. of Computer & Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104
lhuang3@cis.upenn.edu

David Chiang
Inst. for Advanced Computer Studies
University of Maryland
3161 AV Williams
College Park, MD 20742
dchiang@umiacs.umd.edu

Abstract

We discuss the relevance of *k-best parsing* to recent applications in natural language processing, and develop efficient algorithms for k-best trees in the framework of hypergraph parsing. To demonstrate the efficiency, scalability and accuracy of these algorithms, we present experiments on Bikel's implementation of Collins' lexicalized PCFG model, and on Chiang's CFG-based decoder for hierarchical phrase-based translation. We show in particular how the improved output of our algorithms has the potential to improve results from parse reranking systems and other applications.

1 Introduction

Many problems in natural language processing (NLP) involve optimizing some objective function over a set of possible analyses of an input string. This set is often exponential-sized but can be compactly represented by merging equivalent subanalyses. If the objective function is compatible with a packed representation, then it can be optimized efficiently by dynamic programming. For example, the distribution of parse trees for a given sentence under a PCFG can be represented as a packed forest from which the highest-probability tree can be easily extracted.

However, when the objective function f has no compatible packed representation, exact inference would be intractable. To alleviate this problem, one common approach from machine learning is loopy belief propagation (Pearl, 1988). Another solution (which is popular in NLP) is to split the computation into two phases: in the first phase, use some compatible objective function f' to produce a k-best list (the top k candidates under f'), which serves as an approximation to the full set. Then, in the second phase, optimize f over all the analyses in the k-best list. A typical example is discriminative reranking on k-best lists from a generative module, such as (Collins, 2000) for parsing and (Shen *et al.*, 2004) for translation, where the reranking model has nonlocal features that cannot be computed during parsing proper. Another example is minimum-Bayes-risk decoding (Kumar and Byrne, 2004; Goodman, 1998),where, assuming f' defines a probability distribution over all candidates, one seeks the candidate with the highest expected score according to an arbitrary metric (e.g., PARSEVAL or BLEU); since in general the metric will not be compatible with the parsing algorithm, the k-best lists can be used to approximate the full distribution f'. A similar situation occurs when the parser can produce multiple derivations that are regarded as equivalent (e.g., multiple lexicalized parse trees corresponding to the same unlexicalized parse tree); if we want the maximum *a posteriori* parse, we have to sum over equivalent derivations. Again, the equivalence relation will in general not be compatible with the parsing algorithm, so the k-best lists can be used to approximate f', as in Data Oriented Parsing (Bod, 2000) and in speech recognition (Mohri and Riley, 2002).

Another instance of this k-best approach is *cascaded optimization*. NLP systems are often cascades of modules, where we want to optimize the modules' objective functions jointly. However, often a module is incompatible with the packed representation of the previous module due to factors like non-local dependencies. So we might want to postpone some disambiguation by propagating k-best lists to subsequent phases, as in joint parsing and semantic role labeling (Gildea and Jurafsky, 2002; Sutton and McCallum, 2005), information extraction and coreference resolution (Wellner *et al.*, 2004), and formal semantics of TAG (Joshi and Vijay-Shanker, 1999).

Moreover, much recent work on *discriminative training* uses k-best lists; they are sometimes used to approximate the normalization constant or partition function (which would otherwise be intractable), or to train a model by optimizing some metric incompatible with the packed representation. For example, Och (2003) shows how to train a log-linear translation model not by maximizing the likelihood of training data, but maximizing the BLEU score (among other metrics) of the model on

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 53–64,
Vancouver, October 2005.

the data. Similarly, Chiang (2005) uses the k-best parsing algorithm described below in a CFG-based log-linear translation model in order to learn feature weights which maximize BLEU.

For algorithms whose packed representations are graphs, such as Hidden Markov Models and other finite-state methods, Ratnaparkhi's MXPARSE parser (Ratnaparkhi, 1997), and many stack-based machine translation decoders (Brown *et al.*, 1995; Och and Ney, 2004), the k-best paths problem is well-studied in both pure algorithmic context (see (Eppstein, 2001) and (Brander and Sinclair, 1995) for surveys) and NLP/Speech community (Mohri, 2002; Mohri and Riley, 2002). This paper, however, aims at the k-best tree algorithms whose packed representations are hypergraphs (Gallo *et al.*, 1993; Klein and Manning, 2001) (equivalently, and/or graphs or packed forests), which includes most parsers and parsing-based MT decoders. Any algorithm expressible as a weighted deductive system (Shieber *et al.*, 1995; Goodman, 1999; Nederhof, 2003) falls into this class. In our experiments, we apply the algorithms to the lexicalized PCFG parser of Bikel (2004), which is very similar to Collins' Model 2 (Collins, 2003), and to a synchronous CFG based machine translation system (Chiang, 2005).

2 Previous Work

As pointed out by Charniak and Johnson (2005), the major difficulty in k-best parsing is dynamic programming. The simplest method is to abandon dynamic programming and rely on aggressive pruning to maintain tractability, as is used in (Collins, 2000; Bikel, 2004). But this approach is prohibitively slow, and produces rather low-quality k-best lists (see Sec. 5.1.2). Gildea and Jurafsky (2002) described an $O(k^2)$-overhead extension for the CKY algorithm and reimplemented Collins' Model 1 to obtain k-best parses with an average of 14.9 parses per sentence. Their algorithm turns out to be a special case of our Algorithm 0 (Sec. 4.1), and is reported to also be prohibitively slow.

Since the original design of the algorithm described below, we have become aware of two efforts that are very closely related to ours, one by Jiménez and Marzal (2000) and another done in parallel to ours by Charniak and Johnson (2005). Jiménez and Marzal present an algorithm very similar to our Algorithm 3 (Sec. 4.4) while Charniak and Johnson propose using an algorithm similar to our Algorithm 0, but with multiple passes to improve efficiency. They apply this method to the Charniak (2000) parser to get 50-best lists for reranking, yielding an improvement in parsing accuracy.

Our work differs from Jiménez and Marzal's in the following three respects. First, we formulate the parsing problem in the more general framework of hypergraphs (Klein and Manning, 2001), making it applicable to a very wide variety of parsing algorithms, whereas Jiménez and Marzal define their algorithm as an extension of CKY, for CFGs in Chomsky Normal Form (CNF) only. This generalization is not only of theoretical importance, but also critical in the application to state-of-the-art parsers such as (Collins, 2003) and (Charniak, 2000). In Collins' parsing model, for instance, the rules are dynamically generated and include unary productions, making it very hard to convert to CNF by preprocessing, whereas our algorithms can be applied directly to these parsers. Second, our Algorithm 3 has an improvement over Jiménez and Marzal which leads to a slight theoretical and empirical speedup. Third, we have implemented our algorithms on top of state-of-the-art, large-scale statistical parser/decoders and report extensive experimental results while Jiménez and Marzal's was tested on relatively small grammars.

On the other hand, our algorithms are more scalable and much more general than the coarse-to-fine approach of Charniak and Johnson. In our experiments, we can obtain 10000-best lists nearly as fast as 1-best parsing, with very modest use of memory. Indeed, Charniak (p.c.) has adopted our Algorithm 3 into his own parser implementation and confirmed our findings.

In the literature of k shortest-path problems, Minieka (1974) generalized the Floyd algorithm in a way very similar to our Algorithm 0 and Lawler (1977) improved it using an idea similar to but a little slower than the binary branching case of our Algorithm 1. For hypergraphs, Gallo *et al.* (1993) study the shortest hyperpath problem and Nielsen *et al.* (2005) extend it to k shortest hyperpath. Our work differes from (Nielsen *et al.*, 2005) in two aspects. First, we solve the problem of k-best derivations (i.e., trees), not the k-best hyperpaths, although in many cases they coincide (see Sec. 3 for further discussions). Second, their work assumes non-negative costs (or probabilities ≤ 1) so that they can apply Dijkstra-like algorithms. Although generative models, being probability-based, do not suffer from this problem, more general models (e.g., log-linear models) may require negative edge costs (McDonald *et al.*, 2005; Taskar *et al.*, 2004). Our work, based on the Viterbi algorithm, is still applicable as long as the hypergraph is acyclic, and is used by McDonald *et al.* (2005) to get the k-best parses.

3 Formulation

Following Klein and Manning (2001), we use weighted directed hypergraphs (Gallo *et al.*, 1993) as an abstraction of the probabilistic parsing problem.

Definition 1. An *ordered hypergraph* (henceforth *hypergraph*) H is a tuple $\langle V, E, t, \mathbf{R} \rangle$, where V is a finite set of *vertices*, E is a finite set of *hyperarcs*, and $\mathbf{R}$ is the set of *weights*. Each hyperarc $e \in E$ is a triple

$e = \langle T(e), h(e), f(e)\rangle$, where $h(e) \in V$ is its *head* and $T(e) \in V^*$ is a vector of *tail* nodes. $f(e)$ is a *weight function* from $\mathbf{R}^{|T(e)|}$ to $\mathbf{R}$. $t \in V$ is a distinguished vertex called *target vertex*.

Note that our definition is different from those in previous work in the sense that the tails are now *vectors* rather than sets, so that we can allow multiple occurrences of the same vertex in a tail and there is an ordering among the components of a tail.

Definition 2. A hypergraph H is said to be *monotonic* if there is a total ordering $\preceq$ on $\mathbf{R}$ such that every weight function f in H is monotonic in each of its arguments according to $\preceq$, i.e., if $f : \mathbf{R}^m \mapsto \mathbf{R}$, then $\forall 1 \leq i \leq m$, if $a_i \preceq a_i'$, then $f(a_1, \cdots, a_i, \cdots, a_m) \preceq f(a_1, \cdots, a_i', \cdots, a_m)$. We also define the comparison function $\min_{\preceq}(a, b)$ to output a if $a \preceq b$, or b if otherwise.

In this paper we will assume this monotonicity, which corresponds to the optimal substructure property in dynamic programming (Cormen *et al.*, 2001).

Definition 3. We denote $|e| = |T(e)|$ to be the *arity* of the hyperarc. If $|e| = 0$, then $f(e) \in \mathbf{R}$ is a constant and we call $h(e)$ a *source vertex*. We define the *arity* of a hypergraph to be the maximum arity of its hyperarcs.

Definition 4. The *backward-star* $BS(v)$ of a vertex v is the set of incoming hyperarcs $\{e \in E \mid h(e) = v\}$. The *in-degree* of v is $|BS(v)|$.

Definition 5. A *derivation* D of a vertex v in a hypergraph H, its size $|D|$ and its weight $w(D)$ are recursively defined as follows:

- If $e \in BS(v)$ with $|e| = 0$, then $D = \langle e, \epsilon\rangle$ is a derivation of v, its size $|D| = 1$, and its weight $w(D) = f(e)()$.
- If $e \in BS(v)$ where $|e| > 0$ and D_i is a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, then $D = \langle e, D_1 \cdots D_{|e|}\rangle$ is a derivation of v, its size $|D| = 1 + \sum_{i=1}^{|e|} |D_i|$ and its weight $w(D) = f(e)(w(D_1), \ldots, w(D_{|e|}))$.

The ordering on weights in $\mathbf{R}$ induces an ordering on derivations: $D \preceq D'$ iff $w(D) \preceq w(D')$.

Definition 6. Define $D_i(v)$ to be the i^{th}-best derivation of v. We can think of $D_1(v), \ldots, D_k(v)$ as the components of a vector we shall denote by $\mathbf{D}(v)$. The *k-best derivations problem* for hypergraphs, then, is to find $\mathbf{D}(t)$ given a hypergraph $\langle V, E, t, \mathbf{R}\rangle$.

With the derivations thus ranked, we can introduce a nonrecursive representation for derivations that is analogous to the use of *back-pointers* in parser implementation.

Definition 7. A *derivation with back-pointers (dbp)* $\hat{D}$ of v is a tuple $\langle e, \mathbf{j}\rangle$ such that $e \in BS(v)$, and $\mathbf{j} \in \{1, 2, \ldots, k\}^{|e|}$. There is a one-to-one correspondence $\sim$ between dbps of v and derivations of v:

$$\langle e, (j_1 \cdots j_{|e|})\rangle \sim \langle e, D_{j_1}(T_1(e)) \cdots D_{j_{|e|}}(T_{|e|}(e))\rangle$$

Accordingly, we extend the weight function w to dbps: $w(\hat{D}) = w(D)$ if $\hat{D} \sim D$. This in turn induces an ordering on dbps: $\hat{D} \preceq \hat{D}'$ iff $w(\hat{D}) \preceq w(\hat{D}')$. Let $\hat{D}_i(v)$ denote the i^{th}-best dbp of v.

Where no confusion will arise, we use the terms 'derivation' and 'dbp' interchangeably.

Computationally, then, the k-best problem can be stated as follows: given a hypergraph H with arity a, compute $\hat{D}_1(t), \ldots, \hat{D}_k(t)$.[1]

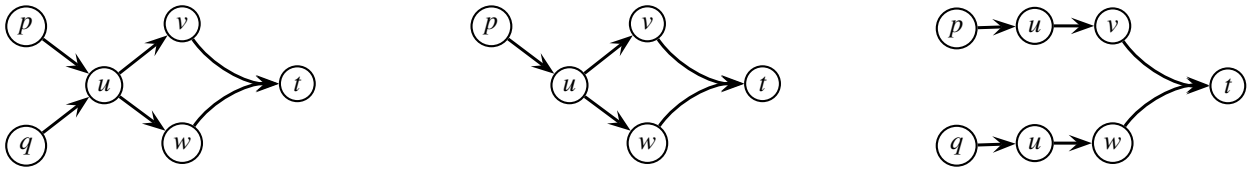
As shown by Klein and Manning (2001), hypergraphs can be used to represent the search space of most parsers (just as graphs, also known as trellises or lattices, can represent the search space of finite-state automata or HMMs). More generally, hypergraphs can be used to represent the search space of most *weighted deductive system* (Nederhof, 2003). For example, the weighted CKY algorithm given a context-free grammar $G = \langle N, T, P, S\rangle$ in Chomsky Normal Form (CNF) and an input string w can be represented as a hypergraph of arity 2 as follows. Each *item* $[X, i, j]$ is represented as a vertex v, corresponding to the recognition of nonterminal X spanning w from positions $i+1$ through j. For each production rule $X \to YZ$ in P and three free indices $i < j < k$, we have a hyperarc $\langle((Y, i, k), (Z, k, j)), (X, i, k), f\rangle$ corresponding to the *instantiation* of the inference rule COMPLETE in the deductive system of (Shieber *et al.*, 1995), and the weight function f is defined as $f(a, b) = ab \cdot \Pr(X \to YZ)$, which is the same as in (Nederhof, 2003). In this sense, hypergraphs can be thought of as *compiled* or *instantiated* versions of weighted deductive systems.

A parser does nothing more than traverse this hypergraph. In order that derivation values be computed correctly, however, we need to traverse the hypergraph in a particular order:

Definition 8. The *graph projection* of a hypergraph $H = \langle V, E, t, \mathbf{R}\rangle$ is a directed graph $G = \langle V, E'\rangle$ where $E' = \{(u, v) \mid \exists e \in BS(v), u \in T(e)\}$. A hypergraph H is said to be *acyclic* if its graph projection G is a directed acyclic graph; then a *topological ordering* of H is an ordering of V that is a topological ordering in G (from sources to target).

We assume the input hypergraph is acyclic so that we can use its topological ordering to traverse it. In practice the hypergraph is typically not known in advance, but the

[1] Note that although we have defined the weight of a derivation as a function on derivations, in practice one would store a derivation's weight inside the dbp itself, to avoid recomputing it over and over.



(a) (b) (c)

Figure 1: Examples of hypergraph, hyperpath, and derivation: (a) a hypergraph H, with t as the target vertex and p, q as source vertices, (b) a hyperpath π_t in H, and (c) a derivation of t in H, where vertex u appears twice with two different (sub-)derivations. This would be impossible in a hyperpath.

topological ordering often is, so that the (dynamic) hypergraph can be generated in that order. For example, for CKY it is sufficient to generate all items $[X, i, j]$ before all items $[Y, i', j']$ when $j' - i' > j - i$ (X and Y are arbitrary nonterminals).

Excursus: Derivations and Hyperpaths

The work of Klein and Manning (2001) introduces a correspondence between hyperpaths and derivations. When extended to the k-best case, however, that correspondence no longer holds.

Definition 9. (Nielsen *et al.*, 2005) Given a hypergraph $H = \langle V, E, t, \mathbf{R} \rangle$, a *hyperpath* π_v of destination $v \in V$ is an *acyclic minimal* hypergraph $H_\pi = \langle V_\pi, E_\pi, v, \mathbf{R} \rangle$ such that

1. $E_\pi \subseteq E$
2. $v \in V_\pi = \bigcup_{e \in E_\pi} (T(e) \cup \{h(e)\})$
3. $\forall u \in V_\pi$, u is either a source vertex or connected to a source vertex in H_π.

As illustrated by Figure 1, derivations (as trees) are different from hyperpaths (as minimal hypergraphs) in the sense that in a derivation the same vertex can appear more than once with possibly different sub-derivations while it is represented at most once in a hyperpath. Thus, the k-best derivations problem we solve in this paper is very different in nature from the k-shortest hyperpaths problem in (Nielsen *et al.*, 2005).

However, the two problems do coincide when $k = 1$ (since all the sub-derivations must be optimal) and for this reason the 1-best hyperpath algorithm in (Klein and Manning, 2001) is very similar to the 1-best tree algorithm in (Knuth, 1977). For k-best case ($k > 1$), they also coincide when the hypergraph is isomorphic to a Case-Factor Diagram (CFD) (McAllester *et al.*, 2004) (proof omitted). The derivation forest of CFG parsing under the CKY algorithm, for instance, can be represented as a CFD while the forest of Earley algorithm can not. An item (or equivalently, a vertex in hypergraph) can appear twice in an Earley derivation because of the prediction rule (see Figure 2 for an example).

$$\dfrac{\dfrac{(A \to \alpha.B\beta, i, j)}{(B \to .\gamma, j, j)} \quad \cdots}{(A \to \alpha.B\beta, i, j) \qquad (B \to \gamma., j, k)} \quad \cdots$$
$$(A \to \alpha B.\beta, i, k)$$

Figure 2: An Earley derivation. Note that item $(A \to \alpha.B\beta, i, j)$ appears twice (predict and complete).

1: **procedure** VITERBI(k)
2: **for** $v \in V$ in topological order **do**
3: **for** $e \in BS(v)$ **do** ▷ for all incoming hyperarcs
4: $\hat{D}_1(v) \leftarrow \min_{\preceq}(\hat{D}_1(v), \langle e, \mathbf{1} \rangle)$ ▷ update

Figure 3: The generic 1-best Viterbi algorithm

The k-best derivations problem has potentially more applications in tree generation (Knight and Graehl, 2005), which can not be modeled by hyperpaths. But detailed discussions along this line are out of the scope of this paper.

4 Algorithms

The traditional 1-best Viterbi algorithm traverses the hypergraph in topological order and for each vertex v, calculates its 1-best derivation $D_1(v)$ using all incoming hyperarcs $e \in BS(v)$ (see Figure 3). If we take the arity of the hypergraph to be constant, then the overall time complexity of this algorithm is $O(|E|)$.

4.1 Algorithm 0: naïve

Following (Goodman, 1999; Mohri, 2002), we isolate two basic operations in line 4 of the 1-best algorithm that

can be generalized in order to extend the algorithm: first, the formation of the derivation $\langle e, \mathbf{1} \rangle$ out of $|e|$ best subderivations (this is a generalization of the binary operator $\otimes$ in a semiring); second, $\min_{\preceq}$, which chooses the better of two derivations (same as the $\oplus$ operator in an *idempotent* semiring (Mohri, 2002)). We now generalize these two operations to operate on k-best lists.

Let $r = |e|$. The new multiplication operation, $\mathbf{mult}_{\preceq k}(e)$, is performed in three steps:

1. enumerate the k^r derivations $\{\langle e, j_1 \cdots j_r \rangle \mid \forall i, 1 \leq j_i \leq k\}$. Time: $O(k^r)$.
2. sort these k^r derivations (according to weight). Time: $O(k^r \log(k^r)) = O(rk^r \log k)$.
3. select the first k elements from the sorted list of k^r elements. Time: $O(k)$.

So the overall time complexity of $\mathbf{mult}_{\preceq k}$ is $O(rk^r \log k)$.

We also have to extend $\min_{\preceq}$ to $\mathbf{merge}_{\preceq k}$, which takes two vectors of length k (or fewer) as input and outputs the top k (in sorted order) of the $2k$ elements. This is similar to merge-sort (Cormen *et al.*, 2001) and can be done in linear time $O(k)$. Then, we only need to rewrite line 4 of the Viterbi algorithm (Figure 3) to extend it to the k-best case:

$$4: \quad \hat{\mathbf{D}}(v) \leftarrow \mathbf{merge}_{\preceq k}(\hat{\mathbf{D}}(v), \mathbf{mult}_{\preceq k}(e))$$

and the time complexity for this line is $O(|e|k^{|e|} \log k)$, making the overall complexity $O(|E|k^a \log k)$ if we consider the arity a of the hypergraph to be constant.[2] The overall space complexity is $O(|V|k)$ since for each vertex we need to store a vector of length k.

In the context of CKY parsing for CFG, the 1-best Viterbi algorithm has complexity $O(n^3|P|)$ while the k-best version is $O(n^3|P|k^2 \log k)$, which is slower by a factor of $O(k^2 \log k)$.

4.2 Algorithm 1: speed up $\mathbf{mult}_{\preceq k}$

First we seek to exploit the fact that input vectors are all sorted and the function f is monotonic; moreover, we are only interested in the top k elements of the $k^{|e|}$ possibilities.

Define $\mathbf{1}$ to be the vector whose elements are all 1; define $\mathbf{b}^i$ to be the vector whose elements are all 0 except $b^i_i = 1$.

As we compute $\mathbf{p}_e = \mathbf{mult}_{\preceq k}(e)$, we maintain a *candidate set* C of derivations that have the potential to be the next best derivation in the list. If we picture the input as an $|e|$-dimensional space, C contains those derivations that have not yet been included in $\mathbf{p}_e$, but are on the boundary with those which have. It is initialized to $\{\langle e, \mathbf{1} \rangle\}$. At each step, we extract the best derivation from C—call it $\langle e, \mathbf{j} \rangle$—and append it to $\mathbf{p}_e$. Then $\langle e, \mathbf{j} \rangle$ must be replaced in C by its neighbors,

$$\{\langle e, \mathbf{j} + \mathbf{b}^l \rangle \mid 1 \leq l \leq |e|\}$$

(see Figure 4.2 for an illustration). We implement C as a priority queue (Cormen *et al.*, 2001) to make the extraction of its best derivation efficient. At each iteration, there are one Extract-Min and $|e|$ Insert operations. If we use a binary-heap implementation for priority queues, we get $O(|e| \log k|e|)$ time complexity for each iteration.[3] Since we are only interested in the top k elements, there are k iterations and the time complexity for a single $\mathbf{mult}_{\preceq k}$ is $O(k|e| \log k|e|)$, yielding an overall time complexity of $O(|E|k \log k)$ and reducing the multiplicative overhead by a factor of $O(k^{a-1})$ (again, assuming a is constant). In the context of CKY parsing, this reduces the overhead to $O(k \log k)$. Figure 5 shows the additional pseudocode needed for this algorithm. It is integrated into the Viterbi algorithm (Figure 3) simply by rewriting line 4 of to invoke the function $\textsc{Mult}(e, k)$:

$$4: \quad \hat{\mathbf{D}}(v) \leftarrow \mathbf{merge}_{\preceq k}(\hat{\mathbf{D}}(v), \textsc{Mult}(e, k))$$

4.3 Algorithm 2: combine $\mathbf{merge}_{\preceq k}$ into $\mathbf{mult}_{\preceq k}$

We can further speed up both $\mathbf{merge}_{\preceq k}$ and $\mathbf{mult}_{\preceq k}$ by a similar idea. Instead of letting each $\mathbf{mult}_{\preceq k}$ generate a full k derivations for each hyperarc e and only then applying $\mathbf{merge}_{\preceq k}$ to the results, we can combine the candidate sets for all the hyperarcs into a single candidate set. That is, we initialize C to $\{\langle e, \mathbf{1} \rangle \mid e \in BS(v)\}$, the set of all the top parses from each incoming hyperarc (cf. Algorithm 1). Indeed, it suffices to keep only the top k out of the $|BS(v)|$ candidates in C, which would lead to a significant speedup in the case where $|BS(v)| \gg k$. [4] Now the top derivation in C is the top derivation for v. Then, whenever we remove an element $\langle e, \mathbf{j} \rangle$ from C, we replace it with the $|e|$ elements $\{\langle e, \mathbf{j} + \mathbf{b}^l \rangle \mid 1 \leq l \leq |e|\}$ (again, as in Algorithm 1). The full pseudocode for this algorithm is shown in Figure 6.

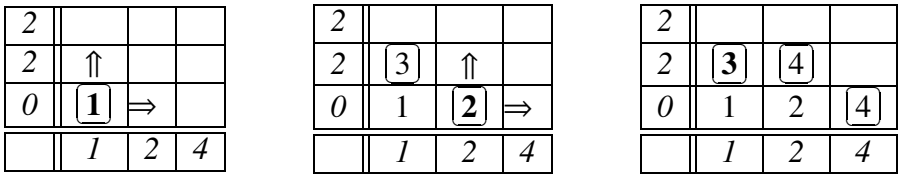
4.4 Algorithm 3: compute $\mathbf{mult}_{\preceq k}$ lazily

Algorithm 2 exploited the idea of lazy computation: performing $\mathbf{mult}_{\preceq k}$ only as many times as necessary. But this algorithm still calculates a full k-best list for every vertex in the hypergraph, whereas we are only interested in

[2] Actually, we do not need to sort all $k^{|e|}$ elements in order to extract the top k among them; there is an efficient algorithm (Cormen *et al.*, 2001) that can select the kth best element from the $k^{|e|}$ elements in time $O(k^{|e|})$. So we can improve the overhead to $O(k^a)$.

[3] If we maintain a Min-Heap along with the Min-Heap, we can reduce the per-iteration cost to $O(|e| \log k)$, and with Fibonacci heap we can further improve it to be $O(|e| + \log k)$. But these techniques do not change the overall complexity when a is constant, as we will see.

[4] This can be implemented by a linear-time randomized-selection algorithm (a.k.a. quick-select) (Cormen *et al.*, 2001).



(a) (b) (c)

Figure 4: An illustration of Algorithm 1 in $|e| = 2$ dimensions. Here $k = 3$, $\preceq$ is the numerical $\leq$, and the monotonic function f is defined as $f(a, b) = a + b$. Italic numbers on the x and y axes are a_i's and b_j's, respectively. We want to compute the top 3 results from $f(a_i, b_j)$ with $1 \leq i, j \leq 3$. In each iteration the current *frontier* is shown in oval boxes, with the bold-face denoting the best element among them. That element will be extracted and replaced by its two neighbors (⇑ and ⇒) in the next iteration.

```
1: function Mult(e, k)
2:     cand ← {⟨e, 1⟩}                          ▷ initialize the heap
3:     p ← empty list                           ▷ the result of mult_{≤k}
4:     while |p| < k and |cand| > 0 do
5:         AppendNext(cand, p, k)
6:     return p
7:
8: procedure AppendNext(cand, p)
9:     ⟨e, j⟩ ← Extract-Min(cand)
10:    append ⟨e, j⟩ to p
11:    for i ← 1 . . . |e| do                   ▷ add the |e| neighbors
12:        j′ ← j + b^i
13:        if j′_i ≤ |D̂(T_i(e))| and ⟨e, j′⟩ ∉ cand then
14:            Insert(cand, ⟨e, j′⟩)            ▷ add to heap
```

Figure 5: Part of Algorithm 1.

```
1: procedure FindAllKBest(k)
2:     for v ∈ V in topological order do
3:         FindKBest(v, k)
4:
5: procedure FindKBest(v, k)
6:     GetCandidates(v, k)                      ▷ initialize the heap
7:     while |D̂(v)| < k and |cand[v]| > 0 do
8:         AppendNext(cand[v], D̂(v))
9:
10: procedure GetCandidates(v, k)
11:    temp ← {⟨e, 1⟩ | e ∈ BS(v)}
12:    cand[v] ← the top k elements in temp     ▷ prune
           away useless candidates
13:    Heapify(cand[v])
```

Figure 6: Algorithm 2

```
1: procedure LazyKthBest(v, k, k′)                          ▷ k′ is the global k
2:     if |D̂(v)| ≥ k then                                  ▷ k^th derivation already computed?
3:         return
4:     if cand[v] is not defined then                       ▷ first visit of vertex v?
5:         GetCandidates(v, k′)                             ▷ initialize the heap
6:         append Extract-Min(cand[v]) to D̂(v)              ▷ 1-best
7:     while |D̂(v)| < k and |cand[v]| > 0 do
8:         ⟨e, j⟩ ← D̂_{|D̂(v)|}(v)                          ▷ last derivation
9:         LazyNext(cand[v], e, j, k′)                      ▷ update the heap, adding the successors of last derivation
10:        append Extract-Min(cand[v]) to D̂(v)              ▷ get the next best derivation and delete it from the heap
11:
12: procedure LazyNext(cand, e, j, k′)
13:    for i ← 1 . . . |e| do                               ▷ add the |e| neighbors
14:        j′ ← j + b^i
15:        LazyKthBest(T_i(e), j′_i, k′)                    ▷ recursively solve a sub-problem
16:        if j′_i ≤ |D̂(T_i(e))| and ⟨e, j′⟩ ∉ cand then    ▷ if it exists and is not in heap yet
17:            Insert(cand, ⟨e, j′⟩)                        ▷ add to heap
```

Figure 7: Algorithm 3

Algorithm	Time Complexity
1-best Viterbi	$O(E)$
Algorithm 0	$O(Ek^a \log k)$
Algorithm 1	$O(Ek \log k)$
Algorithm 2	$O(E + Vk \log k)$
Algorithm 3	$O(E + \lvert D_{max}\rvert k \log k)$
generalized J&M	$O(E + \lvert D_{max}\rvert k \log(d + k))$

Table 1: Summary of Algorithms.

the k-best derivations of the target vertex (goal item). We can therefore take laziness to an extreme by delaying the whole k-best calculation until after parsing. Algorithm 3 assumes an initial parsing phase that generates the hypergraph and finds the 1-best derivation of each item; then in the second phase, it proceeds as in Algorithm 2, but starts at the goal item and calls itself recursively only as necessary. The pseudocode for this algorithm is shown in Figure 7. As a side note, this second phase should be applicable also to a cyclic hypergraph as long as its derivation weights are bounded.

Algorithm 2 has an overall complexity of $O(|E| + |V|k \log k)$ and Algorithm 3 is $O(|E| + |D_{\max}|k \log k)$ where $|D_{\max}|$ is the size of the longest among all top k derivations (for CFG in CNF, $|D| = 2n - 1$ for all D, so $|D_{\max}|$ is $O(n)$). These are significant improvements against Algorithms 0 and 1 since it turns the multiplicative overhead into an additive overhead. In practice, $|E|$ usually dominates, as in CKY parsing of CFG. So theoretically the running times grow very slowly as k increases, which is exactly demonstrated by our experiments below.

4.5 Summary and Discussion of Algorithms

The four algorithms, along with the 1-best Viterbi algorithm and the generalized Jiménez and Marzal algorithm, are compared in Table 1.

The key difference between our Algorithm 3 and Jiménez and Marzal's algorithm is the restriction of top k candidates before making heaps (line 11 in Figure 6, see also Sec. 4.3). Without this line Algorithm 3 could be considered as a generalization of the Jiménez and Marzal algorithm to the case of acyclic monotonic hypergraphs. This line is also responsible for improving the time complexity from $O(|E| + |D_{\max}|k \log(d + k))$ (generalized Jiménez and Marzal algorithm) to $O(|E| + |D_{\max}|k \log k)$, where $d = \max_v |BS(v)|$ is the maximum in-degree among all vertices. So in case $k < d$, our algorithm outperforms Jiménez and Marzal's.

5 Experiments

We report results from two sets of experiments. For probabilistic parsing, we implemented Algorithms 0, 1, and 3 on top of a widely-used parser (Bikel, 2004) and conducted experiments on parsing efficiency and the quality of the k-best-lists. We also implemented Algorithms 2 and 3 in a parsing-based MT decoder (Chiang, 2005) and report results on decoding speed.

5.1 Experiment 1: Bikel Parser

Bikel's parser (2004) is a state-of-the-art multilingual parser based on lexicalized context-free models (Collins, 2003; Eisner, 2000). It does support k-best parsing, but, following Collins' parse-reranking work (Collins, 2000) (see also Section 5.1.2), it accomplishes this by simply abandoning dynamic programming, i.e., no items are considered equivalent (Charniak and Johnson, 2005). Theoretically, the time complexity is *exponential* in n (the input sentence length) and constant in k, since, without merging of equivalent items, there is no limit on the number of items in the chart. In practice, beam search is used to reduce the observed time.[5] But with the standard beam width of 10^{-4}, this method becomes prohibitively expensive for $n \geq 25$ on Bikel's parser. Collins (2000) used a narrower 10^{-3} beam and further applied a cell limit of 100,[6] but, as we will show below, this has a detrimental effect on the quality of the output. We therefore omit this method from our speed comparisons, and use our implementation of Algorithm 0 (naïve) as the baseline.

We implemented our k-best Algorithms 0, 1, and 3 on top of Bikel's parser and conducted experiments on a 2.4 GHz 64-bit AMD Opteron with 32 GB memory. The program is written in Java 1.5 running on the Sun JVM in server mode with a maximum heap size of 5 GB. For this experiment, we used sections 02–21 of the Penn Treebank (PTB) (Marcus *et al.*, 1993) as the training data and section 23 (2416 sentences) for evaluation, as is now standard. We ran Bikel's parser using its settings to emulate Model 2 of (Collins, 2003).

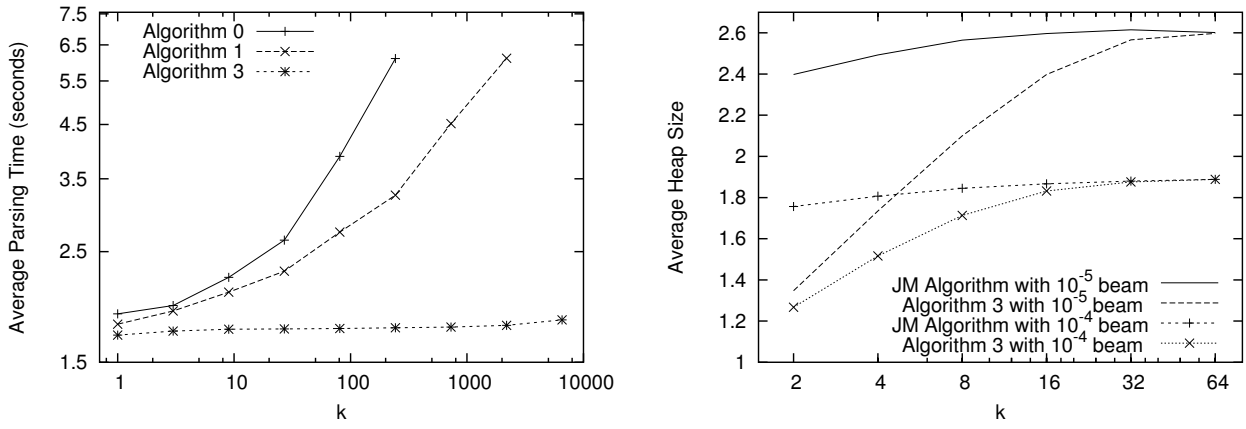
5.1.1 Efficiency

We tested our algorithms under various conditions. We first did a comparison of the average parsing time per sentence of Algorithms 0, 1, and 3 on section 23, with $k \leq 10000$ for the standard beam of width 10^{-4}. Figure 8(a) shows that the parsing speed of Algorithm 3 improved dramatically against the other algorithms and is nearly constant in k, which exactly matches the complexity analysis. Algorithm 1 ($k \log k$) also significantly outperforms the baseline naïve algorithm ($k^2 \log k$).

We also did a comparison between our Algorithm 3 and the Jiménez and Marzal algorithm in terms of average

[5] In beam search, or *threshold pruning*, each cell in the chart (typically containing all the items corresponding to a span $[i, j]$) is reduced by discarding all items that are worse than β times the score of the best item in the cell. This β is known as the *beam width*.

[6] In this type of pruning, also known as *histogram pruning*, only the α best items are kept in each cell. This α is called the *cell limit*.



(a) Average parsing speed (Algs. 0 vs. 1 vs. 3, log-log)

(b) Average heap size (Alg. 3 vs. Jiménez and Marzal)

Figure 8: Efficiency results of the k-best Algorithms, compared to Jiménez and Marzal's algorithm

heap size. Figure 8(b) shows that for larger k, the two algorithms have the same average heap size, but for smaller k, our Algorithm 3 has a considerably smaller average heap size. This difference is useful in applications where only short k-best lists are needed. For example, McDonald *et al.* (2005) find that $k = 5$ gives optimal parsing accuracy.

5.1.2 Accuracy

Our efficient k-best algorithms enable us to search over a larger portion of the whole search space (e.g. by less aggressive pruning), thus producing k-best lists with better quality than previous methods. We demonstrate this by comparing our k-best lists to those in (Ratnaparkhi, 1997), (Collins, 2000) and the parallel work by Charniak and Johnson (2005) in several ways, including oracle reranking and average number of found parses.

Ratnaparkhi (1997) introduced the idea of oracle reranking: suppose there exists a perfect reranking scheme that magically picks the best parse that has the highest F-score among the top k parses for each sentence. Then the performance of this oracle reranking scheme is the upper bound of any actual reranking system like (Collins, 2000).As k increases, the F-score is nondecreasing, and there is some k (which might be very large) at which the F-score converges.

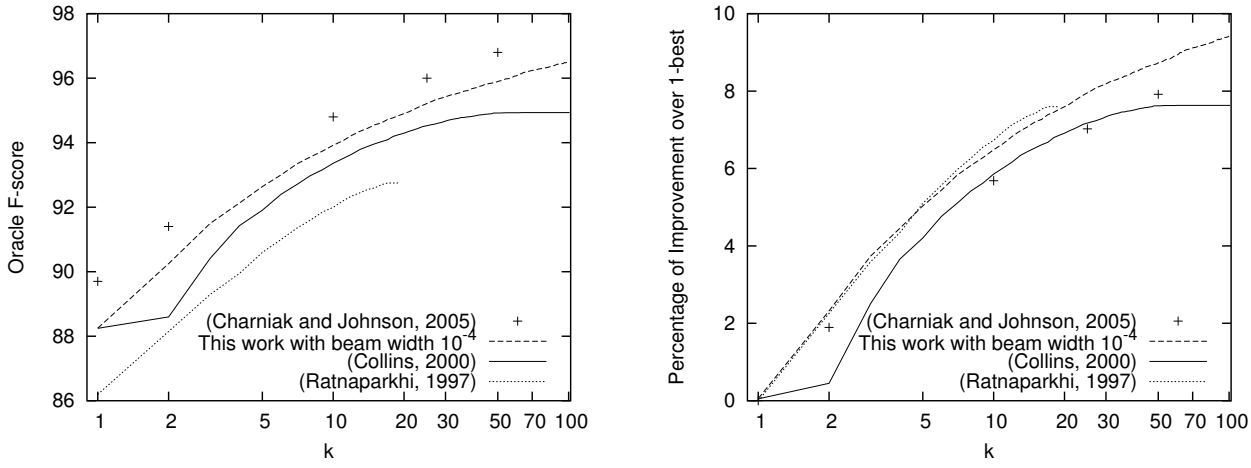
Ratnaparkhi reports experiments using oracle reranking with his statistical parser MXPARSE, which can compute its k-best parses (in his experiments, $k = 20$). Collins (2000), in his parse-reranking experiments, used his Model 2 parser (Collins, 2003) with a beam width of 10^{-3} together with a cell limit of 100 to obtain k-best lists; the average number of parses obtained per sentence was 29.2, the maximum, 101.[7] Charniak and Johnson (2005) use coarse-to-fine parsing on top of the Charniak (2000) parser and get 50-best lists for section 23.

Figure 9(a) compares the results of oracle reranking. Collins' curve converges at around $k = 50$ while ours continues to increase. With a beam width of 10^{-4} and $k = 100$, our parser plus oracle reaches an F-score of 96.4%, compared to Collins' 94.9%. Charniak and Johnson's work, however, is based on a completely different parser whose 1-best F-score is 1.5 points higher than the 1-bests of ours and Collins', making it difficult to compare in absolute numbers. So we instead compared the *relative* improvement over 1-best. Figure 9(b) shows that our work has the largest percentage of improvement in terms of F-score when $k > 20$.

To further explore the impact of Collins' cell limit on the quality of k-best lists, we plotted average number of parses for a given sentence length (Figure 10). Generally speaking, as input sentences get longer, the number of parses grows (exponentially). But we see that the curve for Collins' k-best list goes down for large k (> 40). We suspect this is due to the cell limit of 100 pruning away potentially good parses too early in the chart. As sentences get longer, it is more likely that a lower-probability parse might contribute eventually to the k-best parses. So we infer that Collins' k-best lists have limited quality for large k, and this is demonstrated by the early convergence of its oracle-reranking score. By comparison, our curves of both beam widths continue to grow with $k = 100$.

All these experiments suggest that our k-best parses are of better quality than those from previous k-best parsers,

[7] The reason the maximum is 101 and not 100 is that Collins merged the 100-best list using a beam of 10^{-3} with the 1-best list using a beam of 10^{-4} (Collins, p.c.).



(a) Oracle Reranking

(b) Relative Improvement

Figure 9: Absolutive and Relative F-scores of oracle reranking for the top k ($\leq$ 100) parses for section 23, compared to (Charniak and Johnson, 2005), (Collins, 2000) and (Ratnaparkhi, 1997).

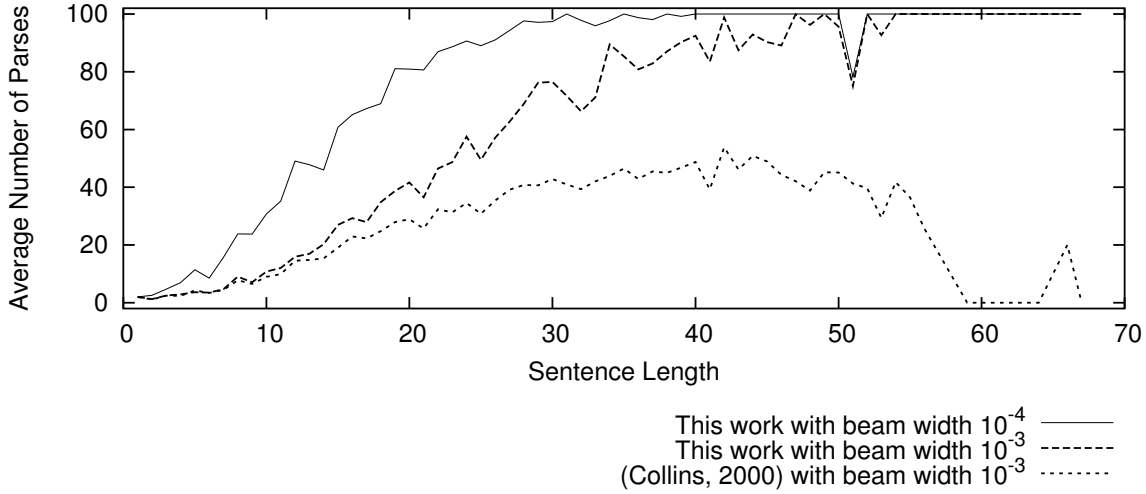


Figure 10: Average number of parses for each sentence length in section 23, using k=100, with beam width 10^{-4} and 10^{-3}, compared to (Collins, 2000).

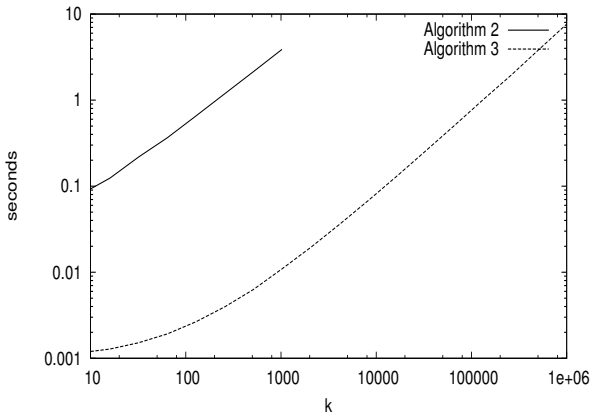


Figure 11: Algorithm 2 compared with Algorithm 3 (offline) on MT decoding task. Average time (both excluding initial 1-best phase) vs. k (log-log).

and similar quality to those from (Charniak and Johnson, 2005) which has so far the highest F-score after reranking, and this might lead to better results in real parse reranking.

5.2 Experiment 2: MT decoder

Our second experiment was on a CKY-based decoder for a machine translation system (Chiang, 2005), implemented in Python 2.4 accelerated with Psyco 1.3 (Rigo, 2004). We implemented Algorithms 2 and 3 to compute k-best English translations of Mandarin sentences. Because the CFG used in this system is large to begin with (millions of rules), and then effectively intersected with a finite-state machine on the English side (the language model), the grammar constant for this system is quite large. The decoder uses a relatively narrow beam search for efficiency.

We ran the decoder on a 2.8 GHz Xeon with 4 GB of memory, on 331 sentences from the 2002 NIST MTEval test set. We tested Algorithm 2 for $k = 2^i, 3 \leq i \leq 10$, and Algorithm 3 (offline algorithm) for $k = 2^i, 3 \leq i \leq 20$. For each sentence, we measured the time to calculate the k-best list, *not* including the initial 1-best parsing phase. We then averaged the times over our test set to produce the graph of Figure 11, which shows that Algorithm 3 runs an average of about 300 times faster than Algorithm 2. Furthermore, we were able to test Algorithm 3 up to $k = 10^6$ in a reasonable amount of time.[8]

[8]The curvature in the plot for Algorithm 3 for $k < 1000$ may be due to lack of resolution in the timing function for short times.

6 Conclusion

The problem of k-best parsing and the effect of k-best list size and quality on applications are subjects of increasing interest for NLP research. We have presented here a general-purpose algorithm for k-best parsing and applied it to two state-of-the-art, large-scale NLP systems: Bikel's implementation of Collins' lexicalized PCFG model (Bikel, 2004; Collins, 2003) and Chiang's synchronous CFG based decoder (Chiang, 2005) for machine translation. We hope that this work will encourage further investigation into whether larger and better k-best lists will improve performance in NLP applications, questions which we ourselves intend to pursue as well.

Acknowledgements

We would like to thank one of the anonymous reviewers of a previous version of this paper for pointing out the work by Jiménez and Marzal, and Eugene Charniak and Mark Johnson for providing an early draft of their paper and very useful comments. We are also extremely grateful to Dan Bikel for the help in experiments, and Michael Collins for providing the data in his paper. Our thanks also go to Dan Gildea, Jonathan Graehl, Julia Hockenmaier, Aravind Joshi, Kevin Knight, Daniel Marcu, Mitch Marcus, Ryan McDonald, Fernando Pereira, Giorgio Satta, Libin Shen, and Hao Zhang.

References

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics*, **30**, 479–511.

Bod, R. (2000). Parsing with the shortest derivation. In *Proc. Eighteenth International Conference on Computational Linguistics (COLING)*, pages 69–75.

Brander, A. and Sinclair, M. (1995). A comparative study of k-shortest path algorithms. In *Proc. 11th UK Performance Engineering Workshop for Computer and Telecommunications Systems*.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lai, J. C., and Mercer, R. L. (1995). Method and system for natural language translation. U. S. Patent 5,477,451.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proc. First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine-grained n-best parsing and discriminative reranking. In *Proc. ACL 2005*.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proc. ACL 2005*.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *Proc. Seventeenth International Conference on Machine Learning (ICML)*, pages 175–182. Morgan Kaufmann.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, **29**, 589–637.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, second edition.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In H. Bunt and A. Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers.

Eppstein, D. (2001). Bibliography on *k* shortest paths and other "*k* best solutions" problems. http://www.ics.uci.edu/~eppstein/bibs/kpath.bib.

Gallo, G., Longo, G., and Pallottino, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, **42**(2), 177–201.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, **28**(3), 245–288.

Goodman, J. (1998). *Parsing Inside-Out*. Ph.D. thesis, Harvard University.

Goodman, J. (1999). Semiring parsing. *Computational Linguistics*, **25**, 573–605.

Jiménez, V. M. and Marzal, A. (2000). Computation of the *n* best parse trees for weighted and stochastic context-free grammars. In *Proc. of the Joint IAPR International Workshops on Advances in Pattern Recognition*.

Joshi, A. K. and Vijay-Shanker, K. (1999). Compositional semantics with lexicalized tree-adjoining grammar (LTAG): How much underspecification is necessary? In H. C. Bunt and E. G. C. Thijsse, editors, *Proc. IWCS-3*, pages 131–145.

Klein, D. and Manning, C. D. (2001). Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*. Tsinghua University Press.

Knight, K. and Graehl, J. (2005). An overview of probabilistic tree transducers for natural language processing. In *Proc. of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, LNCS.

Knuth, D. (1977). A generalization of Dijkstra's algorithm. *Information Processing Letters*, **6**(1).

Kumar, S. and Byrne, W. (2004). Minimum bayes-risk decoding for statistical machine translation. In *HLT-NAACL*.

Lawler, E. L. (1977). Comment on computing the *k* shortest paths in a graph. *Comm. of the ACM*, **20**(8), 603–604.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, **19**, 313–330.

McAllester, D., Collins, M., and Pereira, F. (2004). Case-factor diagrams for structured probabilistic modeling. In *Proc. UAI 2004*.

McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proc. ACL 2005*.

Minieka, E. (1974). On computing sets of shortest paths in a graph. *Comm. of the ACM*, **17**(6), 351–353.

Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, **7**(3), 321–350.

Mohri, M. and Riley, M. (2002). An efficient algorithm for the n-best-strings problem. In *Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP '02)*, Denver, Colorado.

Nederhof, M.-J. (2003). Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, pages 135–143.

Nielsen, L. R., Andersen, K. A., and Pretolani, D. (2005). Finding the k shortest hyperpaths. *Computers and Operations Research*.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proc. ACL 2003*, pages 160–167.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, **30**, 417–449.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Proc. EMNLP 1997*, pages 1–10.

Rigo, A. (2004). Representation-based just-in-time specialization and the Psyco prototype for Python. In N. Heintze and P. Sestoft, editors, *Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, pages 15–26.

Shen, L., Sarkar, A., and Och, F. J. (2004). Discriminative reranking for machine translation. In *Proc. HLT-NAACL 2004*.

Shieber, S., Schabes, Y., and Pereira, F. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, **24**, 3–36.

Sutton, C. and McCallum, A. (2005). Joint parsing and semantic role labeling. In *Proc. CoNLL 2005*.

Taskar, B., Klein, D., Collins, M., Koller, D., and Manning, C. (2004). Max-margin parsing. In *Proc. EMNLP 2004*.

Wellner, B., McCallum, A., Peng, F., and Hay, M. (2004). An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proc. UAI 2004*.

Machine Translation as Lexicalized Parsing with Hooks

Liang Huang
Dept. of Computer & Information Science
University of Pennsylvania
Philadelphia, PA 19104

Hao Zhang and **Daniel Gildea**
Computer Science Department
University of Rochester
Rochester, NY 14627

Abstract

We adapt the "hook" trick for speeding up bilexical parsing to the decoding problem for machine translation models that are based on combining a synchronous context free grammar as the translation model with an n-gram language model. This dynamic programming technique yields lower complexity algorithms than have previously been described for an important class of translation models.

1 Introduction

In a number of recently proposed synchronous grammar formalisms, machine translation of new sentences can be thought of as a form of parsing on the input sentence. The parsing process, however, is complicated by the interaction of the context-free translation model with an m-gram[1] language model in the output language. While such formalisms admit dynamic programming solutions having polynomial complexity, the degree of the polynomial is prohibitively high.

In this paper we explore parallels between translation and monolingual parsing with lexicalized grammars. Chart items in translation must be augmented with words from the output language in order to capture language model state. This can be thought of as a form of lexicalization with some similarity to that of head-driven lexicalized grammars, despite being unrelated to any notion of syntactic head. We show that techniques for parsing with lexicalized grammars can be adapted to the translation problem, reducing the complexity of decoding with an inversion transduction grammar and a bigram language model from $O(n^7)$ to $O(n^6)$. We present background on this translation model as well as the use of the technique in bilexicalized parsing before describing the new algorithm in detail. We then extend the algorithm to general m-gram language models, and to general synchronous context-free grammars for translation.

[1] We speak of m-gram language models to avoid confusion with n, which here is the length of the input sentence for translation.

2 Machine Translation using Inversion Transduction Grammar

The Inversion Transduction Grammar (ITG) of Wu (1997) is a type of context-free grammar (CFG) for generating two languages synchronously. To model the translational equivalence within a sentence pair, ITG employs a synchronous rewriting mechanism to relate two sentences recursively. To deal with the syntactic divergence between two languages, ITG allows the inversion of rewriting order going from one language to another at any recursive level. ITG in Chomsky normal form consists of unary production rules that are responsible for generating word pairs:

$$X \rightarrow e/f$$
$$X \rightarrow e/\epsilon$$
$$X \rightarrow \epsilon/f$$

where e is a source language word, f is a foreign language word, and ϵ means the null token, and binary production rules in two forms that are responsible for generating syntactic subtree pairs:

$$X \rightarrow [YZ]$$

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 65–73,
Vancouver, October 2005.

and

$$X \rightarrow \langle Y Z \rangle$$

The rules with square brackets enclosing the right-hand side expand the left-hand side symbol into the two symbols on the right-hand side in the same order in the two languages, whereas the rules with angled brackets expand the left hand side symbol into the two right-hand side symbols in reverse order in the two languages. The first class of rules is called straight rule. The second class of rules is called inverted rule.

One special case of 2-normal ITG is the so-called Bracketing Transduction Grammar (BTG) which has only one nonterminal A and two binary rules

$$A \rightarrow [AA]$$

and

$$A \rightarrow \langle AA \rangle$$

By mixing instances of the inverted rule with those of the straight rule hierarchically, BTG can meet the alignment requirements of different language pairs. There exists a more elaborate version of BTG that has 4 nonterminals working together to guarantee the property of one-to-one correspondence between alignments and synchronous parse trees. Table 1 lists the rules of this BTG. In the discussion of this paper, we will consider ITG in 2-normal form.

By associating probabilities or weights with the bitext production rules, ITG becomes suitable for weighted deduction over bitext. Given a sentence pair, searching for the Viterbi synchronous parse tree, of which the alignment is a byproduct, turns out to be a two-dimensional extension of PCFG parsing, having time complexity of $O(n^6)$, where n is the length of the English string and the foreign language string. A more interesting variant of parsing over bitext space is the asymmetrical case in which only the foreign language string is given so that Viterbi parsing involves finding the English string "on the fly". The process of finding the source string given its target counterpart is decoding. Using ITG, decoding is a form of parsing.

2.1 ITG Decoding

Wu (1996) presented a polynomial-time algorithm for decoding ITG combined with an m-gram language model. Such language models are commonly used in noisy channel models of translation, which find the best English translation e of a foreign sentence f by finding the sentence e that maximizes the product of the translation model $P(f|e)$ and the language model $P(e)$.

It is worth noting that since we have specified ITG as a joint model generating both e and f, a language model is not theoretically necessary. Given a foreign sentence f, one can find the best translation e^*:

$$\begin{aligned} e^* &= \operatorname*{argmax}_e P(e, f) \\ &= \operatorname*{argmax}_e \sum_q P(e, f, q) \end{aligned}$$

by approximating the sum over parses q with the probability of the Viterbi parse:

$$e^* = \operatorname*{argmax}_e \max_q P(e, f, q)$$

This optimal translation can be computed in using standard CKY parsing over f by initializing the chart with an item for each possible translation of each foreign word in f, and then applying ITG rules from the bottom up.

However, ITG's independence assumptions are too strong to use the ITG probability alone for machine translation. In particular, the context-free assumption that each foreign word's translation is chosen independently will lead to simply choosing each foreign word's single most probable English translation with no reordering. In practice it is beneficial to combine the probability given by ITG with a local m-gram language model for English:

$$e^* = \operatorname*{argmax}_e \max_q P(e, f, q) P_{lm}(e)^\alpha$$

with some constant language model weight α. The language model will lead to more fluent output by influencing both the choice of English words and the reordering, through the choice of straight or inverted rules. While the use of a language model complicates the CKY-based algorithm for finding the best translation, a dynamic programming solution is still possible. We extend the algorithm by storing in each chart item the English *boundary words* that will affect the m-gram probabilities as the item's English string is concatenated with the string from an adjacent item. Due to the locality of m-gram language

Structural Rules			*Lexical Rules*
$S \rightarrow A$ $S \rightarrow B$ $S \rightarrow C$	$A \rightarrow [AB]$ $A \rightarrow [BB]$ $A \rightarrow [CB]$ $A \rightarrow [AC]$ $A \rightarrow [BC]$ $A \rightarrow [CC]$	$B \rightarrow \langle AA \rangle$ $B \rightarrow \langle BA \rangle$ $B \rightarrow \langle CA \rangle$ $B \rightarrow \langle AC \rangle$ $B \rightarrow \langle BC \rangle$ $B \rightarrow \langle CC \rangle$	$C \rightarrow e_i/f_j$ $C \rightarrow \epsilon/f_j$ $C \rightarrow e_i/\epsilon$

Table 1: Unambiguous BTG

model, only $m-1$ boundary words need to be stored to compute the new m-grams produced by combining two substrings. Figure 1 illustrates the combination of two substrings into a larger one in straight order and inverted order.

3 Hook Trick for Bilexical Parsing

A traditional CFG generates words at the bottom of a parse tree and uses nonterminals as abstract representations of substrings to build higher level tree nodes. Nonterminals can be made more specific to the actual substrings they are covering by associating a representative word from the nonterminal's yield. When the maximum number of lexicalized nonterminals in any rule is two, a CFG is bilexical. A typical bilexical CFG in Chomsky normal form has two types of rule templates:

$$A[h] \rightarrow B[h]C[h']$$

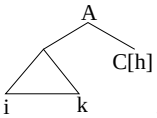
or

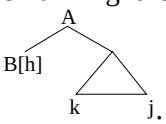
$$A[h] \rightarrow B[h']C[h]$$

depending on which child is the head child that agrees with the parent on head word selection. Bilexical CFG is at the heart of most modern statistical parsers (Collins, 1997; Charniak, 1997), because the statistics associated with word-specific rules are more informative for disambiguation purposes. If we use $A[i, j, h]$ to represent a lexicalized constituent, $\beta(\cdot)$ to represent the Viterbi score function applicable to any constituent, and $P(\cdot)$ to represent the rule probability function applicable to any rule, Figure 2 shows the equation for the dynamic programming computation of the Viterbi parse. The two terms of the outermost $\max$ operator are symmetric cases for heads coming from left and right. Containing five free variables i,j,k,h',h, ranging over 1 to n, the length of input sentence, both terms can be instantiated in n^5 possible ways, implying that the complexity of the parsing algorithm is $O(n^5)$.

Eisner and Satta (1999) pointed out we don't have to enumerate k and h' simultaneously. The trick, shown in mathematical form in Figure 2 (bottom) is very simple. When maximizing over h', j is irrelevant. After getting the intermediate result of maximizing over h', we have one less free variable than before. Throughout the two steps, the maximum number of interacting variables is 4, implying that the algorithmic complexity is $O(n^4)$ after binarizing the factors cleverly. The intermediate result

$$\max_{h',B}[\beta(B[i,k,h']) \cdot P(A[h] \rightarrow B[h']C[h])]$$

can be represented pictorially as 

. The same trick works for the second $\max$ term in Equation 1. The intermediate result coming from binarizing the second term can be visualized as 

. The shape of the intermediate results gave rise to the nickname of "hook". Melamed (2003) discussed the applicability of the hook trick for parsing bilexical multitext grammars. The analysis of the hook trick in this section shows that it is essentially an algebraic manipulation. We will formulate the ITG Viterbi decoding algorithm in a dynamic programming equation in the following section and apply the same algebraic manipulation to produce hooks that are suitable for ITG decoding.

4 Hook Trick for ITG Decoding

We start from the bigram case, in which each decoding constituent keeps a left boundary word and

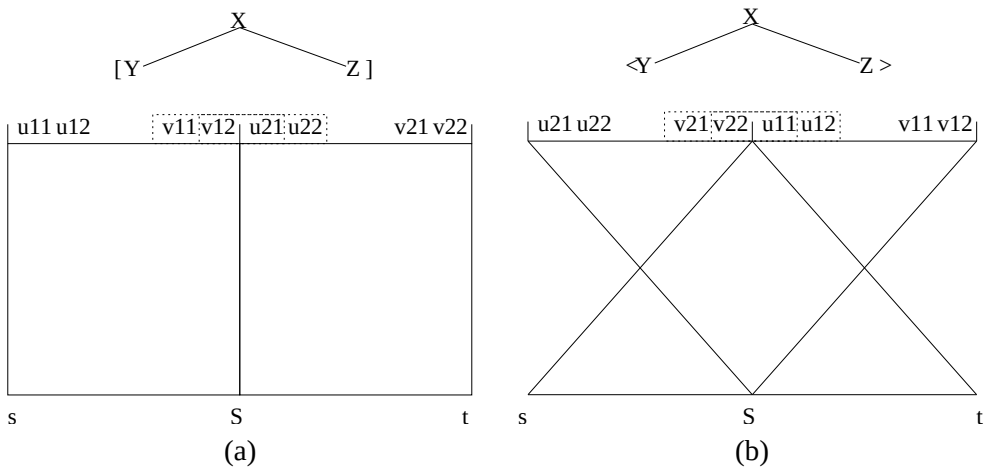


Figure 1: ITG decoding using 3-gram language model. Two boundary words need to be kept on the left (u) and right (v) of each constituent. In (a), two constituents Y and Z spanning substrings s, S and S, t of the input are combined using a straight rule $X \rightarrow [YZ]$. In (b), two constituents are combined using a inverted rule $X \rightarrow \langle YZ \rangle$. The dashed line boxes enclosing three words are the trigrams produced from combining two substrings.

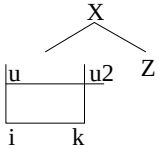
$$\beta(A[i,j,h]) = \max \left\{ \begin{array}{c} \max\limits_{k,h',B,C} \left[\beta(B[i,k,h']) \cdot \beta(C[k,j,h]) \cdot P(A[h] \rightarrow B[h']C[h]) \right], \\ \max\limits_{k,h',B,C} \left[\beta(B[i,k,h]) \cdot \beta(C[k,j,h']) \cdot P(A[h] \rightarrow B[h]C[h']) \right] \end{array} \right\} \quad (1)$$

$$\begin{aligned} & \max_{k,h',B,C} \left[\beta(B[i,k,h']) \cdot \beta(C[k,j,h]) \cdot P(A[h] \rightarrow B[h']C[h]) \right] \\ = \; & \max_{k,C} \left[\max_{h',B} \left[\beta(B[i,k,h']) \cdot P(A[h] \rightarrow B[h']C[h]) \right] \cdot \beta(C[k,j,h]) \right] \end{aligned}$$

Figure 2: Equation for bilexical parsing (top), with an efficient factorization (bottom)

a right boundary word. The dynamic programming equation is shown in Figure 3 (top) where i,j,k range over 1 to n, the length of input foreign sentence, and u,v,v_1,u_2 (or u,v,v_2,u_1) range over 1 to V, the size of English vocabulary. Usually we will constrain the vocabulary to be a subset of words that are probable translations of the foreign words in the input sentence. So V is proportional to n. There are seven free variables related to input size for doing the maximization computation. Hence the algorithmic complexity is $O(n^7)$.

The two terms in Figure 3 (top) within the first level of the $\max$ operator, corresponding to straight rules and inverted rules, are analogous to the two terms in Equation 1. Figure 3 (bottom) shows how to decompose the first term; the same method applies to the second term. Counting the free variables enclosed in the innermost $\max$ operator, we get five: i, k, u, v_1, and u_2. The decomposition eliminates one free variable, v_1. In the outermost level, there are six free variables left. The maximum number of interacting variables is six overall. So, we reduced the complexity of ITG decoding using bigram language model from $O(n^7)$ to $O(n^6)$.



The hooks that we have built for decoding with a bigram language model turn out to be similar to the hooks for bilexical parsing if we focus on the two boundary words v_1 and u_2 (or v_2 and u_1)

$$\beta(X[i,j,u,v]) \quad = \quad \max \left\{ \begin{array}{l} \max\limits_{k,v_1,u_2,Y,Z} \left[\begin{array}{l} \beta(Y[i,k,u,v_1]) \cdot \beta(Z[k,j,u_2,v]) \\ \cdot P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \end{array} \right], \\ \max\limits_{k,v_2,u_1,Y,Z} \left[\begin{array}{l} \beta(Y[i,k,u_1,v]) \cdot \beta(Z[k,j,u,v_2]) \\ \cdot P(X \rightarrow \langle YZ \rangle) \cdot bigram(v_2,u_1) \end{array} \right] \end{array} \right\} \tag{2}$$

$$\begin{aligned} & \max_{k,v_1,u_2,Y,Z} \Big[\beta(Y[i,k,u,v_1]) \cdot \beta(Z[k,j,u_2,v]) \cdot P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \Big] \\ = & \max_{k,u_2,Z} \Big[\max_{v_1,Y} \Big[\beta(Y[i,k,u,v_1]) \cdot P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \Big] \cdot \beta(Z[k,j,u_2,v]) \Big] \end{aligned}$$

Figure 3: Equation for ITG decoding (top), with an efficient factorization (bottom)

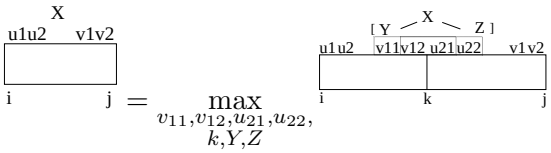
that are interacting between two adjacent decoding constituents and relate them with the h' and h that are interacting in bilexical parsing. In terms of algebraic manipulation, we are also rearranging three factors (ignoring the non-lexical rules), trying to reduce the maximum number of interacting variables in any computation step.

4.1 Generalization to m-gram Cases

In this section, we will demonstrate how to use the hook trick for trigram decoding which leads us to a general hook trick for any m-gram decoding case.

We will work only on straight rules and use icons of constituents and hooks to make the equations easier to interpret.

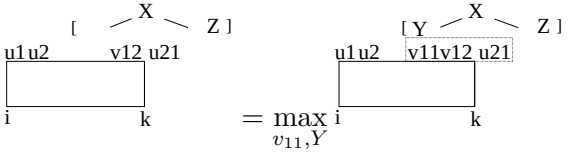
The straightforward dynamic programming equation is:



(3)

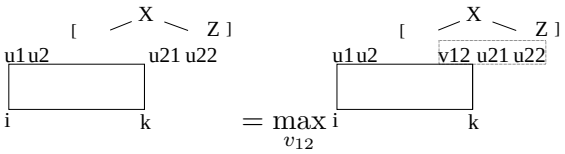
By counting the variables that are dependent on input sentence length on the right hand side of the equation, we know that the straightforward algorithm's complexity is $O(n^{11})$. The maximization computation is over four factors that are dependent on n: $\beta(Y[i,k,u_1,u_2,v_{11},v_{12}])$, $\beta(Z[k,j,u_{21},u_{22},v_1,v_2])$, $trigram(v_{11},v_{12},u_{21})$, and $trigram(v_{12},u_{21},u_{22})$. As before, our goal is to cleverly bracket the factors.

By bracketing $trigram(v_{11},v_{12},u_{21})$ and $\beta(Y[i,k,u_1,u_2,v_{11},v_{12}])$ together and maximizing over v_{11} and Y, we can build the the level-1 hook:

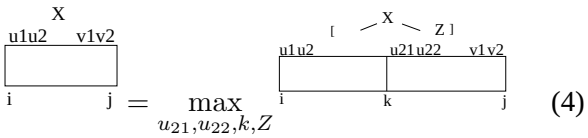


The complexity is $O(n^7)$.

Grouping the level-1 hook and $trigram(v_{12},u_{21},u_{22})$, maximizing over v_{12}, we can build the level-2 hook:



The complexity is $O(n^7)$. Finally, we can use the level-2 hook to combine with $Z[k,j,u_{21},u_{22},v_1,v_2]$ to build $X[i,j,u_1,u_2,v_1,v_2]$. The complexity is $O(n^9)$ after reducing v_{11} and v_{12} in the first two steps.



(4)

Using the hook trick, we have reduced the complexity of ITG decoding using bigrams from $O(n^7)$ to $O(n^6)$, and from $O(n^{11})$ to $O(n^9)$ for trigram

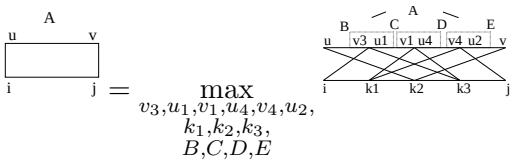
case. We conclude that for m-gram decoding of ITG, the hook trick can change the the time complexity from $O(n^{3+4(m-1)})$ to $O(n^{3+3(m-1)})$. To get an intuition of the reduction, we can compare Equation 3 with Equation 4. The variables v_{11} and v_{12} in Equation 3, which are independent of v_1 and v_2 for maximizing the product have been concealed under the level-2 hook in Equation 4. In general, by building $m-1$ intermediate hooks, we can reduce $m-1$ free variables in the final combination step, hence having the reduction from $4(m-1)$ to $3(m-1)$.

5 Generalization to Non-binary Bitext Grammars

Although we have presented our algorithm as a decoder for the binary-branching case of Inversion Transduction Grammar, the same factorization technique can be applied to more complex synchronous grammars. In this general case, items in the dynamic programming chart may need to represent non-contiguous span in either the input or output language. Because synchronous grammars with increasing numbers of children on the right hand side of each production form an infinite, non-collapsing hierarchy, there is no upper bound on the number of discontinuous spans that may need to be represented (Aho and Ullman, 1972). One can, however, choose to factor the grammar into binary branching rules in one of the two languages, meaning that discontinuous spans will only be necessary in the other language.

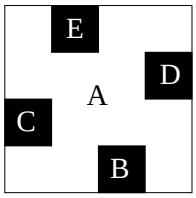
If we assume m is larger than 2, it is likely that the language model combinations dominate computation. In this case, it is advantageous to factor the grammar in order to make it binary in the output language, meaning that the subrules will only need to represent adjacent spans in the output language. Then the hook technique will work in the same way, yielding $O(n^{2(m-1)})$ distinct types of items with respect to language model state, and $3(m-1)$ free indices to enumerate when combining a hook with a complete constituent to build a new item. However, a larger number of indices pointing into the input language will be needed now that items can cover discontinuous spans. If the grammar factorization yields rules with at most R spans in the input language, there may be $O(n^{2R})$ distinct types of chart items with respect to the input language, because each span has an index for its beginning and ending points in the input sentence. Now the upper bound of the number of free indices with respect to the input language is $2R+1$, because otherwise if one rule needs $2R+2$ indices, say $i_1, \cdots, i_{2R+2}$, then there are $R+1$ spans $(i_1, i_2), \cdots, (i_{2R+1}, i_{2R+2})$, which contradicts the above assumption. Thus the time complexity at the input language side is $O(n^{2R+1})$, yielding a total algorithmic complexity of $O(n^{3(m-1)+(2R+1)})$.

To be more concrete, we will work through a 4-ary translation rule, using a bigram language model. The standard DP equation is:



(5)

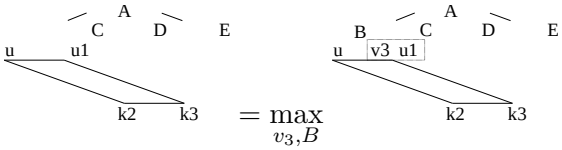
This 4-ary rule is a representative difficult case. The underlying alignment pattern for this rule is as follows:



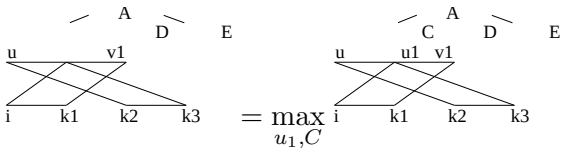
It is a rule that cannot be binarized in the bitext space using ITG rules. We can only binarize it in one dimension and leave the other dimension having discontinuous spans. Without applying binarization and hook trick, decoding parsing with it according to Equation 5 requires time complexity of $O(n^{13})$.

However, we can build the following partial constituents and hooks to do the combination gradually.

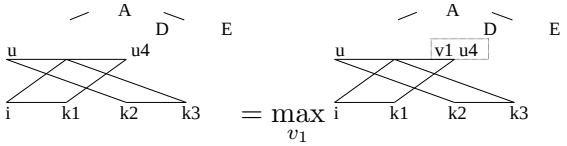
The first step finishes a hook by consuming one bigram. Its time complexity is $O(n^5)$:



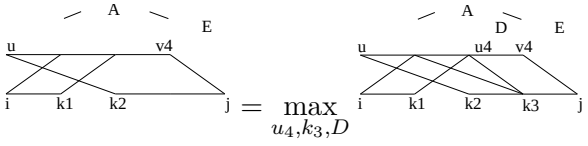
The second step utilizes the hook we just built and builds a partial constituent. The time complexity is $O(n^7)$:



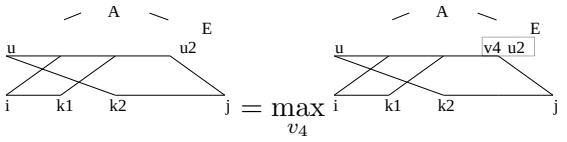
By "eating" another bigram, we build the second hook using $O(n^7)$:



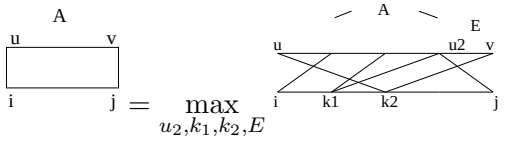
We use the last hook. This step has higher complexity: $O(n^8)$:



The last bigram involved in the 4-ary rule is completed and leads to the third hook, with time complexity of $O(n^7)$:

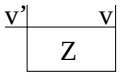


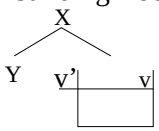
The final combination is $O(n^7)$:



The overall complexity has been reduced to $O(n^8)$ after using binarization on the output side and using the hook trick all the way to the end. The result is one instance of our general analysis: here $R = 2$, $m = 2$, and $3(m-1) + (2R+1) = 8$.

6 Implementation

The implementation of the hook trick in a practical decoder is complicated by the interaction with pruning. If we build hooks looking for all words in the vocabulary whenever a complete constituent is added to the chart, we will build many hooks that are never used, because partial hypotheses with many of the boundary words specified by the hooks may never be constructed due to pruning. Instead of actively building hooks, which are intermediate results, we can build them only when we need them and then cache them for future use. To make this idea concrete, we sketch the code for bigram integrated decoding using ITG as in Algorithm 1. It is worthy of noting that for clarity we are building hooks in shape of 

, instead of 

as we have been showing in the previous sections. That is, the probability for the grammar rule is multiplied in when a complete constituent is built, rather than when a hook is created. If we choose the original representation, we would have to create both straight hooks and inverted hooks because the straight rules and inverted rules are to be merged with the "core" hooks, creating more specified hooks.

7 Conclusion

By showing the parallels between lexicalization for language model state and lexicalization for syntactic heads, we have demonstrated more efficient algorithms for previously described models of machine translation. Decoding for Inversion Transduction Grammar with a bigram language model can be done in $O(n^6)$ time. This is the same complexity as the ITG alignment algorithm used by Wu (1997) and others, meaning complete Viterbi decoding is possible without pruning for realistic-length sentences. More generally, ITG with an m-gram language model is $O(n^{3+3(m-1)})$, and a synchronous context-free grammar with at most R spans in the input language is $O(n^{3(m-1)+(2R+1)})$. While this improves on previous algorithms, the degree in n is probably still too high for complete search to be practical with such models. The interaction of the hook technique with pruning is an interesting

Algorithm 1 ITGDecode(N_t)

for all s, t such that $0 \leq s < t \leq N_t$ **do**
  for all S such that $s < S < t$ **do**
    $\triangleright$ straight rule
    for all rules $X \rightarrow [YZ] \in G$ **do**
      for all (Y, u_1, v_1) possible for the span of (s, S) **do**
        $\triangleright$ a hook who is on (S, t), nonterminal as Z, and outside expectation being v_1 is required
        if not $exist\_hooks(S, t, Z, v_1)$ **then**
          $build\_hooks(S, t, Z, v_1)$
        end if
        for all v_2 possible for the hooks in (S, t, Z, v_1) **do**
          $\triangleright$ combining a hook and a hypothesis, using straight rule
          $\beta(s, t, X, u_1, v_2) =$
            $\max \left\{ \beta(s, t, X, u_1, v_2),\ \beta(s, S, Y, u_1, v_1) \cdot \beta^{+}(S, t, Z, v_1, v_2) \cdot P(X \rightarrow [YZ]) \right\}$
        end for
      end for
    end for
    $\triangleright$ inverted rule
    for all rules $X \rightarrow \langle YZ \rangle \in G$ **do**
      for all (Z, u_2, v_2) possible for the span of (S, t) **do**
        $\triangleright$ a hook who is on (s, S), nonterminal as Y, and outside expectation being v_2 is required
        if not $exist\_hooks(s, S, Y, v_2)$ **then**
          $build\_hooks(s, S, Y, v_2)$
        end if
        for all v_1 possible for the hooks in (s, S, Y, v_2) **do**
          $\triangleright$ combining a hook and a hypothesis, using inverted rule
          $\beta(s, t, X, u_2, v_1) =$
            $\max \left\{ \beta(s, t, X, u_2, v_1),\ \beta(S, t, Z, u_2, v_2) \cdot \beta^{+}(s, S, Y, v_2, v_1) \cdot P(X \rightarrow \langle YZ \rangle) \right\}$
        end for
      end for
    end for
  end for
end for

routine $build\_hooks(s, t, X, v')$
for all (X, u, v) possible for the span of (s, t) **do**
  $\triangleright$ combining a bigram with a hypothesis
  $\beta^{+}(s, t, X, v', v) =$
    $\max \left\{ \beta^{+}(s, t, X, v', v),\ bigram(v', u) \cdot \beta(s, t, X, u, v) \right\}$
end for

area for future work. Building the chart items with hooks may take more time than it saves if many of the hooks are never combined with complete constituents due to aggressive pruning. However, it may be possible to look at the contents of the chart in order to build only those hooks which are likely to be useful.

References

Aho, Albert V. and Jeffery D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Englewood Cliffs, NJ: Prentice-Hall.

Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 598–603, Menlo Park, August. AAAI Press.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Conference of the Association for Computational Linguistics (ACL-97)*, pages 16–23, Madrid, Spain.

Eisner, Jason and Giorgio Satta. 1999. Effi cient parsing for bilexical context-free grammars and head automaton grammars. In *37th Annual Meeting of the Association for Computational Linguistics*.

Melamed, I. Dan. 2003. Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-03)*, Edmonton.

Wu, Dekai. 1996. A polynomial-time algorithm for statistical machine translation. In *34th Annual Meeting of the Association for Computational Linguistics*.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Treebank Transfer

Martin Jansche
Center for Computational Learning Systems
Columbia University
New York, NY 10027, USA
jansche@acm.org

Abstract

We introduce a method for transferring annotation from a syntactically annotated corpus in a source language to a target language. Our approach assumes only that an (unannotated) text corpus exists for the target language, and does not require that the parameters of the mapping between the two languages are known. We outline a general probabilistic approach based on Data Augmentation, discuss the algorithmic challenges, and present a novel algorithm for sampling from a posterior distribution over trees.

1 Introduction

Annotated corpora are valuable resources for Natural Language Processing (NLP) which often require significant effort to create. Syntactically annotated corpora – *treebanks*, for short – currently exist for a small number of languages; but for the vast majority of the world's languages, treebanks are unavailable and unlikely to be created any time soon.

The situation is especially difficult for dialectal variants of many languages. A prominent example is Arabic: syntactically annotated corpora exist for the common written variety (Modern Standard Arabic or MSA), but the spoken regional dialects have a lower status in written communication and lack annotated resources. This lack of dialect treebanks hampers the development of syntax-based NLP tools, such as parsers, for Arabic dialects.

On the bright side, there exist very large annotated (Maamouri et al., 2003, 2004a,b) corpora for Modern Standard Arabic. Furthermore, unannotated text corpora for the various Arabic dialects can also be assembled from various sources on the Internet. Finally, the syntactic differences between the Arabic dialects and Modern Standard Arabic are relatively minor (compared with the lexical, phonological, and morphological differences). The overall research question is then how to combine and exploit these resources and properties to facilitate, and perhaps even automate, the creation of syntactically annotated corpora for the Arabic dialects.

We describe a general approach to this problem, which we call *treebank transfer*: the goal is to project an existing treebank, which exists in a source language, to a target language which lacks annotated resources. The approach we describe is not tied in any way to Arabic, though for the sake of concreteness one may equate the source language with Modern Standard Arabic and the target language with a dialect such as Egyptian Colloquial Arabic.

We link the two kinds of resources that are available – a treebank for the source language and an unannotated text corpus for the target language – in a generative probability model. Specifically, we construct a joint distribution over source-language trees, target-language trees, as well as parameters, and draw inferences by iterative simulation. This allows us to impute target-language trees, which can then be used to train target-language parsers and other NLP components.

Our approach does not require aligned data, unlike related proposals for transferring annotations from one language to another. For example, Yarowksy and Ngai (2001) consider the transfer of word-level annotation (part-of-speech labels and bracketed NPs). Their approach is based on aligned

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 74–82,
Vancouver, October 2005.

corpora and only transfers annotation, as opposed to generating the raw data plus annotation as in our approach.

We describe the underlying probability model of our approach in Section 2 and discuss issues pertaining to simulation and inference in Section 3. Sampling from the posterior distribution of target-language trees is one of the key problems in iterative simulation for this model. We present a novel sampling algorithm in Section 4. Finally in Section 5 we summarize our approach in its full generality.

2 The Probability Model

Our approach assumes that two kinds of resources are available: a source-language treebank, and a target-language text corpus. This is a realistic assumption, which is applicable to many source-language/target-language pairs. Furthermore, some knowledge of the mapping between source-language syntax and target-language syntax needs to be incorporated into the model. Parallel corpora are not required, but may help when constructing this mapping.

We view the source-language treebank as a sequence of trees $S_1,\ldots,S_n$, and assume that these trees are generated by a common process from a corresponding sequence of latent target-language trees $T_1,\ldots,T_n$. The parameter vector of the process which maps target-language trees to source-language trees will be denoted by Ξ. The mapping itself is expressed as a conditional probability distribution $p(S_i \mid T_i,\Xi)$ over source-language trees. The parameter vector Ξ is assumed to be generated according to a prior distribution $p(\Xi \mid \xi)$ with hyper-parameter ξ, assumed to be fixed and known.

We further assume that each target-language tree T_i is generated from a common language model Λ for the target language, $p(T_i \mid \Lambda)$. For expository reasons we assume that Λ is a bigram language model over the terminal yield (also known as the *fringe*) of T_i. Generalizations to higher-order n-gram models are completely straightforward; more general models that can be expressed as stochastic finite automata are also possible, as discussed in Section 5. Let $t_1,\ldots,t_k$ be the terminal yield of tree T. Then

$$p(T \mid \Lambda) = \Lambda(t_1 \mid \#) \left(\prod_{j=2}^{k} \Lambda(t_j \mid t_{j-1}) \right) \Lambda(\$ \mid t_k),$$

where # marks the beginning of the string and \$ marks the end of the string.

There are two options for incorporating the language model Λ into the overall probability model. In the first case – which we call the *full model* – Λ is generated by an informative prior distribution $p(\Lambda \mid \lambda)$ with hyper-parameter λ. In the second case – the *reduced model* – the language model Λ is fixed.

The structure of the full model is specified graphically in Figure 1. In a directed acyclic graphical model such as this one, we equate vertices with random variables. Directed edges are said to go from a parent to a child node. Each vertex depends directly on all of its parents. Any particular vertex is conditionally independent from all other vertices given its parents, children, and the parents of its children.

The portion of the full model we are interested in is the following factored distribution, as specified by Figure 1:

$$\begin{aligned} &p(S_1,\ldots,S_n,T_1,\ldots,T_n,\Lambda,\Xi \mid \lambda,\xi) \\ &= p(\Lambda \mid \lambda)\, p(\Xi \mid \xi) \prod_{i=1}^{n} p(T_i \mid \Lambda)\, p(S_i \mid T_i,\Xi) \end{aligned} \quad (1)$$

In the reduced model, we drop the leftmost term/vertex, corresponding to the prior for Λ with hyper-parameter λ, and condition on Λ instead:

$$\begin{aligned} &p(S_1,\ldots,S_n,T_1,\ldots,T_n,\Xi \mid \Lambda,\xi) \\ &\qquad = p(\Xi \mid \xi) \prod_{i=1}^{n} p(T_i \mid \Lambda)\, p(S_i \mid T_i,\Xi) \end{aligned} \quad (2)$$

The difference between the full model (1) and the reduced model (2) is that the reduced model assumes that the language model Λ is fixed and will not be informed by the latent target-language trees T_i. This is an entirely reasonable assumption in a situation where the target-language text corpus is much larger than the source-language treebank. This will typically be the case, since it is usually very easy to collect large corpora of unannotated text which exceed the largest existing annotated corpora by several orders of magnitude. When a sufficiently large target-language text corpus is available, Λ is simply a smoothed bigram model which is estimated once from the target-language corpus.

If the target-language corpus is relatively small, then the bigram model Λ can be refined on the basis of the imputed target-language trees. A bigram

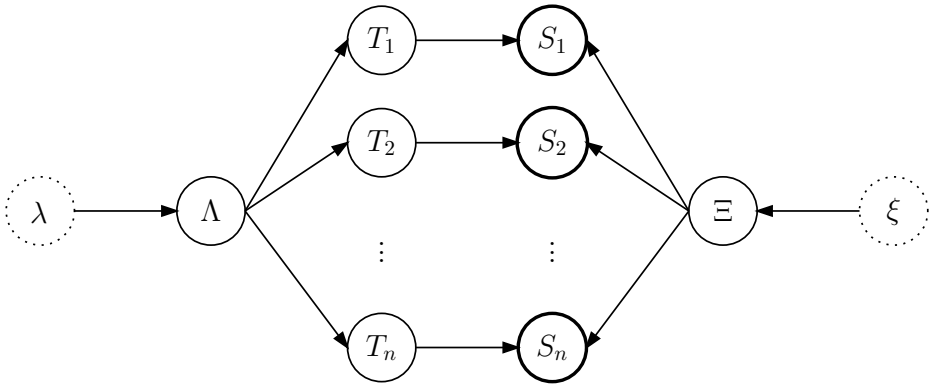


Figure 1: The graphical structure of the full probability model. Bold circles indicate observed variables, dotted circles indicate parameters.

model is simply a discrete collection of multinomial distributions. A simple prior for Λ takes the form of a product of Dirichlet distributions, so that the hyper-parameter λ is a vector of bigram counts. In the full model (1), we assume λ is fixed and set it to the observed bigram counts (plus a constant) in the target-language text corpus. This gives us an informative prior for Λ. If the bigram counts are sufficiently large, Λ will be fully determined by this informative prior distribution, and the reduced model (2) can be used instead.

By contrast, usually very little is known a priori about the syntactic transfer model Ξ. Instead Ξ needs to be estimated from data. We assume that Ξ too is a discrete collection of multinomial distributions, governed by Dirichlet priors. However, unlike in the case of Λ, the priors for Ξ are noninformative. This is not a problem, since a lot of information about the target language is provided by the language model Λ.

As one can see in Figure 1 and equation (1), the overall probability model constrains the latent target-language trees T_i in two ways: From the left, the language model Λ serves as a prior distribution over target-language trees. On the one hand, Λ is an informative prior, based on large bigram counts obtained from the target-language text corpus; on the other hand, it only informs us about the fringe of the target-language trees and has very little directly to say about their syntactic structure. From the right, the observed source-language trees constrain the latent target-language trees in a complementary fashion. Each target-language tree T_i gives rise to a corresponding source-language tree S_i according to the syntactic transfer mapping Ξ. This mapping is initially known only qualitatively, and comes with a noninformative prior distribution.

Our goal is now to simultaneously estimate the transfer parameter Ξ and impute the latent trees T_i. This is simplified by the following observation: if $T_1, \ldots, T_n$ are known, then finding Ξ is easy; vice versa, if Ξ is known, then finding T_i is easy. Simultaneous inference for Ξ and $T_1, \ldots, T_n$ is possible via Data Augmentation (Tanner and Wong, 1987), or, more generally, Gibbs sampling (Geman and Geman, 1984).

3 Simulation of the Joint Posterior Distribution

We now discuss the simulation of the joint posterior distribution over the latent trees $T_1, \ldots, T_n$, the transfer model parameter Ξ, and the language model parameter Λ. This joint posterior is derived from the overall full probability model (1). Using the reduced model (2) instead of the full model amounts to simply omitting Λ from the joint posterior. We will deal primarily with the more general full model in this section, since the simplification which results in the reduced model will be straightforward.

The posterior distribution we focus on is $p(T_1, \ldots, T_n, \Lambda, \Xi \mid S_1, \ldots, S_n, \lambda, \xi)$, which provides us with information about all the variables of interest, including the latent target-language trees T_i, the syntactic transfer model Ξ, and the target-language

language model Λ. It is possible to simulate this joint posterior distribution using simple sampling-based approaches (Gelfand and Smith, 1990), which are instances of the general Markov-chain Monte Carlo method (see, for example, Liu, 2001).

Posterior simulation proceeds iteratively, as follows. In each iteration we draw the three kinds of random variables – latent trees, language model parameters, and transfer model parameters – from their conditional distributions while holding the values of all other variables fixed. Specifically:

- Initialize Λ and Ξ by drawing each from its prior distribution.
- Iterate the following three steps:
  1. Draw each T_i from its posterior distribution given S_i, Λ, and Ξ.
  2. Draw Λ from its posterior distribution given $T_1, \ldots, T_n$ and λ.
  3. Draw Ξ from its posterior distribution given $S_1, \ldots, S_n$, $T_1, \ldots, T_n$, and ξ.

This simulation converges in the sense that the draws of $T_1, \ldots, T_n$, Λ, and Ξ converge in distribution to the joint posterior distribution over those variables. Further details can be found, for example, in Liu, 2001, as well as the references cited above.

We assume that the bigram model Λ is a family of multinomial distributions, and we write $\Lambda(t_j \mid t_{j-1})$ for the probability of the word t_j following t_{j-1}. Using creative notation, $\Lambda(\cdot \mid t_{j-1})$ can be seen as a multinomial distribution. Its conjugate prior is a Dirichlet distribution whose parameter vector λ_w are the counts of words types occurring immediately after the word type w of t_{j-1}. Under the conventional assumptions of exchangeability and independence, the prior distribution for Λ is just a product of Dirichlet priors. Since we employ a conjugate prior, the posterior distribution of Λ

$$p(\Lambda \mid S_1, \ldots, S_n, T_1, \ldots, T_n, \Xi, \lambda, \xi) = p(\Lambda \mid T_1, \ldots, T_n, \lambda) \quad (3)$$

has the same form as the prior – it is likewise a product of Dirichlet distributions. In fact, for each word type w the posterior Dirichlet density has parameter $\lambda_w + c_w$, where λ_w is the parameter of the prior distribution and c_w is a vector of counts for all word forms appearing immediately after w along the fringe of the imputed trees.

We make similar assumptions about the syntactic transfer model Ξ and its posterior distribution, which is

$$p(\Xi \mid S_1, \ldots, S_n, T_1, \ldots, T_n, \Lambda, \lambda, \xi) = p(\Xi \mid S_1, \ldots, S_n, T_1, \ldots, T_n, \xi). \quad (4)$$

In particular, we assume that syntactic transfer involves only multinomial distributions, so that the prior and posterior for Ξ are products of Dirichlet distributions. This means that sampling Λ and Ξ from their posterior distributions is straightforward.

The difficult part is the first step in each scan of the Gibbs sampler, which involves sampling each target-language latent tree from the corresponding posterior distribution. For a particular tree T_j, the posterior takes the following form:

$$\begin{aligned} p(T_j \mid S_1, \ldots, S_n, T_1, \ldots, T_{j-1}, T_{j+1}, \ldots, T_n, \Lambda, \Xi, \lambda, \xi) \\ = p(T_j \mid S_j, \Lambda, \Xi) = \frac{p(T_j, S_j \mid \Lambda, \Xi)}{\sum_{T_j} p(T_j, S_j \mid \Lambda, \Xi)} \\ \propto p(T_j \mid \Lambda)\, p(S_j \mid T_j, \Xi) \end{aligned} \quad (5)$$

The next section discusses sampling from this posterior distribution in the context of a concrete example and presents an algorithmic solution.

4 Sampling from the Latent Tree Posterior

We are faced with the problem of sampling T_j from its posterior distribution, which is proportional to the product of its language model prior $p(T_j \mid \Lambda)$ and transfer model likelihood $p(S_j \mid T_j, \Xi)$. Rejection sampling using the prior as the proposal distribution will not work, for two reasons: first, the prior is only defined on the yield of a tree and there are potentially very many tree structures with the same fringe; second, even if the first problem could be overcome, it is unlikely that a random draw from an n-gram prior would result in a target-language tree that corresponds to a particular source-language tree, as the prior has no knowledge of the source-language tree.

Fortunately, efficient direct sampling from the latent tree posterior is possible, under one very reasonable assumption: the set of all target-language trees which map to a given source-language tree S_j

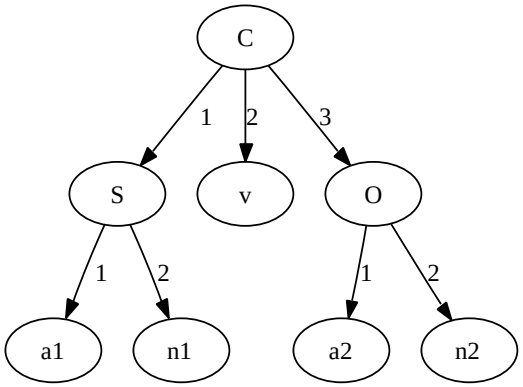


Figure 2: Syntax tree illustrating SVO constituent order within a sentence, and prenominal adjectives within noun phrases.

should be finite and representable as a packed forest. More specifically, we assume that there is a compact (polynomial space) representation of potentially exponentially many trees. Moreover, each tree in the packed forest has an associated weight, corresponding to its likelihood under the syntactic transfer model.

If we rescale the weights of the packed forest so that it becomes a normalized probabilistic context-free grammar (PCFG), we can sample from this new distribution (corresponding to the normalized likelihood) efficiently. For example, it is then possible to use the PCFG as a proposal distribution for rejection sampling.

However, we can go even further and sample from the latent tree posterior directly. The key idea is to intersect the packed forest with the *n*-gram language model and then to normalize the resulting augmented forest. The intersection operation is a special case of the intersection construction for context-free grammars and finite automata (Bar-Hillel et al., 1961, pp. 171–172). We illustrate it here for a bigram language model.

Consider the tree in Figure 2 and assume it is a source-language tree, whose root is a clause (C) which consists of a subject (S), verb (v) and object (O). The subject and object are noun phrases consisting of an adjective (a) and a noun (n). For simplicity, we treat the part-of-speech labels (a, n, v) as terminal symbols and add numbers to distinguish multiple occurrences. The syntactic transfer model is stated as a conditional probability distribution over source-language trees conditional on target language trees. Syntactic transfer amounts to independently changing the order of the subject, verb, and object, and changing the order of adjectives and nouns, for example as follows:

$$
\begin{aligned}
p(SvO \mid SvO) &= \Xi_1 \\
p(SOv \mid SvO) &= (1-\Xi_1)\,\Xi_2 \\
p(vSO \mid SvO) &= (1-\Xi_1)\,(1-\Xi_2)
\end{aligned}
$$

$$
\begin{aligned}
p(SvO \mid SOv) &= \Xi_3 \\
p(SOv \mid SOv) &= (1-\Xi_3)\,\Xi_4 \\
p(vSO \mid SOv) &= (1-\Xi_3)\,(1-\Xi_4)
\end{aligned}
$$

$$
\begin{aligned}
p(SvO \mid vSO) &= \Xi_5 \\
p(SOv \mid vSO) &= (1-\Xi_5)\,\Xi_6 \\
p(vSO \mid vSO) &= (1-\Xi_5)\,(1-\Xi_6)
\end{aligned}
$$

$$
\begin{aligned}
p(an \mid an) &= \Xi_7 \\
p(na \mid an) &= 1-\Xi_7
\end{aligned}
$$

$$
\begin{aligned}
p(an \mid na) &= \Xi_8 \\
p(na \mid na) &= 1-\Xi_8
\end{aligned}
$$

Under this transfer model, the likelihood of a target-language tree $[_A\, v\, [_S\, a_1\, n_1]\, [_O\, n_2\, a_2]]$ corresponding to the source-language tree shown in Figure 2 is $\Xi_5 \times \Xi_7 \times \Xi_8$. It is easy to construct a packed forest of all target-language trees with non-zero likelihood that give rise to the source-language tree in Figure 2. Such a forest is shown in Figure 3. Forest nodes are shown as ellipses, choice points as rectangles connected by dashed lines. A forest node is to be understood as an (unordered) disjunction of the choice points directly underneath it, and a choice point as an (ordered, as indicated by numbers) conjunction of the forest nodes directly underneath it. In other words, a packed forest can be viewed as an acyclic and-or graph, where choice points represent and-nodes (whose children are ordered). As a simplifying convention, for nodes that dominate a single choice node, that choice node is not shown. The forest in Figure 3 represents *SvO*, *SOv*, and *vSO* permutations at the sentence level and *an*, *na* permutations below the two noun phrases. The twelve overall permutations are represented compactly in terms of two choices for the subject, two choices for the object, and three choices for the root clause.

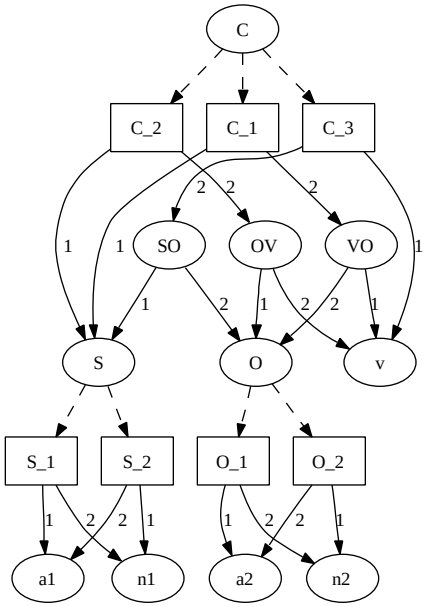


Figure 3: Plain forest of target-language trees that can correspond to the source-language tree in Figure 2.

We intersect/compose the packed forest with the bigram language model Λ by augmenting each node in the forest with a left context word and a right peripheral word: a node N is transformed into a triple (a,N,b) that dominates those trees which N dominates in the original forest and which can occur after a word a and end with a word b. The algorithm is roughly[1] as shown in Figure 5 for binary branching forests; it requires memoization (not shown) to be efficient. The generalization to forests with arbitrary branching factors is straightforward, but the presentation of that algorithm less so. At the root level, we call `forest_composition` with a left context of # (indicating the start of the string) and add dummy nodes of the form $(a,\$,\$)$ (indicating the end of the string). Further details can be found in the prototype implementation. Each node in the original forest is augmented with two words; if there are n leaf nodes in the original forest, the total number of nodes in the augmented forest will be at most n^2 times larger than in the original forest. This means that the compact encoding property of the packed forest (exponentially many trees can be represented in polynomial space) is preserved by the composition algorithm. An example of composing a packed forest with a bigram language model appears in Figure 4, which shows the forest that results from composing the forest in Figure 3 with a bigram language model.

The result of the composition is an augmented forest from which sampling is almost trivial. The first thing we have to do is to recursively propagate weights from the leaves upwards to the root of the forest and associate them with nodes. In the non-recursive case of leaf nodes, their weights are provided by the bigram score of the augmented forest: observe that leaves in the augmented forest have labels of the form (a,b,b), where a and b are terminal symbols, and a represents the immediately preceding left context. The score of such a leaf is simply $\Lambda(b \mid a)$. There are two recursive cases: For choice nodes (and-nodes), their weight is the product of the weights of the node's children times a local likelihood score. For example, the node (v,O,n) in Figure 4 dominates a single choice node (not shown, per the earlier conventions), whose weight is $\Lambda(a \mid v)\,\Lambda(n \mid a)\,\Xi_7$. For other forest nodes (or-nodes), their weight is the sum of the weights of the node's children (choice nodes).

Given this very natural weight-propagation algorithm (and-nodes correspond to multiplication, or-nodes to summation), it is clear that the weight of the root node is the sum total of the weights of all trees in the forest, where the weight of a tree is the prod-

[1] A detailed implementation is available from `http://www.cs.columbia.edu/~jansche/transfer/`.

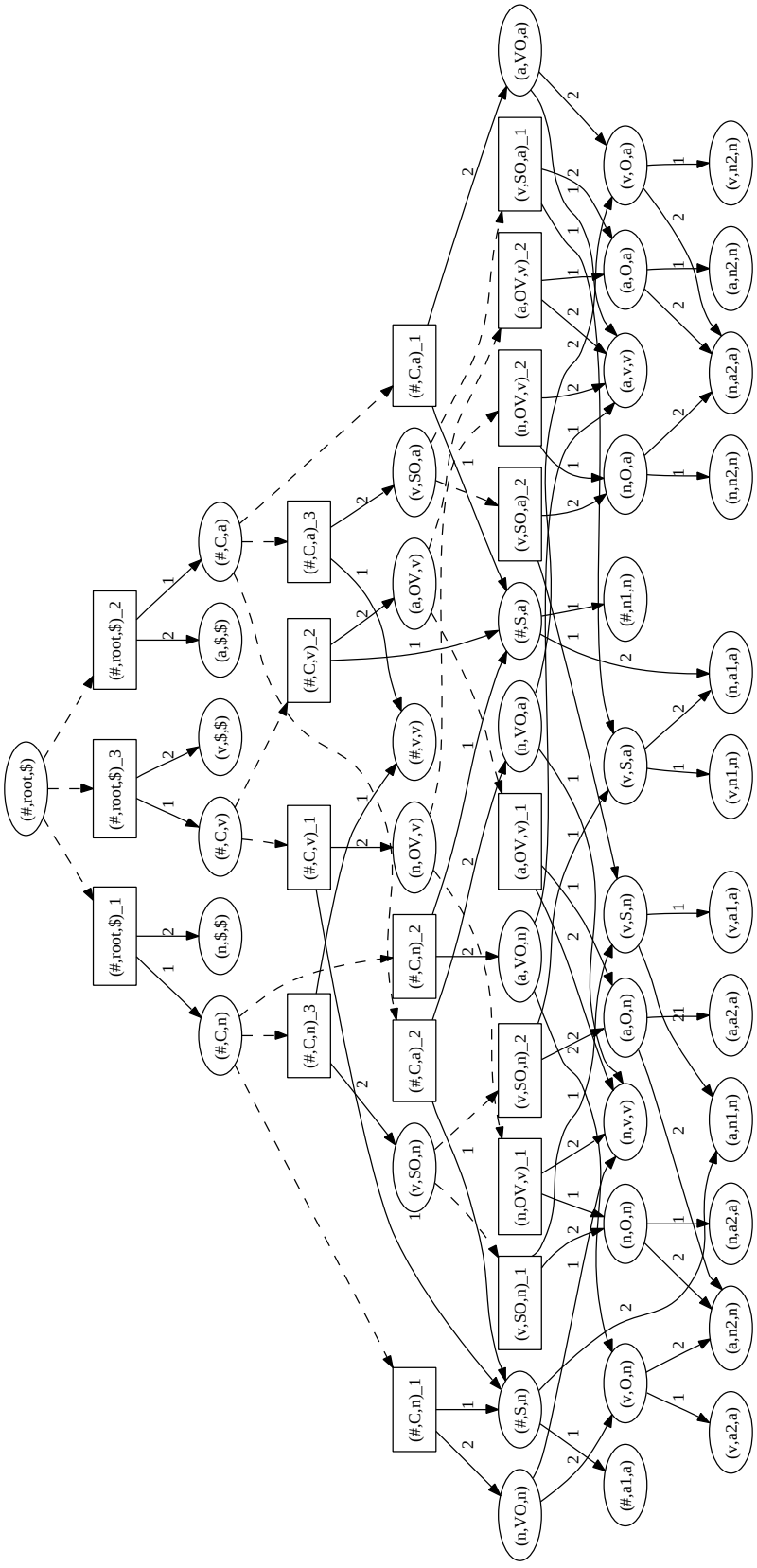


Figure 4: Augmented forest obtained by intersecting the forest in Figure 3 with a bigram language model.

```
forest_composition(N, a):
  if N is a terminal:
    return { (a,N,N) }
  else:
    nodes = {}
    for each (L,R) in N.choices:
      left_nodes <- forest_composition(L, a)
      for each (a,L,b) in left_nodes:
        right_nodes <- forest_composition(R, b)
        for each (b,R,c) in right_nodes:
          new_n = (a,N,c)
          nodes <- nodes + { new_n }
          new_n.choices <- new_n.choices + [((a,L,b), (b,R,c))]
    return nodes
```

Figure 5: Algorithm for computing the intersection of a binary forest with a bigram language model.

uct of the local likelihood scores times the language model score of the tree's terminal yield. We can then associate outgoing normalized weights with the children (choice points) of each or-node, where the probability of going to a particular choice node from a given or-node is equal to the weight of the choice node divided by the weight of the or-node.

This means we have managed to calculate the normalizing constant of the latent tree posterior (5) without enumerating the individual trees in the forest. Normalization ensures that we can sample from the augmented and normalized forest efficiently, by proceeding recursively in a top-down fashion, picking a child of an or-node at random with probability proportional to the outgoing weight of that choice. It is easy to see (by a telescoping product argument) that by multiplying together the probabilities of each such choice we obtain the posterior probability of a latent tree. We thus have a method for sampling latent trees efficiently from their posterior distribution.

The sampling procedure described here is very similar to the lattice-based generation procedure with *n*-gram rescoring developed by Langkilde (2000), and is in fact based on the same intersection construction (Langkilde seems to be unaware that the CFG-intersection construction from (Bar-Hillel et al., 1961) is involved). However, Langkilde is interested in optimization (finding the best tree in the forest), which allows her to prune away less probable trees from the composed forest in a procedure that combines composition, rescoring, and pruning. Alternatively, for a somewhat different but related formulation of the probability model, the sampling method developed by Mark et al. (1992) can be used. However, its efficiency is not well understood.

5 Conclusions

The approach described in this paper was illustrated using very simple examples. The simplicity of the exposition should not obscure the full generality of our approach: it is applicable in the following situations:

- A prior over latent trees is defined in terms of stochastic finite automata.

  We have described the special case of bigram models, and pointed out how our approach will generalize to higher-order *n*-gram models. However, priors are not generally constrained to be *n*-gram models; in fact, any stochastic finite automaton can be employed as a prior, since the intersection of context-free grammars and finite automata is well-defined. However, the intersection construction that appears to be necessary for sampling from the posterior distribution over latent trees may be rather cumbersome when higher-order *n*-gram models or more complex finite automata are used as priors.

- The inverse image of an observed tree under the mapping from latent trees to observed trees can be expressed in terms of a finite context-free language, or equivalently, a packed forest.

The purpose of Gibbs sampling is to simulate the posterior distribution of the unobserved variables in the model. As the sampling procedure converges, knowledge contained in the informative but structurally weak prior Λ is effectively passed to the syntactic transfer model Ξ. Once the sampling procedure has converged to a stationary distribution, we can run it for as many additional iterations as we want and sample the imputed target-language trees. Those trees can then be collected in a treebank, thus creating novel syntactically annotated data in the target language, which can be used for further processing in syntax-based NLP tasks.

Acknowledgements

I would like to thank Steven Abney, the participants of the 2005 Johns Hopkins workshop on Arabic dialect parsing, and the anonymous reviewers for helpful discussions. The usual disclaimers apply.

References

Y. Bar-Hillel, M. Perles, and E. Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14(2):143–172.

Alan E. Gelfand and Adrian F. M. Smith. 1990. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409.

Stuart Geman and Donald Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 6(6):721–741.

Irene Langkilde. 2000. Forest-based statistical sentence generation. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 170–177. ACL Anthology A00-2023.

Jun S. Liu. 2001. *Monte Carlo Strategies in Scientific Computing*. Springer.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Hubert Jin. 2004a. Arabic Treebank: Part 2 v 2.0. Electronic resource, available from LDC.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Hubert Jin. 2004b. Arabic Treebank: Part 3 v 1.0. Electronic resource, available from LDC.

Mohamed Maamouri, Ann Bies, Hubert Jin, and Tim Buckwalter. 2003. Arabic Treebank: Part 1 v 2.0. Electronic resource, available from LDC.

Kevin Mark, Michael Miller, Ulf Grenander, and Steve Abney. 1992. Parameter estimation for constrained context-free language models. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23–26, 1992*, pages 146–149. ACL Anthology H92-1028.

Martin A. Tanner and Wing Hung Wong. 1987. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540.

David Yarowksy and Grace Ngai. 2001. Inducing multilingual POS taggers and NP bracketers via robust projection across aligned corpora. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 200–207.

Lexical and Structural Biases for Function Parsing

Gabriele Musillo
Depts of Linguistics and Computer Science
University of Geneva
2 Rue de Candolle
1211 Geneva 4
Switzerland
musillo4@etu.unige.ch

Paola Merlo
Department of Linguistics
University of Geneva
2 Rue de Candolle
1211 Geneva 4
Switzerland
merlo@lettres.unige.ch

Abstract

In this paper, we explore two extensions to an existing statistical parsing model to produce richer parse trees, annotated with function labels. We achieve significant improvements in parsing by modelling directly the specific nature of function labels, as both expressions of the lexical semantics properties of a constituent and as syntactic elements whose distribution is subject to structural locality constraints. We also reach state-of-the-art accuracy on function labelling. Our results suggest that current statistical parsing methods are sufficiently robust to produce accurate shallow functional or semantic annotation, if appropriately biased.

1 Introduction

Natural language processing methods producing shallow semantic output are starting to emerge as the next step towards successful developments in natural language understanding. Incremental, robust parsing systems will be the core enabling technology for interactive, speech-based question answering and dialogue systems. In recent years, corpora annotated with semantic and function labels have seen the light (Palmer et al., 2005; Baker et al., 1998) and semantic role labelling has taken centre-stage as a challenging new task. State-of-the-art statistical parsers have not yet responded to this challenge.

State-of-the-art statistical parsers trained on the Penn Treebank (PTB) (Marcus et al., 1993) produce trees annotated with bare phrase structure labels (Collins, 1999; Charniak, 2000). The trees of the Penn Treebank, however, are also decorated with function labels, labels that indicate the grammatical and semantic relationship of phrases to each other in the sentence. Figure 1 shows the simplified tree representation with function labels for a sample sentence from the PTB corpus (section 00) *The Government's borrowing authority dropped at midnight Tuesday to 2.80 trillion from 2.87 trillion*. Unlike phrase structure labels, function labels are context-dependent and encode a shallow level of phrasal and lexical semantics, as observed first in (Blaheta and Charniak, 2000). For example, while *the authority* in Figure 1 will always be a Noun Phrase, it could be a subject, as in the example, or an object, as in the sentence *They questioned his authority*, depending on its position in the sentence. To some extent, function labels overlap with semantic role labels as defined in PropBank (Palmer et al., 2005). Table 1

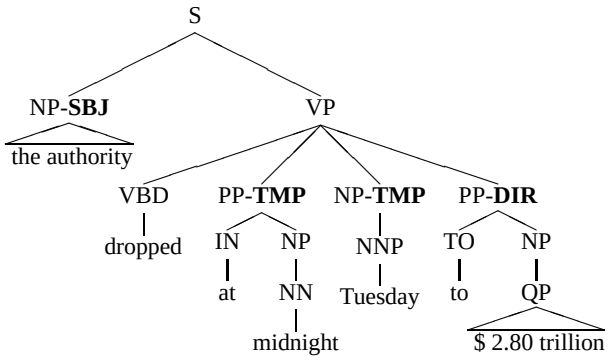


Figure 1: A sample syntactic structure with function labels.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 83–92,
Vancouver, October 2005.

Syntactic Labels		Semantic Labels	
DTV	dative	ADV	adverbial
LGS	logical subject	BNF	benefactive
PRD	predicate	DIR	direction
PUT	compl of *put*	EXT	extent
SBJ	surface subject	LOC	locative
VOC	vocative	MNR	manner
Miscellaneous Labels		NOM	nominal
CLF	*it*-cleft	PRP	purpose or reason
HLN	headline	TMP	temporal
TTL	title	**Topic Labels**	
CLR	closely related	TPC	topicalized

Table 1: Complete set of function labels in the Penn Treebank.

illustrates the complete list of function labels in the Penn Treebank, partitioned into four classes. [1]

Current statistical parsers do not use or output this richer information because performance of the parser usually decreases considerably, since a more complex task is being solved. (Klein and Manning, 2003), for instance report a reduction in parsing accuracy of an unlexicalised PCFG from 77.8% to 72.9% if using function labels in training. (Blaheta, 2004) also reports a decrease in performance when attempting to integrate his function labelling system with a full parser. Conversely, researchers interested in producing richer semantic outputs have concentrated on two-stage systems, where the semantic labelling task is performed on the output of a parser, in a pipeline architecture divided in several stages (Gildea and Jurafsky, 2002; Nielsen and Pradhan, 2004; Xue and Palmer, 2004). See also the common task of (CoNLL, 2004; CoNLL, 2005; Senseval, 2004), where parsing has sometimes not been used and has been replaced by chunking.

In this paper, we present a parser that produces richer output using information available in a corpus incrementally. Specifically, the parser outputs additional labels indicating the function of a constituent in the tree, such as NP-SBJ or PP-TMP in the tree shown in Figure 1.

Following (Blaheta and Charniak, 2000), we concentrate on syntactic and semantic function labels. We will ignore the other two classes, for they do not form natural classes. Like previous work, constituents that do not bear any function label will receive a NULL label. Strictly speaking, this label corresponds to two NULL labels: the SYN-NULL and the SEM-NULL. A node bearing the SYN-NULL label is a node that does not bear any other syntactic label. Analogously, the SEM-NULL label completes the set of semantic labels. Note that both the SYN-NULL label and the SEM-NULL are necessary, since both a syntactic and a semantic label can label a given constituent.

We present work to test the hypothesis that a current statistical parser (Henderson, 2003) can output richer information robustly, that is without any degradation of the parser's accuracy on the original parsing task, by explicitly modelling function labels as the locus where the lexical semantics of the elements in the sentence and syntactic locality domains interact. Briefly, our method consists in augmenting the parser with features and biases that capture both lexical semantics projections and structural regularities underlying the distribution of sequences of function labels in a sentence. We achieve state-of-the-art results both in parsing and function labelling. This result has several consequences.

On the one hand, we show that it is possible to build a single integrated robust system successfully. This is an interesting achievement, as a task combining function labelling and parsing is more complex than simple parsing. While the function of a constituent and its structural position are often correlated, they sometimes diverge. For example, some nominal temporal modifiers occupy an object position without being objects, like *Tuesday* in the tree above. Moreover, given current limited availability of annotated tree banks, this more complex task will have to be solved with the same overall amount of data, aggravating the difficulty of estimating the model's parameters due to sparse data. Solving this more complex problem successfully, then, indicates that the models used are robust. Our results also provide some new insights into the discussion about the necessity of parsing for function or semantic role labelling (Gildea and Palmer, 2002; Punyakanok et al.,

[1] (Blaheta and Charniak, 2000) talk of function *tags*.We will instead use the term function *label*, to indicate function identifiers, as they can decorate any node in the tree. We keep the word *tag* to indicate only those labels that decorate preterminal nodes in a tree – part-of-speech tags – as is standard use.

2005), showing that parsing is beneficial.

On the other hand, function labelling while parsing opens the way to interactive applications that are not possible in a two-stage architecture. Because the parser produces richer output incrementally at the same time as parsing, it can be integrated in speech-based applications, as well as be used for language models. Conversely, output annotated with more informative labels, such as function or semantic labels, underlies all domain-independent question answering (Jijkoun et al., 2004) or shallow semantic interpretation systems (Collins and Miller, 1998; Ge and Mooney, 2005).

2 The Basic Architecture

To achieve the complex task of assigning function labels while parsing, we use a family of statistical parsers, the Simple Synchrony Network (SSN) parsers (Henderson, 2003), which do not make any explicit independence assumptions, and are therefore likely to adapt without much modification to the current problem. This architecture has shown state-of-the-art performance.

SSN parsers comprise two components, one which estimates the parameters of a stochastic model for syntactic trees, and one which searches for the most probable syntactic tree given the parameter estimates. As with many other statistical parsers (Collins, 1999; Charniak, 2000), SSN parsers use a history-based model of parsing. Events in such a model are derivation moves. The set of well-formed sequences of derivation moves in this parser is defined by a Predictive LR pushdown automaton (Nederhof, 1994), which implements a form of left-corner parsing strategy.

This pushdown automaton operates on configurations of the form (Γ, v), where Γ represents the stack, whose right-most element is the top, and v the remaining input. The initial configuration is $(ROOT, w)$ where $ROOT$ is a distinguished non-terminal symbol. The final configuration is $(ROOT, \epsilon)$. Assuming standard notation for context-free grammars (Nederhof, 1994), three derivation moves are defined:

shift

$([B \rightarrow \beta], av) \vdash ([B \rightarrow \beta][A \rightarrow a], v)$
where $A \rightarrow a$ and $B \rightarrow \beta C \gamma$ are productions such that A is a left-corner of C.

project

$([B \rightarrow \beta][A \rightarrow \alpha], v) \vdash$
$([B \rightarrow \beta][D \rightarrow A], v)$
where $A \rightarrow \alpha$, $D \rightarrow A\delta$ and $B \rightarrow \beta C \gamma$ are productions such that D is a left-corner of C.

attach

$([B \rightarrow \beta][A \rightarrow \alpha], v) \vdash ([B \rightarrow \beta A], v)$
where both $A \rightarrow \alpha$ and $B \rightarrow \beta A \gamma$ are productions.

The joint probability of a phrase-structure tree and its terminal yield can be equated to the probability of a finite (but unbounded) sequence of derivation moves. To bound the number of parameters, standard history-based models partition the set of well-formed sequences of transitions into equivalence classes. While such a partition makes the problem of searching for the most probable parse polynomial, it introduces hard independence assumptions: a derivation move only depends on the equivalence class to which its history belongs. SSN parsers, on the other hand, do not state any explicit independence assumptions: they use a neural network architecture, called Simple Synchrony Network (Henderson and Lane, 1998), to induce a finite history representation of an unbounded sequence of moves. The history representation of a parse history $d_1, \ldots, d_{i-1}$, which we denote $h(d_1, \ldots, d_{i-1})$, is assigned to the constituent that is on the top of the stack before the ith move.

The representation $h(d_1, \ldots, d_{i-1})$ is computed from a set f of features of the derivation move d_{i-1} and from a finite set D of recent history representations $h(d_1, \ldots, d_j)$, where $j < i - 1$. Because the history representation computed for the move $i-1$ is included in the inputs to the computation of the representation for the next move i, virtually any information about the derivation history could flow from history representation to history representation and be used to estimate the probability of a derivation move. However, the recency preference exhibited by recursively defined neural networks biases learning towards information which flows through fewer

history representations. (Henderson, 2003) exploits this bias by directly inputting information which is considered relevant at a given step to the history representation of the constituent on the top of the stack before that step. To determine which history representations are input to which others and provide SSNs with a linguistically appropriate inductive bias, the set D includes history representations which are assigned to constituents that are structurally local to a given node on the top of the stack. In addition to history representations, the inputs to $h(d_1, \ldots, d_{i-1})$ include hand-crafted features of the derivation history that are meant to be relevant to the move to be chosen at step i. For each of the experiments reported here, the set D that is input to the computation of the history representation of the derivation moves $d_1, \ldots, d_{i-1}$ includes the most recent history representation of the following nodes: top_i, the node on top of the pushdown stack before the ith move; the left-corner ancestor of top_i (that is, the second top-most node on the parser's stack); the leftmost child of top_i; and the most recent child of top_i, if any. The set of features f includes the last move in the derivation, the label or tag of top_i, the tag-word pair of the most recently shifted word, and the leftmost tag-word pair that top_i dominates. Given the hidden history representation $h(d_1, \cdots, d_{i-1})$ of a derivation, a normalized exponential output function is computed by SSNs to estimate a probability distribution over the possible next derivation moves d_i.[2]

The second component of SSN parsers, which searches for the best derivation given the parameter estimates, implements a severe pruning strategy. Such pruning handles the high computational cost of computing probability estimates with SSNs, and renders the search tractable. The space of possible derivations is pruned in two different ways. The first pruning occurs immediately after a tag-word pair has been pushed onto the stack: only a fixed beam of the 100 best derivations ending in that tag-word pair are expanded. For training, the width of such beam is set to five. A second reduction of the search space prunes the space of possible project or attach derivation moves: the best-first search strategy is applied to the five best alternative decisions only.

[2]The on-line version of Backpropagation is used to train SSN parsing models. It performs the gradient descent with a maximum likelihood objective function and weight decay regularization (Bishop, 1995).

3 Learning Lexical Projection and Locality Domains of Function Labels

Recent approaches to functional or semantic labels are based on two-stage architectures. The first stage selects the elements to be labelled, while the second determines the labels to be assigned to the selected elements. While some of these models are based on full parse trees (Gildea and Jurafsky, 2002; Blaheta, 2004), other methods have been proposed that eschew the need for a full parse (CoNLL, 2004; CoNLL, 2005). Because of the way the problem has been formulated, – as a pipeline of parsing feeding into labelling – specific investigations of the interaction of lexical projections with the relevant structural parsing notions during function labelling has not been studied.

The starting point of our augmentation of SSN models is the observation that the distribution of function labels can be better characterised structurally than sequentially. Function labels, similarly to semantic roles, represent the interface between lexical semantics and syntax. Because they are projections of the lexical semantics of the elements in the sentence, they are projected bottom-up, they tend to appear low in the tree and they are infrequently found on the higher levels of the parse tree, where projections of grammatical, as opposed to lexical, elements usually reside. Because they are the interface level with syntax, function and semantic labels are also subject to distributional constraints that govern syntactic dependencies, especially those governing the distribution of sequences of long distance elements. These relations often correspond to top-down constraints. For example, languages like Italian allow inversion of the subject (the Agent) in transitive sentences, giving rise to a linear sequence where the Theme precedes the Agent (*Mangia la mela Gianni, eats the apple Gianni*). Despite this freedom in the linear order, however, it is never the case that the *structural* positions can be switched. It is a well-attested typological generalisation that one does not find sentences where the subject is a Theme and the object is the Agent. The hierarchical description, then, captures the underlying generalisa-

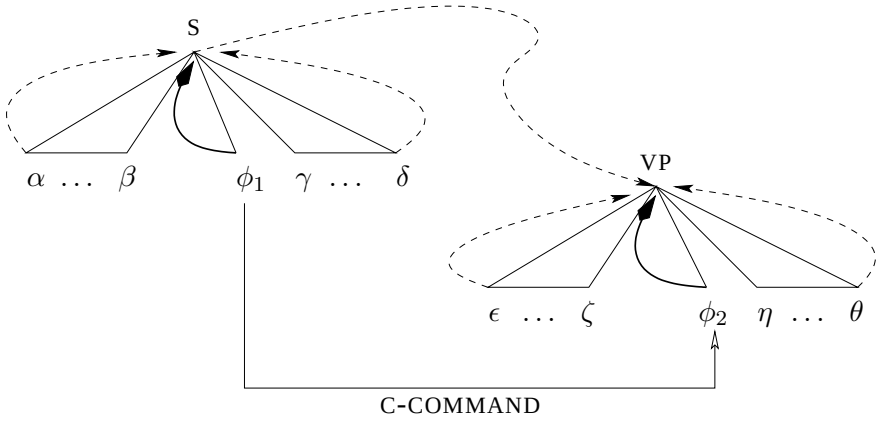


Figure 2: Flow of information in an SSN parser (dashed lines), enhanced by biases specific to function labels to capture the notion of c-command (solid lines).

tion better than a model based on a linear sequence.

In our augmented model, inputs to each history representation are selected according to a linguistically motivated notion of structural locality over which dependencies such as argument structure or subcategorization could be specified. We attempt to capture the sequence and the structural position by indirectly modelling the main definition of syntactic domain, the notion of c-command. Recall that the c-command relation defines the domain of interaction between two nodes in a tree, even if they are not close to each other, provided that the first node dominating one node also dominates the other. This notion of c-command captures both linear and hierarchical constraints and defines the domain in which semantic role labelling applies, as well as many other linguistic operations.

In SSN parsing models, the set D of nodes that are structurally local to a given node on the top of the stack defines the structural distance between this given node and other nodes in the tree. Such a notion of distance determines the number of history representations through which information passes to flow from the representation assigned to a node i to the representation assigned to a node j. By adding nodes to the set D, one can shorten the structural distance between two nodes and enlarge the locality domain over which dependencies can be specified. To capture a locality domain appropriate for function parsing, we include two additional nodes in the set D: the most recent child of top_i labelled with a syntactic function label and the most recent child of top_i labelled with a semantic function label. These additions yield a model that is sensitive to regularities in structurally defined sequences of nodes bearing function labels, within and across constituents. First, in a sequence of nodes bearing function labels within the same constituent – possibly interspersed with nodes not bearing function labels – the structural distance between a node bearing a function label and any of its right siblings is shortened and constant. This effect comes about because the representation of a node bearing a function label is directly input to the representation of its parent, until a farther node with a function label is attached. Second, the distance between a node labelled with a function label and any node that it c-commands is kept constant: since the structural distance between a node $[A \rightarrow \alpha]$ on top of the stack and its left-corner ancestor $[B \rightarrow \beta]$ is constant, the distance between the most recent child node of B labelled with a function label and any child of A is kept constant. This modification of the biases is illustrated in Figure 2.

This figure displays two constituents, S and VP with some of their respective child nodes. The VP node is assumed to be on the top of the parser's stack, and the S one is supposed to be its left-corner ancestor. The directed arcs represent the information that flows from one node to another. According to the original SSN model in (Henderson, 2003), only the information carried over by the leftmost child and the most recent child of a constituent di-

rectly flows to that constituent. In the figure above, only the information conveyed by the nodes α and δ is directly input to the node S. Similarly, the only bottom-up information directly input to the VP node is conveyed by the child nodes ϵ and θ. In both the no-biases and H03 models, nodes bearing a function label such as ϕ_1 and ϕ_2 are not directly input to their respective parents. In our extended model, information conveyed by ϕ_1 and ϕ_2 directly flows to their respective parents. So the distance between the nodes ϕ_1 and ϕ_2, which stand in a c-command relation, is shortened and kept constant.

As well as being subject to locality constraints, functional labels are projected by the lexical semantics of the words in the sentence. We introduce this bottom-up lexical information by fine-grained modelling of function tags in two ways. On the one hand, extending a technique presented in (Klein and Manning, 2003), we split some part-of-speech tags into tags marked with semantic function labels. The labels attached to a non-terminal which appeared to cause the most trouble to the parser in a separate experiment (DIR, LOC, MNR, PRP or TMP) were propagated down to the pre-terminal tag of its head. To affect only labels that are projections of lexical semantics properties, the propagation takes into account the distance of the projection from the lexical head to the label, and distances greater than two are not included. Figure 3 illustrates the result of the tag splitting operation.

On the other hand, we also split the NULL label into mutually exclusive labels. We hypothesize that the label NULL (ie. SYN-NULL and SEM-NULL) is a mixture of types, some of which of semantic nature, such as CLR, which will be more accurately learnt separately. The NULL label was split into the mutually exclusive labels CLR, OBJ and OTHER. Constituents were assigned the OBJ label according to the conditions stated in (Collins, 1999). Roughly, an OBJ non-terminal is an NP, SBAR or S whose parent is an S, VP or SBAR. Any such non-terminal must not bear either syntactic or semantic function labels, or the CLR label. In addition, the first child following the head of a PP is marked with the OBJ label. (For more detail on this lexical semantics projection, see (Merlo and Musillo, 2005).)

We report the effects of these augmentations on parsing results in the experiments described below.

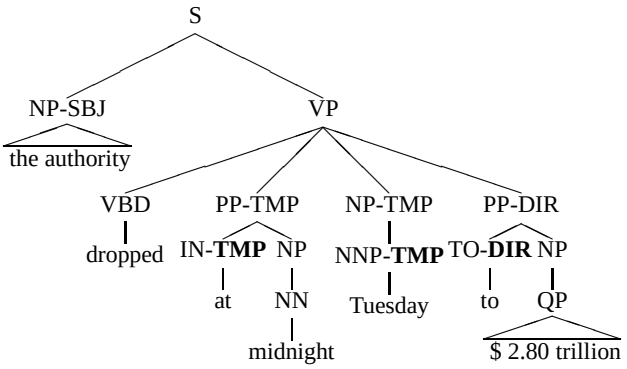


Figure 3: A sample syntactic structure with function labels lowered onto the preterminals.

4 Experiments and Discussion

To assess the relevance of our fine-grained tags and history representations for functional labelling, we compare two augmented models to two baseline models without these augmentations indicated in Table 2 as no-biases and H03. The baseline called H03 refers to our runs of the parser described in (Henderson, 2003), which is not trained on input annotated with function labels. Comparison to this model gives us an external reference to whether function labelling improves parsing. The baseline called no-biases refers to a model without any structural or lexical biases, but trained on input annotated with function labels. This comparison will tell us if the biases are useful or if the reported improvements could have been obtained without explicit manipulation of the parsing biases.

All SSN function parsers were trained on sections 2-21 from the PTB and validated on section 24. They are trained on parse trees whose labels include syntactic and semantic function labels. The models, as well as the parser described in (Henderson, 2003), are run only once. This explains the little difference in performance between our results for H03 in our table of results and those cited in (Henderson, 2003), where the best of three runs on the validation set is chosen. To evaluate the performance of our function parsing experiments, we extend standard Parseval measures of labelled recall and precision to include function labels.

The augmented models have a total of 188 non-terminals to represents labels of constituents, instead of the 33 of the baseline H03 parser. As a result

	FLABEL			FLABEL-less		
	F	R	P	F	R	P
H03				88.6	88.3	88.9
no-biases	84.6	84.4	84.9	88.2	88.0	88.4
split-tags	86.1	85.8	86.5	88.9	88.6	89.3
split-tags+locality	86.4	86.1	86.8	89.2	88.9	89.5

Table 2: Percentage F-measure (F), recall (R), and precision (P) of the SSN baseline and augmented parsers.

of lowering the five function labels, 83 new part-of-speech tags were introduced to partition the original tag set. SSN parsers do not tag their input sentences. To provide the augmented models with tagged input sentences, we trained an SVM tagger whose features and parameters are described in detail in (Gimenez and Marquez, 2004). Trained on section 2-21, the tagger reaches a performance of 95.8% on the test set (section 23) of the PTB using our new tag set.

Both parsing results taking function labels into account in the evaluation (FLABEL) and results not taking them into account in the evaluation (FLABEL-less) are reported in Table 2, which shows results on the test set, section 23 of the PTB. Both the model augmented only with lexical information (through tag splitting) and the one augmented both with finer-grained tags and representations of syntactic locality perform better than our comparison baseline H03, but only the latter is significantly better ($p < .01$, using (Yeh, 2000)'s randomised test). This indicates that while information projected from the lexical items is very important, only a combination of lexical semantics information and careful modelling of syntactic domains provides a significant improvement.

Parsing results outputting function labels (FLABEL columns) reported in Table 2 indicate that parsing function labels is more difficult than parsing bare phrase-structure labels (compare the FLABEL column to the FLABEL-less column). They also show that our model including finer-grained tags and locality biases performs better than the one including only finer-grained tags when outputting function labels. This suggests that our model with both lexical and structural biases performs better than our no-biases comparison baseline precisely because it is able to learn to parse function labels more accurately. Comparisons to the baseline without biases indicates clearly that the observed improvements, both on function parsing and on parsing without taking function labels into consideration would not have been obtained without explicit biases.

Individual performance on syntactic and semantic function labelling compare favourably to previous attempts (Blaheta, 2004; Blaheta and Charniak, 2000). Note that the maximal precision or recall score of function labelling is strictly smaller than one-hundred percent if the precision or the recall of the parser is less than one-hundred percent. Following (Blaheta and Charniak, 2000), incorrectly parsed constituents will be ignored (roughly 11% of the total) in the evaluation of the precision and recall of the function labels, but not in the evaluation of the parser. Of the correctly parsed constituents, some bear function labels, but the overwhelming majority do not bear any label, or rather, in our notation, they bear a NULL label. To avoid calculating excessively optimistic scores, constituents bearing the NULL label are not taken into consideration for computing overall recall and precision figures. NULL-labelled constituents are only needed to calculate the precision and recall of other function labels. For example, consider the confusion matrix M in Table 3 below, which reports scores for the semantic labels recovered by the no-biases model. Precision is computed as $\frac{\sum_{i\in\{\text{ADV}\cdots\text{TMP}\}} M[i,i]}{\sum_{j\in\{\text{ADV}\cdots\text{TMP}\}} M[\text{SUM},j]}$. Recall is computed analogously. Notice that $M[n,n]$, that is the [SEM-NULL,SEM-NULL] cell in the matrix, is never taken into account.

Syntactic labels are recovered with very high accuracy (F 96.5%, R 95.5% and P 97.5%) by the model with both lexical and structural biases, and so are semantic labels, which are considerably more difficult (F 85.6%, R 81.5% and P 90.2%). (Blaheta, 2004) uses specialised models for the two types

		Assigned Labels										
		ADV	BNF	DIR	EXT	LOC	MNR	NOM	PRP	TMP	SEM-NULL	SUM
Actual Labels	ADV	143	0	0	0	0	0	0	1	3	11	158
	BNF	0	0	0	0	0	0	0	0	0	1	1
	DIR	0	0	39	0	3	4	0	0	1	51	98
	EXT	0	0	0	37	0	0	0	0	0	17	54
	LOC	0	0	1	0	345	3	0	0	15	148	512
	MNR	0	0	0	0	3	35	0	0	16	40	94
	NOM	2	0	0	0	0	0	88	0	0	4	94
	PRP	0	0	0	0	0	0	0	54	1	33	88
	TMP	18	0	1	0	24	11	0	1	479	105	639
	SEM-NULL	12	0	13	5	81	28	12	24	97	20292	20564
	SUM	175	0	54	42	456	81	100	80	612	20702	22302

Table 3: Confusion matrix for the no-biases baseline model, tested on the validation set (section 24 of PTB).

of function labels, reaching an F-measure of 98.7% for syntactic labels and 83.4% for semantic labels as best accuracy measure. Previous work that uses, like us, a single model for both types of labels reaches an F measure of 95.7% for syntactic labels and 79.0% for semantic labels (Blaheta and Charniak, 2000).

Although functional information is explicitly annotated in the PTB, it has not yet been exploited by any state-of-the-art statistical parser with the notable exception of the second parsing model of (Collins, 1999). Collins's second model uses a few function labels to discriminate between arguments and adjuncts, and includes parameters to generate subcategorisation frames. Subcategorisation frames are modelled as multisets of arguments that are sisters of a lexicalised head child. Some major differences distinguish Collins's subcategorisation parameters from our structural biases. First, lexicalised head children are not explicitly represented in our model. Second, we do not discriminate between arguments and adjuncts: we only encode the distinctions between syntactic function labels and semantic ones. As shown in (Merlo, 2003; Merlo and Esteve-Ferrer, 2004) this difference does not correspond to the difference between arguments and adjuncts. Finally, our model does not implement any distinction between right and left subcategorisation frames. In Collins's model, the left and right subcategorisation frames are conditionally independent and arguments occupying a complement position (to the right of the head) are independent of arguments occurring in a specifier position (to the left of the head). In our model, no such independence assumptions are stated, because the model is biased towards phrases related to each other by the c-command relation. Such relation could involve both elements at the left and at the right of the head. Relations of functional assignments between subjects and objects, for example, could be captured.

The most important observation, however, is that modelling function labels as the interface between syntax and semantics yields a significant improvement on parsing performance, as can be verified in the FLABEL-less column of Table 2. This is a crucial observation in the light of the current approaches to function or semantic role labelling and its relation to parsing. An improvement in parsing performance by better modelling of function labels indicates that this complex problem is better solved as a single integrated task and that current two-step architectures might be missing on successful ways to improve both the parsing and the labelling task.

In particular, recent models of semantic role labelling separate input indicators of the correlation between the structural position in the tree and the semantic label, such as *path*, from those indicators that encode constraints on the sequence, such as the previously assigned role (Kwon et al., 2004). In this way, they can never encode directly the constraining power of a certain role in a given structural position onto a following node in its structural position. In our augmented model, we attempt to capture these constraints by directly modelling syntactic domains.

Our results confirm the findings in (Palmer et al., 2005). They take a critical look at some commonly used features in the semantic role labelling task, such as the *path* feature. They suggest that the path feature is not very effective because it is sparse. Its

sparseness is due to the occurrence of intermediate nodes that are not relevant for the syntactic relations between an argument and its predicate. Our model of domains is less noisy, because it can focus only on c-commanding nodes bearing function labels, thus abstracting away from those nodes that smear the pertinent relations.

(Yi and Palmer, 2005) share the motivation of our work, although they apply it to a different task. Like the current work, they observe that the distributions of semantic labels could potentially interact with the distributions of syntactic labels and redefine the boundaries of constituents, thus yielding trees that reflect generalisations over both these sources of information.

Our results also confirm the importance of lexical information, the lesson drawn from (Thompson et al., 2004), who find that correctly modelling sequence information is not sufficient. Lexical information is very important, as it reflects the lexical semantics of the constituents. Both factors, syntactic domains and lexical information, are needed to significantly improve parsing.

5 Conclusions

In this paper, we have explored a new way to improve parsing results in a current statistical parser while at the same time enriching its output. We achieve significant improvements in parsing and function labelling by modelling directly the specific nature of function labels, as both expressions of the lexical semantics properties of a constituent and as syntactic elements whose distribution is subject to structural locality constraints. Differently from other approaches, the method we adopt integrates function labelling directly in the parsing process. Future work will lie in exploring new ways of capturing syntactic domains, different from the ones attempted in the current paper, such as developing new derivation moves for nodes bearing function labels. A more detailed analysis of the parser will also shed light on its behaviour on sequences of function labels. Finally, we plan to extend this work to learn Propbank-style semantic role labels, which might require explicit modelling of long distance dependencies and syntactic movement.

Acknowledgements

We thank the Swiss National Science Foundation for supporting this research under grant number 101411-105286/1. We also thank James Henderson for allowing us to use his parser and James Henderson and Mirella Lapata for useful discussion of this work. All remaining errors are our own.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In Christian Boitet and Pete Whitelock, editors, *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics (ACL-COLING'98)*, pages 86–90, Montreal, Canada. Morgan Kaufmann Publishers.

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

Don Blaheta and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Meeting of North American Chapter of Association for Computational Linguistics (NAACL'00)*, pages 234–240, Seattle, Washington.

Don Blaheta. 2004. *Function Tagging*. Ph.D. thesis, Department of Computer Science, Brown University.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of North American Chapter of Association for Computational Linguistics (NAACL'00)*, pages 132–139, Seattle, Washington.

Michael Collins and Scott Miller. 1998. Semantic tagging using a probabilistic context-free grammar. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 38–48, Montreal, CA.

Michael John Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Department of Computer Science, University of Pennsylvania.

CoNLL. 2004. Eighth conference on computational natural language learning (conll-2004). http://cnts.uia.ac.be/conll2004.

CoNLL. 2005. Ninth conference on computational natural language learning (conll-2005). http://cnts.uia.ac.be/conll2005.

Ruifang Ge and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CONLL-05)*, Ann Arbor, Michigan.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 239–246, Philadelphia, PA.

Jesus Gimenez and Lluis Marquez. 2004. Svmtool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal.

James Henderson and Peter Lane. 1998. A connectionist architecture for learning to parse. In *Proceedings of 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, pages 531–537, University of Montreal, Canada.

Jamie Henderson. 2003. Inducing history representations for broad-coverage statistical parsing. In *Proceedings of the Joint Meeting of the North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conference (NAACL-HLT'03)*, pages 103–110, Edmonton, Canada.

Valentin Jijkoun, Maarten de Rijke, and Jori Mur. 2004. Information extraction for question answering: Improving recall through syntactic patterns. In *Proceedings of COLING-2004*, Geneva, Switzerland.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the ACL (ACL'03)*, pages 423–430, Sapporo, Japan.

Namhee Kwon, Michael Fleischman, and Eduard Hovy. 2004. Senseval automatic labeling of semantic roles using maximum entropy models. In *Senseval-3*, pages 129–132, Barcelona, Spain.

Mitch Marcus, Beatrice Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

Paola Merlo and Eva Esteve-Ferrer. 2004. PP attachment and the notion of argument. University of Geneva, manuscript.

Paola Merlo and Gabriele Musillo. 2005. Accurate function parsing. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, Vancouver, Canada.

Paola Merlo. 2003. Generalised PP-attachment disambiguation using corpus-based linguistic diagnostics. In *Proceedings of the Tenth Conference of The European Chapter of the Association for Computational Linguistics (EACL'03)*, pages 251–258, Budapest, Hungary.

Mark Jan Nederhof. 1994. *Linguistic Parsing and Program Transformations*. Ph.D. thesis, Department of Computer Science, University of Nijmegen.

Rodney Nielsen and Sameer Pradhan. 2004. Mixing weak learners in semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-2004)*, pages 80–87, Barcelona, Spain, July.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31:71–105.

V. Punyakanok, D. Roth, and W. Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'05)*.

Senseval. 2004. Third international workshop on the evaluation of systems for the semantic analysis of text (acl 2004). http://www.senseval.org/senseval3.

Cynthia Thompson, Siddharth Patwardhan, and Carolin Arnold. 2004. Generative models for semantic role labeling. In *Senseval-3*, Barcelona, Spain.

Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP-2004)*, pages 88–94, Barcelona, Spain.

Alexander Yeh. 2000. More accurate tests for the statistical signifi cance of the result differences. In *Procs of the 17th International Conf. on Computatioal Linguistics*, pages 947–953, Saarbrucken, Germany.

Szu-ting Yi and Martha Palmer. 2005. The integration of semantic parsing and semantic role labelling. In *Proceedings of CoNLL'05*, Ann Arbor, Michigan.

Probabilistic models for disambiguation of an HPSG-based chart generator

Hiroko Nakanishi†
†Department of Computer Science
University of Tokyo
Hongo 7-3-1, Bunkyo-ku
Tokyo 113-0033, Japan

Yusuke Miyao†
‡CREST, JST
Honcho 4-1-8, Kawaguchi-shi
Saitama 332-0012, Japan

Jun'ichi Tsujii†‡
‡School of Informatics
University of Manchester
POBox 88, Sackville St
MANCHESTER M60 1QD, UK

n165, yusuke, tsujii@is.s.u-tokyo.ac.jp

Abstract

We describe probabilistic models for a chart generator based on HPSG. Within the research field of parsing with lexicalized grammars such as HPSG, recent developments have achieved efficient estimation of probabilistic models and high-speed parsing guided by probabilistic models. The focus of this paper is to show that two essential techniques – model estimation on packed parse forests and beam search during parsing – are successfully exported to the task of natural language generation. Additionally, we report empirical evaluation of the performance of several disambiguation models and how the performance changes according to the feature set used in the models and the size of training data.

1 Introduction

Surface realization is the final stage of natural language generation which receives a semantic representation and outputs a corresponding sentence where all words are properly inflected and ordered. This paper presents log-linear models to address the ambiguity which arises when HPSG (Head-driven Phrase Structure Grammar (Pollard and Sag, 1994)) is applied to sentence generation. Usually a single semantic representation can be realized as several sentences. For example, consider the following two sentences generated from the same input.

- The *complicated* language in the huge new law has muddied the fight.
- The language in the huge new law *complicated* has muddied the fight.

The latter is not an appropriate realization because *"complicated"* tends to be wrongly interpreted to modify *"law"*. Therefore the generator needs to select a candidate sentence which is more fluent and easier to understand than others.

In principle, we need to enumerate all alternative realizations in order to estimate a log-linear model for generation. It therefore requires high computational cost to estimate a probabilistic model for a wide-coverage grammar because there are considerable ambiguities and the alternative realizations are hard to enumerate explicitly. Moreover, even after the model has been estimated, to explore all possible candidates in runtime is also expensive. The same problems also arise with HPSG parsing, and recent studies (Tsuruoka et al., 2004; Miyao and Tsujii, 2005; Ninomiya et al., 2005) proposed a number of solutions including the methods of estimating log-linear models using packed forests of parse trees and pruning improbable candidates during parsing.

The aim of this paper is to apply these techniques to generation. Since parsing and generation both output the best probable tree under some constraints, we expect that techniques that work effectively in parsing are also beneficial for generation. First, we enabled estimation of log-linear models with less cost by representing a set of generation trees in a packed forest. The forest representation was obtained by adopting chart generation (Kay, 1996; Car-

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 93–102,
Vancouver, October 2005.

roll et al., 1999) where ambiguous candidates are packed into an *equivalence class* and mapping a chart into a forest in the same way as parsing. Second, we reduced the search space in runtime by adopting *iterative beam search* (Tsuruoka and Tsujii, 2004) that efficiently pruned improbable candidates. We evaluated the generator on the Penn Treebank (Marcus et al., 1993), which is highly reliable corpus consisting of real-world texts.

Through a series of experiments, we compared the performance of several disambiguation models following an existing study (Velldal and Oepen, 2005) and examined how the performance changed according to the size of training data, the feature set, and the beam width. Comparing the latter half of the experimental results with those on parsing (Miyao and Tsujii, 2005), we investigated similarities and differences between probabilistic models for parsing and generation. The results indicated that the techniques exported from parsing to generation worked well while the effects were slightly different in detail.

The Nitrogen system (Langkilde and Knight, 1998; Langkilde, 2000) maps semantic relations to a packed forest containing all realizations and selects the best one with a bigram model. Our method extends their approach in that we can utilize syntactic features in the disambiguation model in addition to the bigram.

From the perspective of using a lexicalized grammar developed for parsing and importing parsing techniques, our method is similar to the following approaches. The Fergus system (Bangalore and Rambow, 2000) uses LTAG (Lexicalized Tree Adjoining Grammar (Schabes et al., 1988)) for generating a word lattice containing realizations and selects the best one using a trigram model. White and Baldridge (2003) developed a chart generator for CCG (Combinatory Categorial Grammar (Steedman, 2000)) and proposed several techniques for efficient generation such as best-first search, beam thresholding and chunking the input logical forms (White, 2004). Although some of the techniques look effective, the models to rank candidates are still limited to simple language models. Carroll et al. (1999) developed a chart generator using HPSG. After the generator outputs all the sentences the grammar allows, the ranking module (Velldal and Oepen, 2005) selects the best one using a log-linear model. Their model is trained using only 864 sentences where all realizations can be explicitly enumerated.

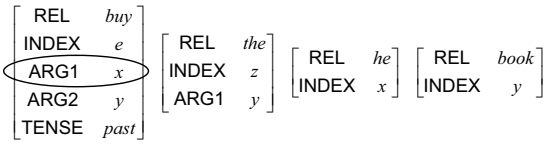


Figure 1: PASs for "*He bought the book.*"

As a grammar is extended to support more linguistic phenomena and to achieve higher coverage, the number of alternative realizations increases and the enumeration requires much higher computational cost. Moreover, using a variety of syntactic features also increases the cost. By representing a set of realizations compactly with a packed forest, we trained the models with rich features on a large corpus using a wide-coverage grammar.

2 Background

This section describes background of this work including the representation of the input to our generator, the algorithm of chart generation, and probabilistic models for HPSG.

2.1 Predicate-argument structures

The grammar we adopted is the Enju grammar, which is an English HPSG grammar extracted from the Penn Treebank by Miyao et al. (2004). In parsing a sentence with the Enju grammar, semantic relations of words is output included in a parse tree. The semantic relations are represented by a set of *predicate-argument structures* (PASs), which in turn becomes the input to our generator. Figure 1 shows an example input to our generator which corresponds to the sentence "*He bought the book.*", which consists of four predicates. REL expresses the base form of the word corresponding to the predicate. INDEX expresses a *semantic variable* to identify each word in the set of relations. ARG1 and ARG2 express relationships between the predicate and its arguments, e.g., the circled part in Figure 1 shows "*he*" is the subject of "*buy*" in this example. The other constraints in the parse tree are omitted in the input for the generator. Since PASs abstract

away superficial differences, generation from a set of PASs contains ambiguities in the order of modifiers like the example in Section 1 or the syntactic categories of phrases. For example, the PASs in Figure 1 can generate the NP, *"the book he bought."*

When processing the input PASs, we split a single PAS into a set of relations like (1) representing the first PAS in Figure 1.

$$\mathrm{buy}(e) \wedge \mathrm{past}(e) \wedge \mathrm{ARG1}(e, x) \wedge \mathrm{ARG2}(e, y) \quad (1)$$

This representation is very similar to the notion of HLDS (Hybrid Logic Dependency Semantics) employed by White and Baldridge (2003), which is a related notion to MRS (Minimal Recursion Semantics) employed by Carroll et al. (1999). The most significant difference between our current input representation (not PAS itself) and the other representations is that each word corresponds to exactly one PAS while words like infinitival *"to"* have no semantic relations in HLDS. This means that *"The book was bought by him."* is not generated from the same PASs as Figure 1 because there must be the PASs for *"was"* and *"by"* to generate the sentence.

We currently adopt this constraint for simple implementation, but it is possible to use the input where PASs for words like *"to"* are removed. As proposed and implemented in the previous studies (Carroll et al., 1999; White and Baldridge, 2003), handling such inputs is feasible by modification in chart generation described in the following section. The algorithms proposed in this paper can be integrated with their algorithms although the implementation is left for future research.

2.2 Chart generation

Chart generation is similar to chart parsing, but what an edge covers is the semantic relations associated with it. We developed a CKY-like generator which deals with binarized grammars including the Enju. Figure 2 shows a chart for generating *"He bought the book."* First, lexical edges are assigned to each PAS. Then the following loop is repeated from $n = 2$ to the cardinality of the input.

- Apply binary rules to existing edges to generate new edges holding n PASs.
- Apply unary rules to the new edges generated in the previous process.
- Store the edges generated in the current loop into the chart[1].

In Figure 2, boxes in the chart represent $cells$, which contain edges covering the same PASs, and solid arrows represent rule applications. Each edge is packed into an equivalence class and stored in a cell. Equivalence classes are identified with their signs and the semantic relations they cover. Edges with different strings (e.g., NPs associated with *"a big white dog"* and *"a white big dog"*) can be packed into the same equivalence class if they have the same feature structure.

In parsing, each edge must be combined with its adjacent edges. Since there is no such constraint in generation, the combinations of edges easily explodes. We followed two partial solutions to this problem by Kay (1996).

The one is indexing edges with the semantic variables (e.g., circled y in Figure 2). For example, since the SUBCAT feature of the edge for *"bought the book"* specifies that it requires an NP with an index x, we can find the required edges efficiently by checking the edges indexed with x.

The other is prohibiting proliferation of grammatically correct, but unusable sub-phrases. During generating the sentence "Newspaper reports said that the tall young Polish athlete ran fast", subphrases with incomplete modifiers such as "the tall young athlete" or "the young Polish athlete" do not construct the final output, but slow down the generation because they can be combined with the rest of the input to construct grammatically correct phrases or sentences. Carroll et al. (1999) and White (2004) proposed different algorithms to address the same problem. We adopted Kay's simple solution in the current ongoing work, but logical form chunking proposed by White is also applicable to our system.

2.3 Probabilistic models for generation with HPSG

Some existing studies on probabilistic models for HPSG parsing (Malouf and van Noord, 2004; Miyao and Tsujii, 2005) adopted log-linear models (Berger et al., 1996). Since log-linear models allow us to

[1] To introduce an edge with no semantic relations as mentioned in the previous section, we need to combine the edges with edges having no relations.

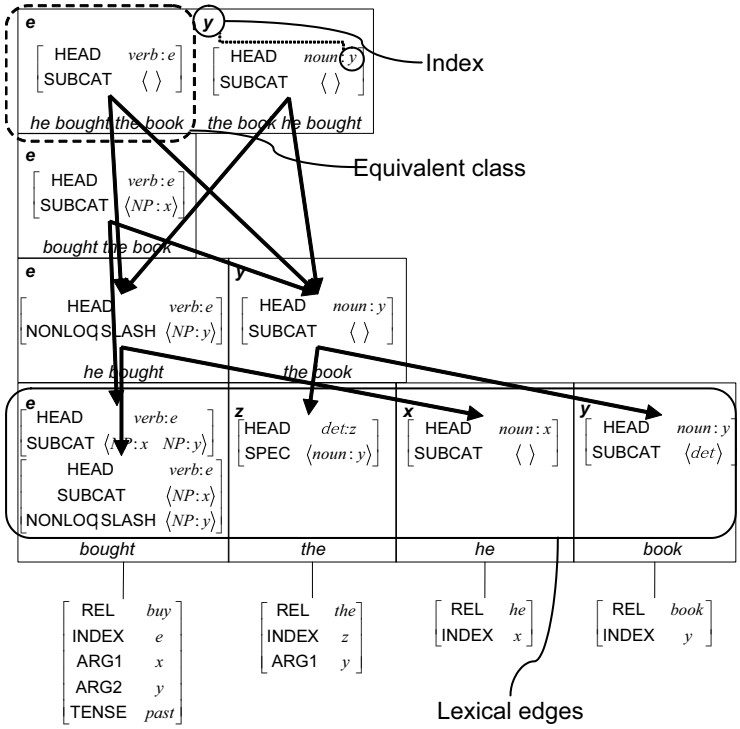


Figure 2: The chart for "*He bought the book.*"

use multiple overlapping features without assuming independence among features, the models are suitable for HPSG parsing where feature structures with complicated constraints are involved and dividing such constraints into independent features is difficult. Log-linear models have also been used for HPSG generation by Velldal and Oepen (2005). In their method, the probability of a realization r given a semantic representation s is formulated as

$$p_\lambda(r|s) = \frac{\exp(\sum_i \lambda_i f_i(r))}{\sum_{r' \in Y(s)} \exp(\sum_i \lambda_i f_i(r'))},$$

where $f_i(r)$ is a feature function observed in r, λ_i is the weight of f_i, and $Y(s)$ represents the set of all possible realizations of s. To estimate λ_i, pairs of $(r, Y(s))$ are needed, where r is the most preferred realization for s. Their method first automatically generates a paraphrase treebank, where $\langle r, s, Y(s)\rangle$ are enumerated. Then, a log-linear model is trained with this treebank, i.e., each λ_i is estimated so as to maximize the likelihood of training data. As well as features used in their previous work on statistical parsing (Toutanova and Manning, 2002), an additional feature that represents sentence probabilities of 4-gram model is incorporated. They showed that the combined model outperforms the model without the 4-gram feature.

3 Disambiguation models for chart generation

3.1 Packed representation of a chart

As mentioned in Section 2.3, to estimate log-linear models for HPSG generation, we need all alternative derivation trees $T(s)$ generated from the input s. However, the size of $T(s)$ is exponential to the cardinality of s and they cannot be enumerated explicitly. This problem is especially serious in wide-coverage grammars because such grammars are designed to cover a wide variety of linguistic phenomena, and thus produce many realizations. In this section, we present a method of making the estimation tractable which is similar to a technique developed for HPSG parsing.

When estimating log-linear models, we map $T(s)$ in the chart into a packed representation called a *feature forest*, intuitively an "AND-OR" graph. Miyao and Tsujii (2005) represented a set of HPSG parse

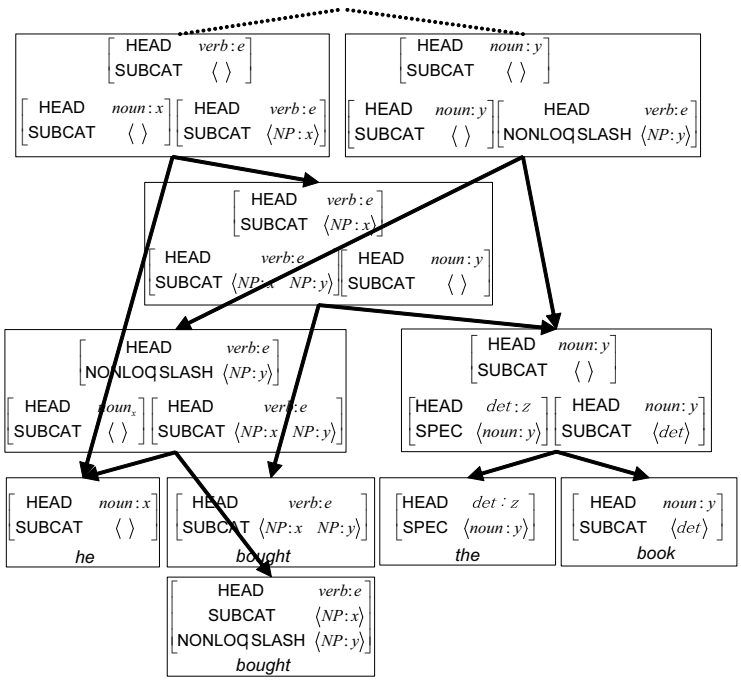


Figure 3: Feature forest for "*He bought the book.*"

trees using a feature forest and succeeded in estimating $p(t|w)$ given a sentence w and a parse tree t using dynamic programming without unpacking the chart. If $T(s)$ is represented in a feature forest in generation, $p(t|s)$ can also be estimated in the same way.

Figure 3 shows a feature forest representing the chart in Figure 2. Each node corresponds to either a lexical entry or a tuple of $\langle e_m, e_l, e_r \rangle$ where e_m, e_l and e_r are respectively the mother edge, the left daughter, and the right daughter in a single rule application. Nodes connected by dotted lines represent OR-nodes, i.e., equivalence classes in the same cell. Feature functions are assigned to OR-nodes. By doing so, we can capture important features for disambiguation in HPSG, i.e., combinations of a mother and its daughter(s). Nodes connected by solid arrows represent AND-nodes corresponding to the daughters of the parent node. By using feature forests, we can efficiently pack the node generated more than once in the set of trees. For example, the nodes corresponding to "*the book*" in "*He bought the book.*" and "*the book he bought*" are identical and described only once in the forest. The merits of using forest representations in generation instead of lattices or simple enumeration are discussed thoroughly by Langkilde (2000).

3.2 Model variation

We implemented and compared four different disambiguation models as Velldal and Oepen (2005) did. Throughout the models, we assigned a score called *figure-of-merit* (FOM) on each edge and calculated the FOM of a mother edge by dynamic programming. FOM represents the log probability of an edge which is not normalized.

Baseline model We started with a simple baseline model, $p(t|s) = \prod_{r \in s} p(l|r)$, where $r \in s$ is a PAS in the input semantic representation s and l is a lexical entry assigned to r. The FOM of the mother edge $\Delta(e_m)$ is computed simply as $\Delta(e_m) = \Delta(e_l) + \Delta(e_r)$. All the other models use this model as a reference distribution (Miyao and Tsujii, 2005), i.e., λ_i is estimated to maximize the likelihood of the training data $\tilde{p}$, which is calculated with the following equation.

$$\tilde{p}_\lambda(t|s) = \prod_{r \in s} p(l|r) \cdot \frac{\exp(\sum_i \lambda_i f_i(t))}{\sum_{t' \in T(s)} \exp(\sum_i \lambda_i f_i(t'))}$$

Bigram model The second model is a log-linear model with only one feature that corresponds to bigram probabilities for adjacent word-pairs in the sentence. We estimated a bigram language model using a part of the British National Corpus as training data[2]. In the chart each edge is identified with the first and last word in the phrase as well as its feature structure and covered relations. When two edges are combined, $\Delta(e_m)$ is computed as $\Delta(e_m) = \Delta(e_l) + \Delta(e_r) - \lambda p_{bigram}(w_r|w_l)$, where λ is the weight of the bigram feature, w_l is the last word of the left daughter, w_r is the first word of the right daughter, and p_{bigram} represents a log probability of a bigram. Contrary to the method of Velldal and Oepen (2005) where the input is a set of sentences and p_{n-gram} is computed on a whole sentence, we computed p_{bigram} on each phrase as Langkilde (2000) did The language model can be extended to n-gram if each edge holds last $n-1$ words although the number of edges increase.

Syntax model The third model incorporates a variety of syntactic features and lexical features where $\Delta(e_m)$ is computed as $\Delta(e_l) + \Delta(e_r) + \sum_i \lambda_i f_i(e_m, e_l, e_r)$. The feature set consists of combinations of atomic features shown in Table 1. The atomic features and their combinations are imported from the previous work on HPSG parsing (Miyao and Tsujii, 2005). We defined three types of feature combinations to capture the characteristics of binary and unary rule applications and root edges as described below.

$$f_{binary} = \left\langle \begin{array}{l} \mathrm{RULE}, \mathrm{DIST}, \mathrm{COMMA}, \\ \mathrm{SPAN_l}, \mathrm{SYM_l}, \mathrm{WORD_l}, \mathrm{LE_l}, \\ \mathrm{SPAN_r}, \mathrm{SYM_r}, \mathrm{WORD_r}, \mathrm{LE_r} \end{array} \right\rangle$$

$$f_{unary} = \langle \mathrm{RULE}, \mathrm{SYM}, \mathrm{WORD}, \mathrm{LE} \rangle$$

$$f_{root} = \langle \mathrm{SYM}, \mathrm{WORD}, \mathrm{LE} \rangle$$

An example of extracted features is shown in Figure 4 where *"bought the book"* is combined with its subject *"he"*. Since the mother edge is a root edge, two features (f_{root} and f_{binary}) are extracted from this node. In the f_{root} feature, the phrasal category SYM becomes S (sentence), the head word

[2]The model estimation was done using the CMU-Cambridge Statistical Language Modeling toolkit (Clarkson and Rosenfeld, 1997).

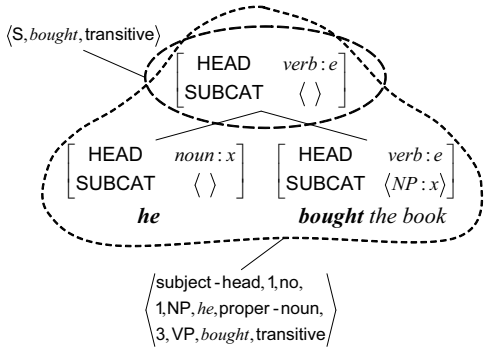


Figure 4: Example of features

Table 1: Atomic features

RULE	the name of the applied schema
DIST	the distance between the head words of the daughters
COMMA	whether a comma exists between daughters and/or inside of daughter phrases
SPAN	the number of words dominated by the phrase
SYM	the symbol of the phrasal category (e.g., NP, VP)
WORD	the surface form of the head word
LE	the lexical entry assigned to the head word

WORD becomes *"bought"*, and its lexical entry LE becomes that of transitive verbs. In the f_{binary} feature, properties of the left and right daughters are instantiated in addition to those of the mother edge.

Combined model The fourth and final model is the combination of the syntax model and the bigram model. This model is obtained by simply adding the bigram feature to the syntax model.

4 Iterative beam search

For efficient statistical generation with a wide-coverage grammar, we reduce the search space by pruning edges during generation. We use beam search where edges with low FOMs are pruned during generation. We use two parameters, n and d: in each cell, the generator prunes except for top n edges, and edges whose FOMs are lower than that of the top edge d are also pruned.

Another technique for achieving efficiency is *iterative generation* which is adopted from iterative CKY parsing (Tsuruoka and Tsujii, 2004). When beam width is too narrow, correct edges to constitute a correct sentence may be discarded during gen-

Table 2: Averaged generation time and accuracy by four models

Model		Baseline	Bigram	Syntax	Combined
Coverage (%)		91.15	90.15	90.75	90.56
Time (ms)		3512	4085	3821	4315
BLEU	length < 5 (89 sentences)	0.7776	0.7503	0.8195	0.7359
	$5 \leq$ length < 10 (179 sentences)	0.5544	0.6323	0.7339	0.7305
	$10 \leq$ length < 15 (326 sentences)	0.5809	0.6415	0.7735	0.7384
	$15 \leq$ length < 20 (412 sentences)	0.5863	0.6542	0.7835	0.7533
	Total (1,006 sentences)	0.5959	0.6544	0.7733	0.7420

eration and it causes degradation in coverage, i.e., the ratio the generator successfully outputs a sentence. The appropriate beam width depends on inputs and cannot be predefined. In iterative generation, the process of chart generation is repeated with increasing beam width until a complete sentence is generated or the beam width exceeds the predefined maximum.

5 Experiments

In this section, we present five experiments: comparison among four models described in Section 3.2, syntax models with different features, different corpus sizes, different beam widths, and the distribution of generation time. The bigram model was trained using 100,000 sentences in the BNC. The unigram and syntax model was trained using Section 02-21 of the WSJ portion of the Penn Treebank (39,832 sentences). Section 22 (1,700 sentences) and 23 (2,416 sentences) were used as the development and test data, respectively.

Because the generator is still slow to generate long sentences, sentences with more than 20 words were not used. We converted the treebank into HPSG-style derivation trees by the method of Miyao et al. (2004) and extracted the semantic relations, which are used as the inputs to the generator. The sentences where this conversion failed were also eliminated although such sentences were few (about 0.3% of the eliminated data). The resulting training data consisted of 18,052 sentences and the test data consisted of 1,006 sentences. During training, *uncovered* sentences – where the lexicon does not include the lexical entry to construct correct derivation – were also ignored, while such sentences remained in the test data. The final training data we can utilize consisted of 15,444 sentences. The average sentence length of the test data was 12.4, which happens to be close to that of Velldal and Oepen (2005) though the test data is different.

The accuracy of the generator outputs was evaluated by the BLEU score (Papineni et al., 2001), which is commonly used for the evaluation of machine translation and recently used for the evaluation of generation (Langkilde-Geary, 2002; Velldal and Oepen, 2005). BLEU is the weighted average of n-gram precision against the reference sentence. We used the sentences in the Penn Treebank as the reference sentences. The beam width was increased from $(n, d) = (8, 4.0)$ to $(20, 10.0)$ in two steps. The parameters were empirically determined using the development set. All the experiments were conducted on AMD Opteron servers with a 2.0-GHz CPU and 12-GB memory.

Table 2 shows the average generation time and the accuracy of the models presented in Section 3. The generation time includes time for the input for which the generator could not output a sentence, while the accuracy was calculated only in the case of successful generation. All models succeeded in generation for over 90% of the test data.

Contrary to the result of the Velldal and Oepen (2005), the syntax model outperformed the combined model. We observed the same result when we varied the parameters for beam thresholding. This is possibly just because the language model was not trained enough as that of the previous research (Velldal and Oepen, 2005) where the model was 4-gram and trained with the entire BNC[3].

[3]We could not use the entire corpus for training because of a problem in implementation. This problem will be fixed in the

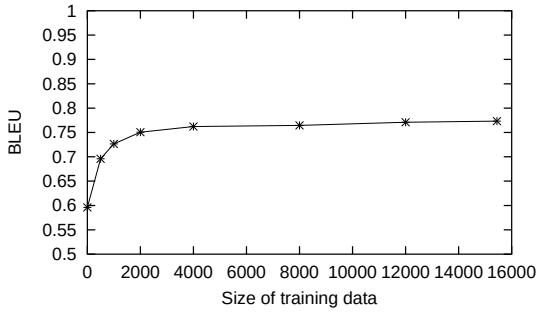


Figure 5: Size of training data vs. performance

Table 3: Feature set vs. performance

Feature	BLEU	diff.
All	0.7734	
-COMMA	**0.7492**	**-0.0242**
-DIST	0.7702	-0.0032
-LE	**0.7423**	**-0.0311**
-RULE	0.7709	-0.0025
-SPAN	0.7640	-0.0094
-SYM	**0.7400**	**-0.0334**
-WORD	0.7610	-0.0124
None	0.5959	-0.1775

Although the accuracy shown in Table 2 was lower than that of Velldal and Oepen, there is little point in direct comparison between the accuracy of the two systems because the settings are considerably different in terms of the grammar, the input representation, and the training and test set. The algorithm we proposed does not depend on our specific setting and can be integrated and evaluated within their setting. We used larger training data (15,444 sentences) and test data (1,006 sentences), compared to their treebank of 864 sentences where the log-linear models were evaluated by cross validation. This is the advantage of adopting feature forests to efficiently estimate the log-linear models.

Figure 5 shows the relationship between the size of training data and the accuracy. All the following experiments were conducted on the syntax model. The accuracy seems to saturate around 4000 sentences, which indicates that a small training set is enough to train the current syntax model and that

future development.

Table 4: n vs. performance

n	Coverage (%)	Time (ms)	BLEU
4	66.10	768	0.7685
8	82.91	3359	0.7654
12	87.89	7191	0.7735
16	89.46	11051	0.7738
20	90.56	15530	0.7723

Table 5: d vs. performance

d	Coverage (%)	Time (ms)	BLEU
4.0	78.23	2450	0.7765
6.0	89.56	9083	0.7693
8.0	91.15	19320	0.7697
10.0	89.86	35897	0.7689

we could use an additional feature set to improve the accuracy. Similar results are reported in parsing (Miyao and Tsujii, 2005) while the accuracy saturated around 16,000 sentences. When we use more complicated features or train the model with longer sentences, possibly the size of necessary training data will increase.

Table 3 shows the performance of syntax models with different feature sets. Each row represents a model where one of the atomic features in Table 1 was removed. The "None" row is the baseline model. The rightmost column represents the difference of the accuracy from the model trained with all features. SYM, LE, and COMMA features had a significant influence on the performance. These results are different from those in parsing reported by Miyao and Tsujii (2005) where COMMA and SPAN especially contributed to the accuracy. This observation implies that there is still room for improvement by tuning the combination of features for generation.

We compared the performance of the generator with different beam widths to investigate the effect of iterative beam search. Table 4 shows the results when we varied n, which is the number of edges, while thresholding by FOM differences is disabled, and Table 5 shows the results when we varied only d, which is the FOM difference.

Intuitively, beam search may decrease the accuracy because it cannot explore all possible candi-

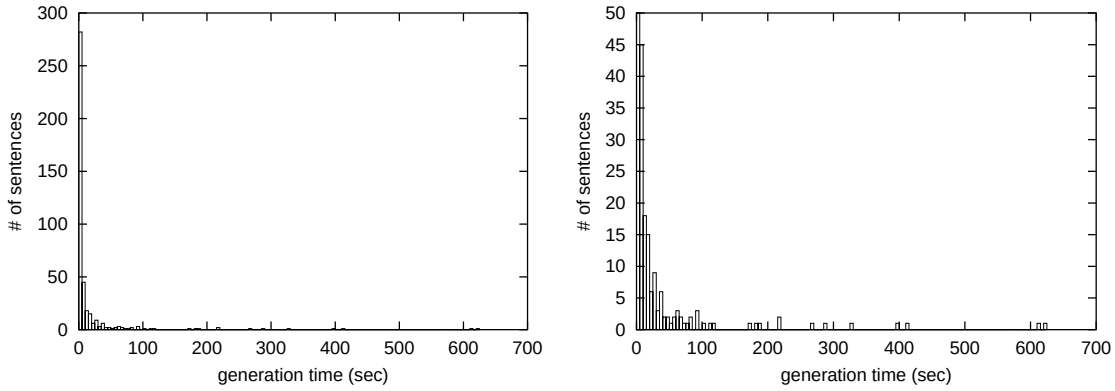


Figure 6: Distribution of generation time

dates during generation. Iterative beam search is more likely to decrease the accuracy than ordinary beam search. However, the results show that the accuracy did not drastically decrease at small widths. Moreover, the accuracy of iterative beam search was almost the same as that of $n = 20$. On the other hand, generation time significantly increased as n or d increased, indicating that iterative beam search efficiently discarded unnecessary edges without loosing the accuracy. Although the coverage increases as the beam width increases, the coverage at $n = 20$ or $d = 10.0$ is lower than that of iterative beam search (Table 2)[4].

Finally, we examined the distribution of generation time without the limitation of sentence length in order to investigate the strategy to improve the efficiency of the generator. Figure 6 is a histogram of generation time for 500 sentences randomly selected from the development set, where 418 sentences were successfully generated and the average BLEU score was 0.705. The average sentence length was 22.1 and the maximum length was 60, and the average generation time was 27.9 sec, which was much longer than that for short sentences. It shows that a few sentences require extremely long time for generation although about 70% of the sentences were generated within 5 sec. Hence, the average time possibly decreases if we investigate what kind of sentences require especially long time and improve the algorithm to remove such time-consuming fractions. The investigation is left for future research.

The closest empirical evaluations on the same task is that of Langkilde-Geary (2002) which reported the performance of the HALogen system while the approach is rather different. Hand-written mapping rules are used to make a forest containing all candidates and the best candidate is selected using the bigram model. The performance of the generator was evaluated on Section 23 of the Penn Treebank in terms of the number of ambiguities, generation time, coverage, and accuracy. Several types of input specifications were examined in order to measure how specific the input should be for generating valid sentences. One of the specifications named "permute, no dir" is similar to our input in that the order of modifiers is not determined at all. The generator produced outputs for 82.7% of the inputs with average generation time 30.0 sec and BLEU score 0.757. The results of our last experiment are comparable to these results though the used section is different.

6 Conclusion

We presented a chart generator using HPSG and developed log-linear models which we believe was essential to develop a sentence generator. Several techniques developed for parsing also worked in generation. The introduced techniques were an estimation method for log-linear models using a packed forest representation of HPSG trees and iterative beam search. The system was evaluated through application to real-world texts. The experimental results

[4]This is because the generator fails when the number of edges exceeds 10,000. Since the number of edges signifi cantly increases when n or d is large, generation fails even if the correct edges are in the chart.

showed that the generator was able to output a sentence for over 90% of the test data when the data was limited to short sentences. The accuracy was significantly improved by incorporating syntactic features into the log-linear model. As future work we intend to tune the feature set for generation. We also plan to further increase the efficiency of the generator so as to generate longer sentences.

References

S. Bangalore and O. Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the COLING'00*.

A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

J. Carroll, A. Copestake, D. Flickinger, and V. Poznanski. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the EWNLG'99*.

P. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of ESCA Eurospeech*.

M. Kay. 1996. Chart generation. In *Proceedings of the ACL'96*, pages 200–204.

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the COLING-ACL'98*, pages 704–710.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the INLG'02*.

I. Langkilde. 2000. Forest-based statistical sentence generation. In *Proceedings of the NAACL'00*.

R. Malouf and G. van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop on Beyond Shallow Analyses*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*.

Y. Miyao and J. Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the ACL'05*.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-Driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the IJCNLP-04*.

T. Ninomiya, Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of the IWPT'05*.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the ACL'01*.

C. Pollard and I. A. Sag. 1994. *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Y. Schabes, A. Abeille, and A. K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammar. In *Proceedings of the COLING 1988*, pages 578–583.

M. Steedman. 2000. *The Syntactic Process*. MIT Press.

K. Toutanova and C. D. Manning. 2002. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of the CoNLL'02*.

Y. Tsuruoka and J. Tsujii. 2004. Iterative CKY parsing for Probabilistic Context-Free Grammars. In *Proceedings of the IJCNLP'04*.

Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2004. Towards efficient probabilistic HPSG parsing: integrating semantic and syntactic preference to guide the parsing. In *Proceedings of the IJCNLP-04 Workshop on Beyond Shallow Analyses*.

E. Velldal and S. Oepen. 2005. Maximum entropy models for realization ranking. In *Proceedings of the MT-Summit'05*.

M. White and J. Baldridge. 2003. Adapting chart realization to CCG. In *Proceedings of the EWNNLG'03*.

M. White. 2004. Reining in CCG chart realization. In *Proceedings of the INLG'04*.

Efficacy of Beam Thresholding, Unification Filtering and Hybrid Parsing in Probabilistic HPSG Parsing

Takashi Ninomiya
CREST, JST
and
Department of Computer Science
The University of Tokyo
ninomi@is.s.u-tokyo.ac.jp

Yoshimasa Tsuruoka
CREST, JST
and
Department of Computer Science
The University of Tokyo
tsuruoka@is.s.u-tokyo.ac.jp

Yusuke Miyao
Department of Computer Science
The University of Tokyo
yusuke@is.s.u-tokyo.ac.jp

Jun'ichi Tsujii
Department of Computer Science
The University of Tokyo
and
School of Informatics
University of Manchester
and
CREST, JST
tsujii@is.s.u-tokyo.ac.jp

Abstract

We investigated the performance efficacy of beam search parsing and deep parsing techniques in probabilistic HPSG parsing using the Penn treebank. We first tested the beam thresholding and iterative parsing developed for PCFG parsing with an HPSG. Next, we tested three techniques originally developed for deep parsing: quick check, large constituent inhibition, and hybrid parsing with a CFG chunk parser. The contributions of the large constituent inhibition and global thresholding were not significant, while the quick check and chunk parser greatly contributed to total parsing performance. The precision, recall and average parsing time for the Penn treebank (Section 23) were 87.85%, 86.85%, and 360 ms, respectively.

1 Introduction

We investigated the performance efficacy of beam search parsing and deep parsing techniques in probabilistic head-driven phrase structure grammar (HPSG) parsing for the Penn treebank. We first applied beam thresholding techniques developed for CFG parsing to HPSG parsing, including local thresholding, global thresholding (Goodman, 1997), and iterative parsing (Tsuruoka and Tsujii, 2005b). Next, we applied parsing techniques developed for deep parsing, including quick check (Malouf et al., 2000), large constituent inhibition (Kaplan et al., 2004) and hybrid parsing with a CFG chunk parser (Daum et al., 2003; Frank et al., 2003; Frank, 2004). The experiments showed how each technique contributes to the final output of parsing in terms of precision, recall, and speed for the Penn treebank.

Unification-based grammars have been extensively studied in terms of linguistic formulation and computation efficiency. Although they provide precise linguistic structures of sentences, their processing is considered expensive because of the detailed descriptions. Since efficiency is of particular concern in practical applications, a number of studies have focused on improving the parsing efficiency of unification-based grammars (Oepen et al., 2002). Although significant improvements in efficiency have been made, parsing speed is still not high enough for practical applications.

The recent introduction of probabilistic models of wide-coverage unification-based grammars (Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005) has opened up the novel possibility of increasing parsing speed by guiding the search path using probabilities. That is, since we often require only the most probable parse result, we can compute partial parse results that are likely to contribute to the final parse result. This approach has been extensively studied in the field of probabilistic

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 103–114,
Vancouver, October 2005.

CFG (PCFG) parsing, such as Viterbi parsing and beam thresholding.

While many methods of probabilistic parsing for unification-based grammars have been developed, their strategy is to first perform exhaustive parsing without using probabilities and then select the highest probability parse. The behavior of their algorithms is like that of the Viterbi algorithm for PCFG parsing, so the correct parse with the highest probability is guaranteed. The interesting point of this approach is that, once the exhaustive parsing is completed, the probabilities of non-local dependencies, which cannot be computed during parsing, are computed after making a packed parse forest. Probabilistic models where probabilities are assigned to the CFG backbone of the unification-based grammar have been developed (Kasper et al., 1996; Briscoe and Carroll, 1993; Kiefer et al., 2002), and the most probable parse is found by PCFG parsing. This model is based on PCFG and not probabilistic unification-based grammar parsing. Geman and Johnson (Geman and Johnson, 2002) proposed a dynamic programming algorithm for finding the most probable parse in a packed parse forest generated by unification-based grammars without expanding the forest. However, the efficiency of this algorithm is inherently limited by the inefficiency of exhaustive parsing.

In this paper we describe the performance of beam thresholding, including iterative parsing, in probabilistic HPSG parsing for a large-scale corpora, the Penn treebank. We show how techniques developed for efficient deep parsing can improve the efficiency of probabilistic parsing. These techniques were evaluated in experiments on the Penn Treebank (Marcus et al., 1994) with the wide-coverage HPSG parser developed by Miyao et al. (Miyao et al., 2005; Miyao and Tsujii, 2005).

2 HPSG and probabilistic models

HPSG (Pollard and Sag, 1994) is a syntactic theory based on lexicalized grammar formalism. In HPSG, a small number of schemata describe general construction rules, and a large number of lexical entries express word-specific characteristics. The structures of sentences are explained using combinations of schemata and lexical entries. Both schemata and lexical entries are represented by typed feature structures, and constraints represented by feature structures are checked with *unification*.

Figure 1 shows an example of HPSG parsing of the sentence "*Spring has come.*" First, each of the lexical entries for "*has*" and "*come*" is unified with a daughter feature structure of the Head-Complement

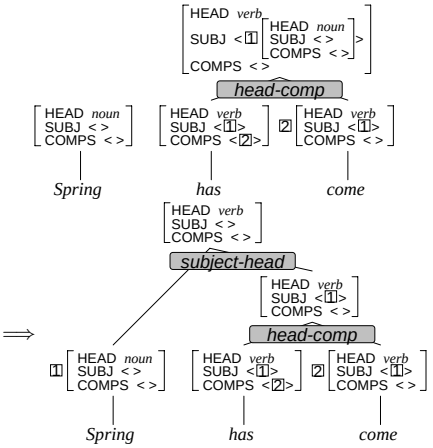


Figure 1: HPSG parsing

Schema. Unification provides the phrasal sign of the mother. The sign of the larger constituent is obtained by repeatedly applying schemata to lexical/phrasal signs. Finally, the parse result is output as a phrasal sign that dominates the sentence.

Given set $\mathcal{W}$ of words and set $\mathcal{F}$ of feature structures, an HPSG is formulated as a tuple, $G = \langle L, R \rangle$, where

$L = \{l = \langle w, F \rangle | w \in \mathcal{W}, F \in \mathcal{F}\}$ is a set of lexical entries, and
R is a set of schemata, i.e., $r \in R$ is a partial function: $\mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$.

Given a sentence, an HPSG computes a set of phrasal signs, i.e., feature structures, as a result of parsing.

Previous studies (Abney, 1997; Johnson et al., 1999; Riezler et al., 2000; Miyao et al., 2003; Malouf and van Noord, 2004; Kaplan et al., 2004; Miyao and Tsujii, 2005) defined a probabilistic model of unification-based grammars as a *log-linear model* or *maximum entropy model* (Berger et al., 1996). The probability of parse result T assigned to given sentence $\mathbf{w} = \langle w_1, \ldots, w_n \rangle$ is

$$p(T|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_i \lambda_i f_i(T)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} \exp\left(\sum_i \lambda_i f_i(T')\right),$$

where λ_i is a model parameter, and f_i is a feature function that represents a characteristic of parse tree T. Intuitively, the probability is defined as the normalized product of the weights $\exp(\lambda_i)$ when a characteristic corresponding to f_i appears in parse result T. Model parameters λ_i are estimated using numer-

ical optimization methods (Malouf, 2002) so as to maximize the log-likelihood of the training data.

However, the above model cannot be easily estimated because the estimation requires the computation of $p(T|\mathbf{w})$ for all parse candidates assigned to sentence $\mathbf{w}$. Because the number of parse candidates is exponentially related to the length of the sentence, the estimation is intractable for long sentences.

To make the model estimation tractable, Geman and Johnson (Geman and Johnson, 2002) and Miyao and Tsujii (Miyao and Tsujii, 2002) proposed a dynamic programming algorithm for estimating $p(T|\mathbf{w})$. They assumed that features are functions on nodes in a packed parse forest. That is, parse tree T is represented by a set of nodes, i.e., $T = \{c\}$, and the parse forest is represented by an and/or graph of the nodes. From this assumption, we can redefine the probability as

$$p(T|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} \exp\left(\sum_{c \in T} \sum_{i} \lambda_i f_i(c)\right)$$

$$Z_{\mathbf{w}} = \sum_{T'} \exp\left(\sum_{c \in T'} \sum_{i} \lambda_i f_i(c)\right).$$

A packed parse forest has a structure similar to a chart of CFG parsing, and c corresponds to an edge in the chart. This assumption corresponds to the independence assumption in PCFG; that is, only a nonterminal symbol of a mother is considered in further processing by ignoring the structure of its daughters. With this assumption, we can compute the figures of merit (FOMs) of partial parse results.

This assumption restricts the possibility of feature functions that represent non-local dependencies expressed in a parse result. Since unification-based grammars can express semantic relations, such as predicate-argument relations, in their structure, the assumption unjustifiably restricts the flexibility of probabilistic modeling. However, previous research (Miyao et al., 2003; Clark and Curran, 2004; Kaplan et al., 2004) showed that predicate-argument relations can be represented under the assumption of feature locality. We thus assumed the locality of feature functions and exploited it for the efficient search of probable parse results.

3 Techniques for efficient deep parsing

Many of the techniques for improving the parsing efficiency of deep linguistic analysis have been developed in the framework of lexicalized grammars such as lexical functional grammar (LFG) (Bresnan, 1982), lexicalized tree adjoining grammar (LTAG) (Shabes et al., 1988), HPSG (Pollard and Sag, 1994) or combinatory categorial grammar (CCG) (Steedman, 2000). Most of them were developed for exhaustive parsing, i.e., producing all parse results that are given by the grammar (Matsumoto et al., 1983; Maxwell and Kaplan, 1993; van Noord, 1997; Kiefer et al., 1999; Malouf et al., 2000; Torisawa et al., 2000; Oepen et al., 2002; Penn and Munteanu, 2003). The strategy of exhaustive parsing has been widely used in grammar development and in parameter training for probabilistic models.

We tested three of these techniques.

Quick check Quick check filters out non-unifiable feature structures (Malouf et al., 2000). Suppose we have two non-unifiable feature structures. They are destructively unified by traversing and modifying them, and then finally they are found to be not unifiable in the middle of the unification process. Quick check quickly judges their unifiability by peeping the values of the given paths. If one of the path values is not unifiable, the two feature structures cannot be unified because of the necessary condition of unification. In our implementation of quick check, each edge had two types of arrays. One contained the path values of the edge's sign; we call this the sign array. The other contained the path values of the right daughter of a schema such that its left daughter is unified with the edge's sign; we call this a schema array. When we apply a schema to two edges, e_1 and e_2, the schema array of e_1 and the sign array of e_2 are *quickly* checked. If it fails, then quick check returns a unification failure. If it succeeds, the signs are unified with the schemata, and the result of unification is returned.

Large constituent inhibition (Kaplan et al., 2004) It is unlikely for a large medial edge to contribute to the final parsing result if it spans more than 20 words and is not adjacent to the beginning or ending of the sentence. Large constituent inhibition prevents the parser from generating medial edges that span more than some word length.

HPSG parsing with a CFG chunk parser A hybrid of deep parsing and shallow parsing was recently found to improve the efficiency of deep parsing (Daum et al., 2003; Frank et al., 2003; Frank, 2004). As a preprocessor, the shallow parsing must be very fast and achieve high precision but not high recall so that the

```
procedure Viterbi(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ)
  for i = 1 to n
    foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
      α = Σ_i λ_i f_i(F_u)
      π[i−1, i] ← π[i−1, i] ∪ {F_u}
      if (α > ρ[i−1, i, F_u]) then
        ρ[i−1, i, F_u] ← α
  for d = 1 to n
    for i = 0 to n − d
      j = i + d
      for k = i + 1 to j − 1
        foreach F_s ∈ π[i, k], F_t ∈ π[k, j], r ∈ R
          if F = r(F_s, F_t) has succeeded
            α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
            π[i, j] ← π[i, j] ∪ {F}
            if (α > ρ[i, j, F]) then
              ρ[i, j, F] ← α
```

Figure 2: Pseudo-code of Viterbi algorithms for probabilistic HPSG parsing

total parsing performance in terms of precision, recall and speed is not degraded. Because there is trade-off between speed and accuracy in this approach, the total parsing performance for large-scale corpora like the Penn treebank should be measured. We introduce a CFG chunk parser (Tsuruoka and Tsujii, 2005a) as a preprocessor of HPSG parsing. Chunk parsers meet the requirements for preprocessors; they are very fast and have high precision. The grammar for the chunk parser is automatically extracted from the CFG treebank translated from the HPSG treebank, which is generated during grammar extraction from the Penn treebank. The principal idea of using the chunk parser is to use the bracket information, i.e., parse trees without non-terminal symbols, and prevent the HPSG parser from generating edges that cross brackets.

4 Beam thresholding for HPSG parsing

4.1 Simple beam thresholding

Many algorithms for improving the efficiency of PCFG parsing have been extensively investigated. They include grammar compilation (Tomita, 1986; Nederhof, 2000), the Viterbi algorithm, controlling search strategies without FOM such as left-corner parsing (Rosenkrantz and Lewis II, 1970) or head-corner parsing (Kay, 1989; van Noord, 1997), and with FOM such as the beam search, the best-first search or A* search (Chitrao and Grishman, 1990; Caraballo and Charniak, 1998; Collins, 1999; Ratnaparkhi, 1999; Charniak, 2000; Roark, 2001; Klein and Manning, 2003). The beam search and best-first search algorithms significantly reduce the time required for finding the best parse at the cost of losing the guarantee of finding the correct parse.

The CYK algorithm, which is essentially a bottom-up parser, is a natural choice for non-probabilistic HPSG parsers. Many of the constraints are expressed as lexical entries in HPSG, and bottom-up parsers can use those constraints to reduce the search space in the early stages of parsing.

For PCFG, extending the CYK algorithm to output the Viterbi parse is straightforward (Ney, 1991; Jurafsky and Martin, 2000). The parser can efficiently calculate the Viterbi parse by taking the maximum of the probabilities of the same nonterminal symbol in each cell. With the probabilistic model defined in Section 2, we can also define the Viterbi search for unification-based grammars (Geman and Johnson, 2002). Figure 2 shows the pseudo-code of Viterbi algorithm. The $\pi[i, j]$ represents the set of partial parse results that cover words $w_{i+1}, \ldots, w_j$, and $\rho[i, j, F]$ stores the maximum FOM of partial parse result F at cell (i, j). Feature functions are defined over lexical entries and results of rule applications, which correspond to conjunctive nodes in a feature forest. The FOM of a newly created partial parse, F, is computed by summing the values of ρ of the daughters and an additional FOM of F.

The Viterbi algorithm enables various pruning techniques to be used for efficient parsing. Beam thresholding (Goodman, 1997) is a simple and effective technique for pruning edges during parsing. In each cell of the chart, the method keeps only a portion of the edges which have higher FOMs compared to the other edges in the same cell.

```
procedure BeamThresholding(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ)
    for i = 1 to n
        foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
            α = Σ_i λ_i f_i(F_u)
            π[i − 1, i] ← π[i − 1, i] ∪ {F_u}
            if (α > ρ[i − 1, i, F_u]) then
                ρ[i − 1, i, F_u] ← α
    for d = 1 to n
        for i = 0 to n − d
            j = i + d
            for k = i + 1 to j − 1
                foreach F_s ∈ π[i, k], F_t ∈ π[k, j], r ∈ R
                    if F = r(F_s, F_t) has succeeded
                        α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
                        π[i, j] ← π[i, j] ∪ {F}
                        if (α > ρ[i, j, F]) then
                            ρ[i, j, F] ← α
            LocalThresholding(κ, δ)
        GlobalThresholding(n, θ)
```

```
procedure LocalThresholding(κ, δ)
    sort π[i, j] according to ρ[i, j, F]
    π[i, j] ← {π[i, j]_1, ..., π[i, j]_κ}
    α_max = max_F ρ[i, j, F]
    foreach F ∈ π[i, j]
        if ρ[i, j, F] < α_max − δ
            π[i, j] ← π[i, j]\{F}
```

```
procedure GlobalThresholding(n, θ)
    f[0..n] ← {0, −∞ − ∞, ..., −∞}
    b[0..n] ← {−∞, −∞, ..., −∞, 0}
    #forward
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                f[j] ← max(f[j], f[i] + ρ[i, j, F])
    #backward
    for i = n − 1 to 0
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                b[i] ← max(b[i], b[j] + ρ[i, j, F])
    #global thresholding
    α_max = f[n]
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                if f[i] + ρ[i, j, F] + b[j] < α_max − θ then
                    π[i, j] ← π[i, j]\{F}
```

Figure 3: Pseudo-code of local beam search and global beam search algorithms for probabilistic HPSG parsing

```
procedure IterativeBeamThresholding(w, G, κ0, δ0, θ0, Δκ, Δδ, Δθ, κlast, δlast, θlast)
  κ ← κ0; δ ← δ0; θ ← θ0
  loop while κ ≤ κlast and δ ≤ δlast and θ ≤ θlast
    call BeamThresholding(w, G, κ, δ, θ)
    if π[1, n] ≠ ∅ then exit
    κ ← κ + Δκ; δ ← δ + Δδ; θ ← θ + Δθ
```

Figure 4: Pseudo-code of iterative beam thresholding

We tested three selection schemes for deciding which edges to keep in each cell.

Local thresholding by number of edges Each cell keeps the top κ edges based on their FOMs.

Local thresholding by beam width Each cell keeps the edges whose FOM is greater than $\alpha_{\max} - \delta$, where $\alpha_{\max}$ is the highest FOM among the edges in the cell.

Global thresholding by beam width Each cell keeps the edges whose *global FOM* is greater than $\alpha_{\max} - \theta$, where $\alpha_{\max}$ is the highest global FOM in the chart.

Figure 3 shows the pseudo-code of local beam search, and global beam search algorithms for probabilistic HPSG parsing. The code for local thresholding is inserted at the end of the computation for each cell. In Figure 3, $\pi[i,j]_k$ denotes the k-th element in sorted set $\pi[i,j]$. We first take the first κ elements that have higher FOMs and then remove the elements with FOMs lower than $\alpha_{\max} - \delta$.

Global thresholding is also used for pruning edges, and was originally proposed for CFG parsing (Goodman, 1997). It prunes edges based on their global FOM and the best global FOM in the chart. The global FOM of an edge is defined as its FOM plus its forward and backward FOMs, where the forward and backward FOMs are rough estimations of the outside FOM of the edge. The global thresholding is performed immediately after each line of the CYK chart is completed. The forward FOM is calculated first, and then the backward FOM is calculated. Finally, all edges with a global FOM lower than $\alpha_{\max} - \theta$ are pruned. Figure 3 gives further details of the algorithm.

4.2 Iterative beam thresholding

We tested the iterative beam thresholding proposed by Tsuruoka and Tsujii (2005b). We started the parsing with a narrow beam. If the parser output results, they were taken as the final parse results. If the parser did not output any results, we widened the beam, and reran the parsing. We continued widening the beam until the parser output results or the beam width reached some limit.

Table 1: Abbreviations used in experimental results

num	local beam thresholding by number
width	local beam thresholding by width
global	global beam thresholding by width
iterative	iterative parsing with local beam thresholding by number and width
chp	parsing with CFG chunk parser

The pseudo-code is presented in Figure 4. It calls the beam thresholding procedure shown in Figure 3 and increases parameters κ and δ until the parser outputs results, i.e., $\pi[1,n] \neq \emptyset$.

Preserved iterative parsing Our implemented CFG parser with iterative parsing cleared the chart and edges at every iteration although the parser regenerated the same edges using those generated in the previous iteration. This is because the computational cost of regenerating edges is smaller than that of reusing edges to which the rules have already been applied. For HPSG parsing, the regenerating cost is even greater than that for CFG parsing. In our implementation of HPSG parsing, the chart and edges were not cleared during the iterative parsing. Instead, the pruned edges were marked as thresholded ones. The parser counted the number of iterations, and when edges were generated, they were marked with the iteration number, which we call the *generation*. If edges were thresholded out, the generation was replaced with the current iteration number plus 1. Suppose we have two edges, e_1 and e_2. The grammar rules are applied iff both e_1 and e_2 are not thresholded out, and the generation of e_1 or e_2 is equal to the current iteration number. Figure 5 shows the pseudo-code of preserved iterative parsing.

```
procedure BeamThresholding(⟨w_1, ..., w_n⟩, ⟨L', R⟩, κ, δ, θ, iternum)
    for i = 1 to n
        foreach F_u ∈ {F|⟨w_i, F⟩ ∈ L}
            α = Σ_i λ_i f_i(F_u)
            π[i−1, i] ← π[i−1, i] ∪ {F_u}
            if (α > ρ[i−1, i, F_u]) then
                ρ[i−1, i, F_u] ← α
    for d = 1 to n
        for i = 0 to n − d
            j = i + d
            for k = i + 1 to j − 1
                foreach F_s ∈ φ[i, k], F_t ∈ φ[k, j], r ∈ R
                    if gen[i, k, F_s] = iternum ∨ gen[k, j, F_t] = iternum
                        if F = r(F_s, F_t) has succeeded
                            gen[i, j, F] ← iternum
                            α = ρ[i, k, F_s] + ρ[k, j, F_t] + Σ_i λ_i f_i(F)
                            π[i, j] ← π[i, j] ∪ {F}
                            if (α > ρ[i, j, F]) then
                                ρ[i, j, F] ← α
            LocalThresholding(κ, δ, iternum)
        GlobalThresholding(n, θ, iternum)
```

```
procedure LocalThresholding(κ, δ, iternum)
    sort π[i, j] according to ρ[i, j, F]
    φ[i, j] ← {π[i, j]_1, ..., π[i, j]_κ}
    α_max = max_F ρ[i, j, F]
    foreach F ∈ φ[i, j]
        if ρ[i, j, F] < α_max − δ
            φ[i, j] ← φ[i, j]\{F}
    foreach F ∈ (π[i, j] − φ[i, j])
        gen[i, j, F] ← iternum + 1
```

```
procedure GlobalThresholding(n, θ, iternum)
    f[0..n] ← {0, −∞ − ∞, ..., −∞}
    b[0..n] ← {−∞, −∞, ..., −∞, 0}
    #forward
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                f[j] ← max(f[j], f[i] + ρ[i, j, F])
    #backward
    for i = n − 1 to 0
        for j = i + 1 to n
            foreach F ∈ π[i, j]
                b[i] ← max(b[i], b[j] + ρ[i, j, F])
    #global thresholding
    α_max = f[n]
    for i = 0 to n − 1
        for j = i + 1 to n
            foreach F ∈ φ[i, j]
                if f[i] + ρ[i, j, F] + b[j] < α_max − θ then
                    φ[i, j] ← φ[i, j]\{F}
            foreach F ∈ (π[i, j] − φ[i, j])
                gen[i, j, F] ← iternum + 1
```

```
procedure IterativeBeamThresholding(w, G, κ_0, δ_0, θ_0, Δκ, Δδ, Δθ, κ_last, δ_last, θ_last)
    κ ← κ_0; δ ← δ_0; θ ← θ_0; iternum = 0
    loop while κ ≤ κ_last and δ ≤ δ_last and θ ≤ θ_last
        call BeamThresholding(w, G, κ, δ, θ, iternum)
        if π[1, n] ≠ ∅ then exit
        κ ← κ + Δκ; δ ← δ + Δδ; θ ← θ + Δθ; iternum ← iternum + 1
```

Figure 5: Pseudo-code of preserved iterative parsing for HPSG

Table 2: Experimental results for development set (section 22) and test set (section 23)

	Precision	Recall	F-score	Avg. Time (ms)	No. of failed sentences
development set	88.21%	87.32%	87.76%	360	12
test set	87.85%	86.85%	87.35%	360	15

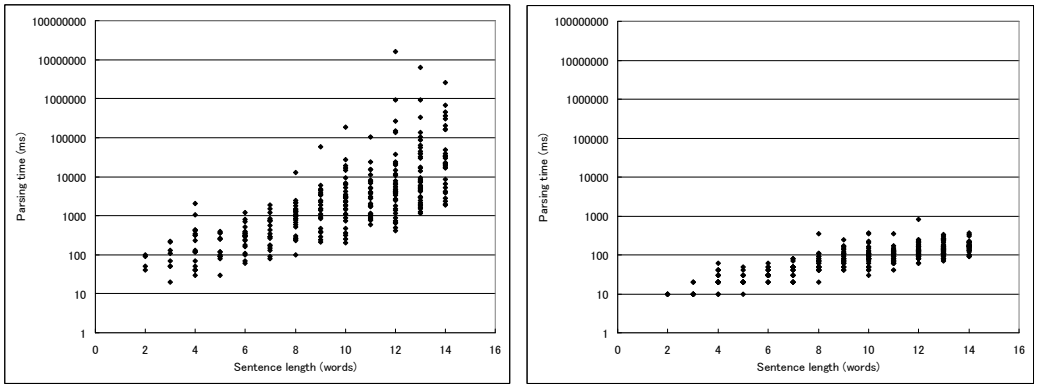


Figure 7: Parsing time for the sentences in Section 24 of less than 15 words of Viterbi parsing (none) (Left) and iterative parsing (iterative) (Right)

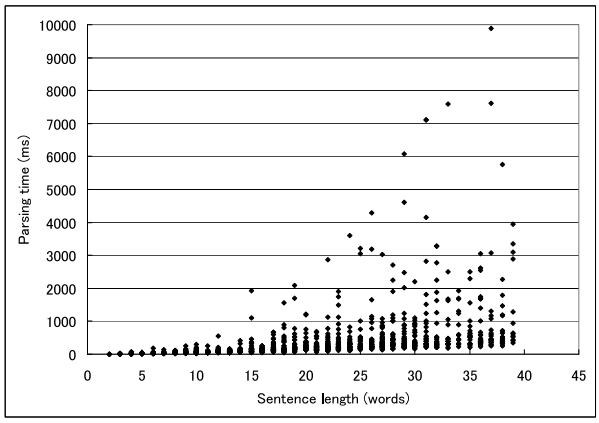


Figure 6: Parsing time versus sentence length for the sentences in Section 23 of less than 40 words

5 Evaluation

We evaluated the efficiency of the parsing techniques by using the HPSG for English developed by Miyao et al. (2005). The lexicon of the grammar was extracted from Sections 02-21 of the Penn Treebank (Marcus et al., 1994) (39,832 sentences). The grammar consisted of 2,284 lexical entry templates for 10,536 words[1]. The probabilistic disambiguation model of the grammar was trained using the same portion of the treebank (Miyao and Tsujii, 2005). The model included 529,856 features. The parameters for beam searching were determined manually by trial and error using Section 22; $\delta_0 = 12, \Delta\delta = 6, \delta_{\text{last}} = 30$, $\kappa_0 = 6.0, \Delta\kappa = 3.0, \kappa_{\text{last}} = 15.1$, $\theta_0 = 8.0, \Delta\theta = 4.0$, and $\theta_{\text{last}} = 20.1$. We used the chunk parser developed by Tsuruoka and Tsujii (2005a). Table 1 shows the abbreviations used in presenting the results.

We measured the accuracy of the predicate-argument relations output by the parser. A predicate-argument relation is defined as a tuple $\langle \sigma, w_h, a, w_a \rangle$, where σ is the predicate type (e.g., adjective, intransitive verb), w_h is the head word of the predicate, a is the argument label (MODARG, ARG1, ..., ARG4), and w_a is the head word of the argument. Precision/recall is the ratio of tuples correctly identified by the parser. This evaluation scheme was the same as used in previous evaluations of lexicalized grammars (Hockenmaier, 2003; Clark and Curran, 2004; Miyao and Tsujii, 2005). The experiments were conducted on an AMD Opteron server with a 2.4-GHz CPU. Section 22 of the Treebank was used as the development set, and performance was evaluated using sentences of less than 40 words in Section 23 (2,164 sentences, 20.3 words/sentence). The performance of each parsing technique was analyzed using the sentences in Section 24 of less than 15 words (305 sentences) and less than 40 words (1145 sentences).

Table 2 shows the parsing performance using all

[1]Lexical entry templates for POS are also developed. They are assigned to unknown words.

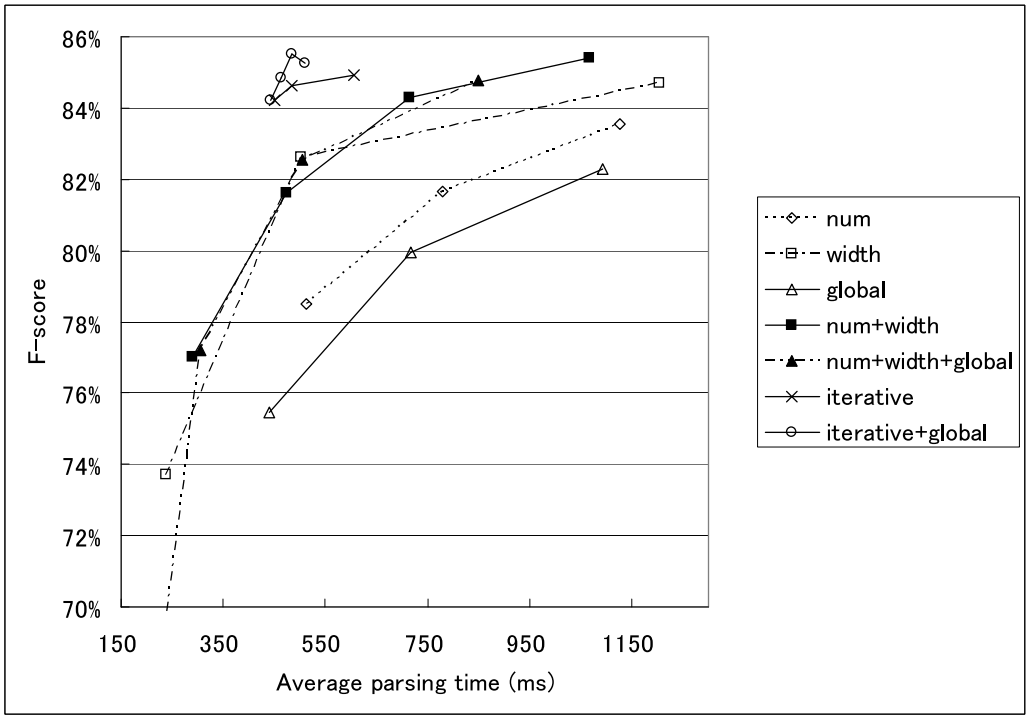


Figure 8: F-score versus average parsing time

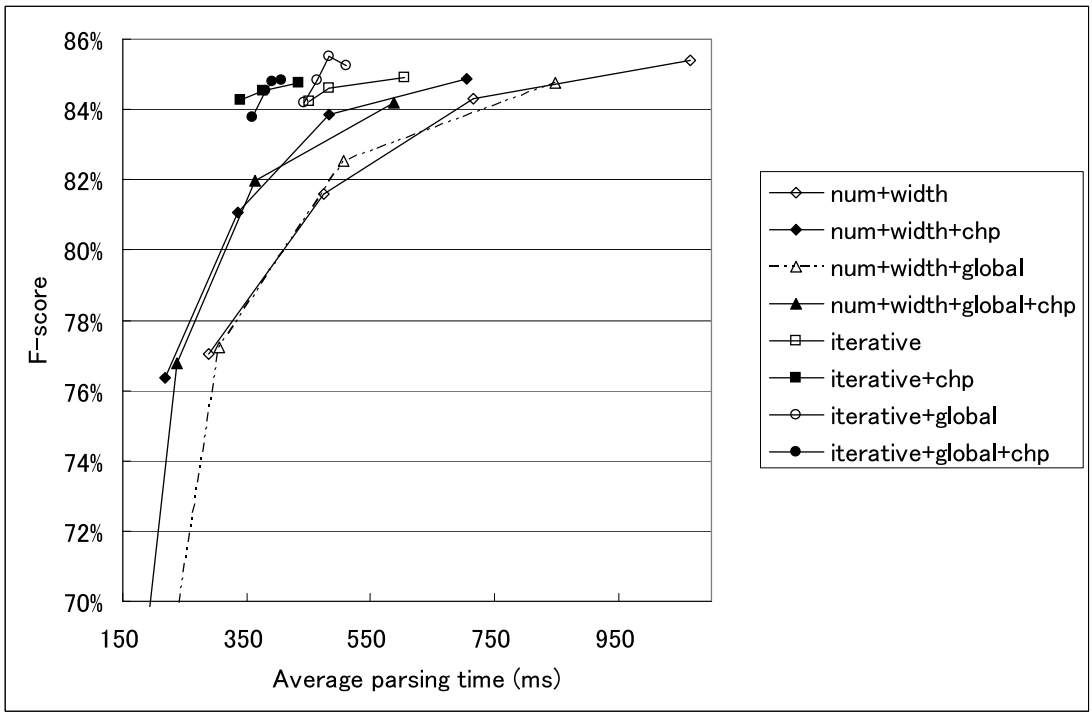


Figure 9: F-score versus average parsing time with/without chunk parser

Table 3: Viterbi parsing versus beam thresholding versus iterative parsing

	Precision	Recall	F-score	Avg. Time (ms)	No. of failed sentences
viterbi parsing (none)	88.22%	87.94%	88.08%	103923	2
beam search parsing (num+width)	88.96%	82.38%	85.54%	88	26
iterative parsing (iterative)	87.61%	87.24%	87.42%	99	2

Table 4: Contribution to performance of each implementation

	Precision	Recall	F-score	Avg. Time (ms)	diff(*)	No. of failed sentences
full	85.49%	84.21%	84.84%	407	0	13
full−piter	85.74%	84.70%	85.22%	631	224	10
full−qc	85.49%	84.21%	84.84%	562	155	13
full−chp	85.77%	84.76%	85.26%	510	103	10
full−global	85.23%	84.32%	84.78%	434	27	9
full−lci	85.68%	84.40%	85.03%	424	17	13
full−piter−qc−chp−global−lci	85.33%	84.71%	85.02%	1033	626	6

full ... iterative + global + chp
piter ... preserved iterative parsing
qc ... quick check
lci ... large constituent inhibition
diff(*) ... (Avg. Time of full) - (Avg. Time)

thresholding techniques and implementations described in Section 4 for the sentences in the development set (Section 22) and the test set (Section 23) of less than 40 words. In the table, precision, recall, average parsing time per sentence, and the number of sentences that the parser failed to parse are detailed. Figure 6 shows the distribution of parsing time for the sentence length.

Table 3 shows the performance of the Viterbi parsing, beam search parsing, and iterative parsing for the sentences in Section 24 of less than 15 words [2]. The parsing without beam searching took more than 1,000 times longer than with beam searching. However, the beam searching reduced the recall from 87.9% to 82.4%. The main reason for this reduction was parsing failure. That is, the parser could not output any results when the beam was too narrow instead of producing incorrect parse results. Although iterative parsing was originally developed for efficiency, the results revealed that it also increases the recall. This is because the parser continues trying until some results are output. Figure 7 shows the logarithmic graph of parsing time for the sentence length. The left side of the figure shows the parsing time of the Viterbi parsing and the right side shows the parsing time of the iterative parsing.

Figure 8 shows the performance of the parsing techniques for different parameters for the sentences in Section 24 of less than 40 words. The combinations of thresholding techniques achieved better results than the single techniques. Local thresholding using the width (width) performed better than that using the number (num). The combination of using width and number (num+width) performed better than single local and single global thresholding. The superiority of iterative parsing (iterative) was again demonstrated in this experiment. Although we did not observe significant improvement with global thresholding, the global plus iterative combination slightly improved performance.

Figure 9 shows the performance with and without the chunk parser. The lines with white symbols represent parsing without the chunk parser, and the lines with black symbols represent parsing with the chunk parser. The chunk parser improved the total parsing performance significantly. The improvements with global thresholding were less with the chunk parser.

Finally, Table 4 shows the contribution to performance of each implementation for the sentences in Section 24 of less than 40 words. The 'full' means the parser including all thresholding techniques and implementations described in Section 4. The 'full − x' means the full minus x. The preserved iterative parsing, the quick check, and the chunk parser greatly contributed to the final parsing speed, while the global thresholding and large constituent inhibition did not.

6 Conclusion

We have described the results of experiments with a number of existing techniques in head-driven phrase

[2]The sentence length was limited to 15 words because of inefficiency of Viterbi parsing

structure grammar (HPSG) parsing. Simple beam thresholding, similar to that for probabilistic CFG (PCFG) parsing, significantly increased the parsing speed over Viterbi algorithm, but reduced the recall because of parsing failure. Iterative parsing significantly increased the parsing speed without degrading precision or recall. We tested three techniques originally developed for deep parsing: quick check, large constituent inhibition, and HPSG parsing with a CFG chunk parser. The contributions of the large constituent inhibition and global thresholding were not significant, while the quick check and chunk parser greatly contributed to total parsing performance. The precision, recall and average parsing time for the Penn treebank (Section 23) were 87.85%, 86.85%, and 360 ms, respectively.

References

Steven P. Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Joan Bresnan. 1982. *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Ted Briscoe and John Carroll. 1993. Generalized probabilistic LR-parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. of NAACL-2000*, pages 132–139.

Mahesh V. Chitrao and Ralph Grishman. 1990. Edge-based best-first chart parsing. In *Proc. of the DARPA Speech and Natural Language Workshop*, pages 263–266.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL'04*, pages 104–111.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Univ. of Pennsylvania.

Michael Daum, Kilian A. Foth, and Wolfgang Menzel. 2003. Constraint based integration of deep and shallow parsing techniques. In *Proc. of EACL-2003*, pages 99–106.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schaefer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proc. of ACL'03*, pages 104–111.

Anette Frank. 2004. Constraint-based RMRS construction from shallow grammars. In *Proc. of COLING-2004*, pages 1269–1272.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proc. of ACL'02*, pages 279–286.

Joshua Goodman. 1997. Global thresholding and multiple pass parsing. In *Proc. of EMNLP-1997*, pages 11–25.

Julia Hockenmaier. 2003. Parsing with generative models of predicate-argument structure. In *Proc. of ACL'03*, pages 359–366.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proc. of ACL '99*, pages 535–541.

Dainiel Jurafsky and James H. Martin. 2000. *Speech and Language Processing*. Prentice Hall.

R. M. Kaplan, S. Riezler, T. H. King, J. T. Maxwell III, and A. Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc. of HLT/NAACL'04*.

Walter Kasper, Hans-Ulrich Krieger, Jörg Spilker, and Hans Weber. 1996. From word hypotheses to logical form: An efficient interleaved approach. In *Proceedings of Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, pages 77–88.

Martin Kay. 1989. Head driven parsing. In *Proc. of IWPT'89*, pages 52–62.

Bernd Kiefer, Hans-Ulrich Krieger, John Carroll, and Robert Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proc. of ACL'99*, pages 473–480, June.

Bernd Kiefer, Hans-Ulrich Krieger, and Detlef Prescher. 2002. A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In *Proc. of COLING 2002*.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Proc. of HLT-NAACL'03*.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proc. of IJCNLP-04 Workshop "Beyond Shallow Analyses"*.

Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6(1):29–46.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. of CoNLL-2002*, pages 49–55.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Yuji Matsumoto, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. 1983. BUP: A bottom up parser embedded in Prolog. *New Generation Computing*, 1(2):145–158.

John Maxwell and Ron Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proc. of HLT 2002*, pages 292–297.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. of ACL'05*, pages 83–90.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 2003. Probabilistic modeling of argument structures including non-local dependencies. In *Proc. of RANLP '03*, pages 285–291.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii, 2005. *Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (Eds.), Natural Language Processing - IJCNLP 2004 LNAI 3248*, chapter Corpus-oriented Grammar Development for Acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank, pages 684–693. Springer-Verlag.

Mark-Jan Nederhof. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44.

H. Ney. 1991. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340.

Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*. CSLI Publications.

Gerald Penn and Cosmin Munteanu. 2003. A tabulation-based parsing method that reduces copying. In *Proc. of ACL'03)*.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proc. of ACL'00*, pages 480–487.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Daniel J. Rosenkrantz and Philip M. Lewis II. 1970. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

Yves Shabes, Anne Abeillè, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proc. of COLING'88*, pages 578–583.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

Masaru Tomita. 1986. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers.

Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Journal of Natural Language Engineering*, 6(1):63–80.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005a. Chunk parsing revisited. In *Proc. of IWPT-2005*.

Yoshimasa Tsuruoka and Jun'ichi Tsujii, 2005b. *Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee and Oi Yee Kwong (Eds.), Natural Language Processing - IJCNLP 2004 LNAI 3248*, chapter Iterative CKY Parsing for Probabilistic Context-Free Grammars, pages 52–60. Springer-Verlag.

Gertjan van Noord. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456.

Head-Driven PCFGs with Latent-Head Statistics

Detlef Prescher
Institute for Logic, Language and Computation
University of Amsterdam
prescher@science.uva.nl

Abstract

Although *state-of-the-art* parsers for natural language are lexicalized, it was recently shown that an *accurate* unlexicalized parser for the Penn tree-bank can be simply read off a manually refined tree-bank. While *lexicalized* parsers often suffer from sparse data, *manual mark-up* is costly and largely based on individual linguistic intuition. Thus, across domains, languages, and tree-bank annotations, a fundamental question arises: Is it possible to *automatically* induce an *accurate* parser from a tree-bank without resorting to full lexicalization? In this paper, we show how to induce head-driven probabilistic parsers with latent heads from a tree-bank. Our automatically trained parser has a performance of 85.7% (LP/LR F_1), which is already better than that of early *lexicalized* ones.

1 Introduction

State-of-the-art statistical parsers for natural language are based on probabilistic grammars acquired from transformed tree-banks. The method of transforming the tree-bank is of major influence on the accuracy and coverage of the statistical parser. The most important tree-bank transformation in the literature is lexicalization: Each node in a tree is labeled with its head word, the most important word of the constituent under the node (Magerman (1995), Collins (1996), Charniak (1997), Collins (1997), Carroll and Rooth (1998), etc.). It turns out, however, that lexicalization is not unproblematic: First, there is evidence that full lexicalization does not carry over across different tree-banks for other languages, annotations or domains (Dubey and Keller, 2003). Second, full lexicalization leads to a serious sparse-data problem, which can only be solved by sophisticated smoothing and pruning techniques.

Recently, Klein and Manning (2003) showed that a carefully performed linguistic mark-up of the tree-bank leads to almost the same performance results as lexicalization. This result is attractive since unlexicalized grammars are easy to estimate, easy to parse with, and time- and space-efficient: Klein and Manning (2003) do not smooth grammar-rule probabilities, except unknown-word probabilities, and they do not prune since they are able to determine the most probable parse of each *full* parse forest. Both facts are noteworthy in the context of statistical parsing with a tree-bank grammar. A drawback of their method is, however, that manual linguistic mark-up is not based on abstract rules but rather on individual linguistic intuition, which makes it difficult to repeat their experiment and to generalize their findings to languages other than English.

Is it possible to automatically acquire a more refined probabilistic grammar from a given tree-bank without resorting to full lexicalization? We present a novel method that is able to induce a parser that is located between two extremes: a fully-lexicalized parser on one side *versus* an accurate unlexicalized parser based on a manually refined tree-bank on the other side.

In short, our method is based on the same linguistic principles of headedness as other methods: We do believe that lexical information represents an important knowledge source. To circumvent data sparseness resulting from full lexicalization

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 115–124,
Vancouver, October 2005.

with words, we simply follow the suggestion of various advanced linguistic theories, e.g. Lexical-Functional Grammar (Bresnan and Kaplan, 1982), where more complex categories based on feature combinations represent the lexical effect. We complement this by a learning paradigm: lexical entries carry latent information to be used as head information, and this head information is induced from the tree-bank.

In this paper, we study two different latent-head models, as well as two different estimation methods: The first model is built around completely hidden heads, whereas the second one uses relatively fine-grained combinations of Part-Of-Speech (POS) tags with hidden extra-information; The first estimation method selects a head-driven probabilistic context-free grammar (PCFG) by exploiting latent-head distributions for each node in the tree-bank, whereas the second one is more traditional, reading off the grammar from the tree-bank annotated with the most probable latent heads only. In other words, both models and estimation methods differ in the degree of information incorporated into them as prior knowledge. In general, it can be expected that the better (sharper or richer, or more accurate) the information is, the better the induced grammar will be. Our empirical results, however, are surprising: First, estimation with latent-head distributions outperforms estimation with most-probable-head annotation. Second, modeling with completely hidden heads is almost as good as modeling with latent heads based on POS tags, and moreover, results in much smaller grammars.

We emphasize that our task is to automatically induce a more refined grammar based on a few linguistic principles. With automatic refinement it is harder to guarantee improved performance than with manual refinements (Klein and Manning, 2003) or with refinements based on direct lexicalization (Magerman (1995), Collins (1996), Charniak (1997), etc.). If, however, our refinement provides improved performance then it has a clear advantage: it is automatically induced, which suggests that it is applicable across different domains, languages and tree-bank annotations.

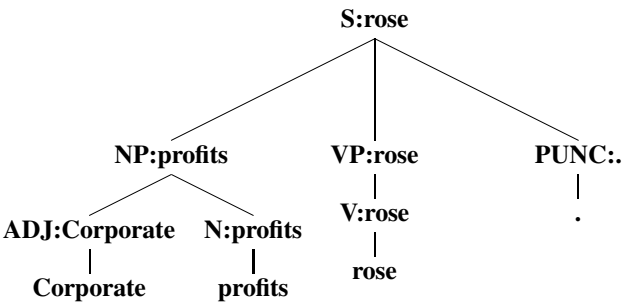
Applying our method to the benchmark Penn tree-bank Wall-Street Journal, we obtain a refined probabilistic grammar that significantly improves over the original tree-bank grammar and that shows performance that is on par with early work on lexicalized probabilistic grammars. This is a promising result given the hard task of automatic induction of improved probabilistic grammars.

2 Head Lexicalization

As previously shown (Charniak (1997), Collins (1997), Carroll and Rooth (1998), etc.), Context-Free Grammars (CFGs) can be transformed to lexicalized CFGs, provided that a head-marking scheme for rules is given. The basic idea is that the head marking on rules is used to project lexical items up a chain of nodes. Figure 1 displays an example.

In this Section, we focus on the approaches of Charniak (1997) and Carroll and Rooth (1998). These approaches are especially attractive for us for two reasons: First, both approaches make use of an *explicit linguistic grammar*. By contrast, alternative approaches, like Collins (1997), apply an additional transformation to each tree in the tree-bank, splitting each rule into small parts, which finally results in a new grammar covering many more sentences than the explicit one. Second, Charniak (1997) and Carroll and Rooth (1998) rely on almost the same lexicalization technique. In fact, the significant difference between them is that, in one case, a lexicalized version of the *tree-bank grammar* is learned from a corpus of trees (supervised learning), whereas, in the other case, a lexicalized version of a *manually written CFG* is learned from a a text corpus (unsupervised learning). As we will see in Section 3, our approach is a blend of these approaches in that it aims at unsupervised learning of a (latent-head-) lexicalized version of the tree-bank grammar.

Starting with Charniak (1997), Figure 2 displays an internal rule as it is used in the parse in Figure1, and its probability as defined by Charniak. Here, H is the head-child of the rule, which inherits the head h from its parent C. The children D_1:d_1, ..., D_m:d_m and D_{m+1}:d_{m+1}, ..., D_{m+n}:d_{m+n} are left and right modifiers of H. Either n or m may be zero, and $n = m = 0$ for unary rules. Because the probabilities occurring in Charniak's definition are already so specific that there is no real chance of obtaining the data empirically, they are smoothed by deleted interpolation:



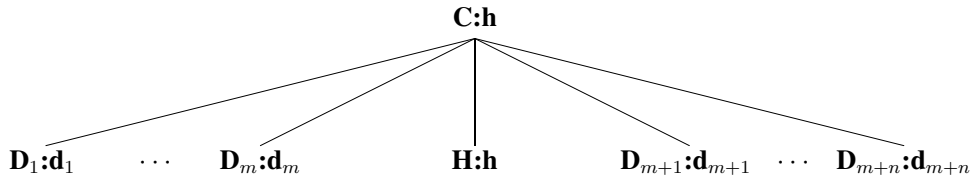
Internal Rules:

S:rose ⟶ NP:profits VP:rose PUNC:.
NP:profits ⟶ ADJ:Corporate N:profits
VP:rose ⟶ V:rose

Lexical Rules:

ADJ:Corporate ⟶ Corporate
N:profits ⟶ profits
V:rose ⟶ rose
PUNC:. ⟶ .

Figure 1: Parse tree, and a list of the rules it contains (Charniak, 1997)



$$p_{\textbf{CHARNIAK97}}(\ \textbf{this local tree}\ ) \quad = \quad p(\ r \mid \mathbf{C}, h, \mathbf{C}_p\ ) \ \times\ \prod_{i=1}^{n+m}\ p(\ d_i \mid \mathbf{D}_i, \mathbf{C}, h\ )$$

(r **is the *unlexicalized* rule,**
$\mathbf{C}_p$ **is C's parent category**)

Figure 2: Internal rule, and its probability (Charniak, 1997)

$$\begin{aligned} p(\ r \mid \mathrm{C},\ h,\ \mathrm{C}_p\ ) \ = \quad & \lambda_1 \cdot \hat{p}(\ r \mid \mathrm{C},\ h,\ \mathrm{C}_p\ ) \\ & + \lambda_2 \cdot \hat{p}(\ r \mid \mathrm{C},\ h\ ) \\ & + \lambda_3 \cdot \hat{p}(\ r \mid \mathrm{C},\ \mathrm{class}(h)\ ) \\ & + \lambda_4 \cdot \hat{p}(\ r \mid \mathrm{C},\ \mathrm{C}_p\ ) \\ & + \lambda_5 \cdot \hat{p}(\ r \mid \mathrm{C}\ ) \end{aligned}$$

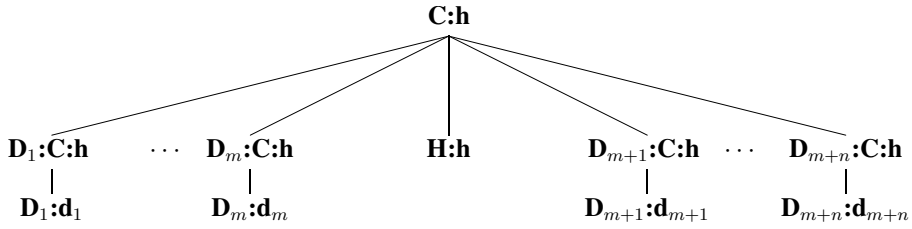
$$\begin{aligned} p(\ d \mid \mathrm{D},\ \mathrm{C},\ h\ ) \ = \quad & \lambda_1 \cdot \hat{p}(\ d \mid \mathrm{D},\ \mathrm{C},\ h\ ) \\ & + \lambda_2 \cdot \hat{p}(\ d \mid \mathrm{D},\ \mathrm{C},\ \mathrm{class}(h)\ ) \\ & + \lambda_3 \cdot \hat{p}(\ d \mid \mathrm{D},\ \mathrm{C}\ ) \\ & + \lambda_4 \cdot \hat{p}(\ d \mid \mathrm{D}\ ) \end{aligned}$$

Here, class(h) denotes a class for the head word h. Charniak takes these word classes from an *external* distributional clustering model, but does not describe this model in detail.

An at a first glance different lexicalization technique is described in Carroll and Rooth (1998). In their approach, a grammar transformation is used to lexicalize a manually written grammar. The key step for understanding their model is to imagine that the rule in Figure 2 is transformed to a *sub-tree*, the one displayed in Figure 3. After this transformation, the sub-tree probability is simply calculated with the PCFG's standard model; The result is also displayed in the figure. Comparing this probability with the probability that Charniak assigns to the rule itself, we see that the subtree probability equals the rule probability[1]. In other words, both probability models are based on the same idea for lexicalization, but the type of the corpora they are estimated from differ (*trees* versus *sentences*).

In more detail, Table 1 displays all four grammar-rule types resulting from the grammar transformation of Carroll and Rooth (1998). The underlying entities from the original CFG are: The starting symbol S (also the starting symbol of the transform), the internal rule C ⟶ $\mathrm{D}_1 \ldots \mathrm{D}_m\ \mathrm{H}\ \mathrm{D}_{m+1} \ldots \mathrm{D}_{m+n}$, and the lexical rule C ⟶ w. From these, the context-free transforms are generated as displayed in the table (for all possible head words h and d, and for all non-head children D=D_1, …, D_{m+n}). Figure 4 displays an example parse on the basis of the

[1] at least, if we ignore Charniak's conditioning on C's parent category C_p for the moment; Note that C's parent category is available in the tree-bank, but may not occur in the left-hand sides of the rules of a manually written CFG



$p_{\text{STANDARD-PCFG}}(\ \textbf{this sub-tree}\ )$

$$= p(\ \mathbf{D}_1{:}\mathbf{C}{:}h \ldots \mathbf{D}_m{:}\mathbf{C}{:}h\ \ \mathbf{H}{:}h\ \ \mathbf{D}_{m+1}{:}\mathbf{C}{:}h \ldots \mathbf{D}_{m+n}{:}\mathbf{C}{:}h\ \mid\ \mathbf{C}{:}h\ )\ \times\ \textstyle\prod_{i=1}^{m+n}\ p(\ \mathbf{D}_i{:}d_i\ \mid\ \mathbf{D}_i{:}\mathbf{C}{:}h\ )$$

$$= p(\ \mathbf{D}_1 \ldots \mathbf{D}_m\ \mathbf{H}\ \mathbf{D}_{m+1} \ldots \mathbf{D}_{m+n}\ \mid\ \mathbf{C}, h\ )\ \times\ \textstyle\prod_{i=1}^{m+n}\ p(\ d_i\ \mid\ \mathbf{D}_i, \mathbf{C}, h\ )$$

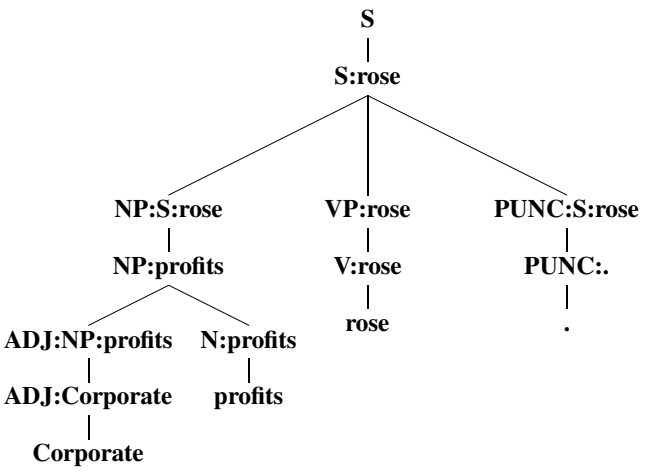
$$= p(\ r\ \mid\ \mathbf{C}, h\ )\ \times\ \textstyle\prod_{i=1}^{m+n}\ p(\ d_i\ \mid\ \mathbf{D}_i, \mathbf{C}, h\ )$$

(r **is the *unlexicalized* rule**)

Figure 3: Transformed internal rule, and its standard-PCFG probability (Carroll and Rooth, 1998)

S	$\longrightarrow$	S:h	**(Starting Rules)**
C:h	$\longrightarrow$	D_1:C:h ... D_m:C:h H:h D_{m+1}:C:h ... D_{m+n}:C:h	**(Lexicalized Rules)**
D:C:h	$\longrightarrow$	D:d	**(Dependencies)**
C:w	$\longrightarrow$	w	**(Lexical Rules)**

Table 1: Context-free rule types in the transform (Carroll and Rooth, 1998)



Starting Rule:
S $\longrightarrow$ S:rose
Lexicalized Rules:
S:rose $\longrightarrow$ NP:S:rose VP:rose PUNC:S:rose
NP:profits $\longrightarrow$ ADJ:NP:profits N:profits
VP:rose $\longrightarrow$ V:rose
Dependencies:
NP:S:rose $\longrightarrow$ NP:profits
PUNC:S:rose $\longrightarrow$ PUNC:.
ADJ:NP:profits $\longrightarrow$ ADJ:Corporate
Lexical Rules:
ADJ:Corporate $\longrightarrow$ Corporate
N:profits $\longrightarrow$ profits
V:rose $\longrightarrow$ rose
PUNC:. $\longrightarrow$.

Figure 4: Transformed parse tree, and a list of the rules it contains (Carroll and Rooth, 1998)

transformed grammar. It is noteworthy that although Carroll and Rooth (1998) learn from a text corpus of about 50 million words, it is still necessary to smooth the rule probabilities of the transform. Unlike Charniak (1997), however, they do not use word classes in their back-off scheme.

To summarize, the major problem of full-lexicalization techniques is that they lead to serious sparse-data problems. For both models presented in this section, a large number $|T|$ of full word forms makes it difficult to reliably estimate the probability weights of the $O(|T|^2)$ dependencies and the $O(|T|)$ lexicalized rules.

A linguistically naive approach to this problem is to use POS tags as heads to decrease the number of heads. From a computational perspective, the sparse data problem would then be completely solved since the number |POS| of POS tags is tiny compared to the number $|T|$ of full-word forms. Although we will demonstrate that parsing results benefit already from this naive lexicalization routine, we expect that (computationally and linguistically) optimal head-lexicalized models are arranged around a number |HEADS| of head elements such that $|\text{POS}| \leq |\text{HEADS}| \ll |T|$.

3 Latent-Head Models

This section defines two probability models over the trees licensed by a head-lexicalized CFG with latent head-information, thereby exploiting three simple linguistic principles: (i) all rules have head markers, (ii) information is projected up a chain of categories marked as heads, (iii) lexical entries carry latent head values which can be learned. Moreover, two estimation methods for the latent-head models are described.

Head-Lexicalized CFGs with Latent Heads

Principles (i) and (ii) are satisfied by all head lexicalized models we know of, and clearly, they are also satisfied by the model of Carroll and Rooth (1998). Principle (iii), however, deals with latent information for lexical entries, which is beyond the capability of this model. To see this, remember that lexical rules $\text{C} \longrightarrow w$ are unambiguously transformed to $\text{C}{:}w \longrightarrow w$. Because this transformation is unambiguous, latent information does not play a role in it. It is surprisingly simple, however, to satisfy principle (iii) with slightly modified versions of Carroll and Rooth's transformation of lexical rules. In the following, we present two of them:

Lexical-Rule Transformation (Model 1): Transform each lexical rule $\text{C} \longrightarrow w$ to a set of rules, having the form $\text{C}{:}h \longrightarrow w$, where $h \in \{1, \ldots, L\}$, and L is a free parameter.

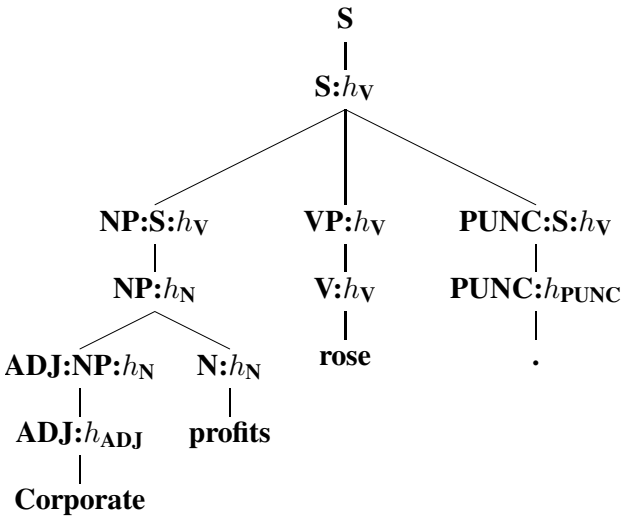
Lexical-Rule Transformation (Model 2): Transform each lexical rule $\text{C} \longrightarrow w$ to a set of rules, having the form $\text{C}{:}h \longrightarrow w$, where $h \in \{\text{C}\} \times \{1, \ldots, L\}$, and L is a free parameter.

Both models introduce latent heads for lexical entries. The difference is that Model 1 introduces completely latent heads h, whereas Model 2 introduces heads h on the basis of the POS tag C of the word w: each such head is a combination of C with an abstract extra-information. Figure 5 gives an example. Because we still apply Carroll and Rooth's grammar transformation scheme to the non-lexical rules, latent heads are percolated up a path of categories marked as heads.

Although our modifications are small, their effect is remarkable. In contrast to Carroll and Rooth (1998), where an unlexicalized tree is unambiguously mapped to a *single* transform, our models map an unlexicalized tree to *multiple* transforms (for free parameters ≥ 2). Note also that although latent information is freely introduced at the lexical level, it is not freely distributed over the nodes of the tree. Rather, the space of latent heads for a tree is constrained according the linguistic principle of headedness. Finally, for the case $L = 1$, our models perform unambiguous transformations: in Model 1 the transformation makes no relevant changes, whereas Model 2 performs unambiguous lexicalization with POS tags. In the rest of the paper, we show how to learn models with hidden, richer, and more accurate head-information from a tree-bank, if $L \geq 2$.

Unsupervised Estimation of Head-Lexicalized CFGs with Latent Heads

In the following, we define two methods for estimating latent-head models. The main difficulty here is that the rules of a head-lexicalized CFG



Starting Rule:

S $\longrightarrow$ S:h_{V}

Lexicalized Rules:

S:h_{V} $\longrightarrow$ NP:S:h_{V} VP:h_{V} PUNC:S:h_{V}

NP:h_{N} $\longrightarrow$ ADJ:NP:h_{N} N:h_{N}

VP:h_{V} $\longrightarrow$ V:h_{V}

Dependencies:

NP:S:h_{V} $\longrightarrow$ NP:h_{N}

PUNC:S:h_{V} $\longrightarrow$ PUNC:h_{PUNC}

ADJ:NP:h_{N} $\longrightarrow$ ADJ:h_{ADJ}

Lexical Rules:

ADJ:h_{ADJ} $\longrightarrow$ Corporate

N:h_{N} $\longrightarrow$ profits

V:h_{V} $\longrightarrow$ rose

PUNC:h_{PUNC} $\longrightarrow$.

Model 1 (Completely Latent Heads):

h_{ADJ}, h_{N}, h_{V}, and $h_{\text{PUNC}} \in \{1, \ldots, L\}$

Model 2 (Latent Heads Based on POS Tags):

$h_{\text{ADJ}} \in \{\text{ADJ}\} \times \{1, \ldots, L\}$

$h_{\text{N}} \in \{\text{N}\} \times \{1, \ldots, L\}$

$h_{\text{V}} \in \{\text{V}\} \times \{1, \ldots, L\}$

$h_{\text{PUNC}} \in \{\text{PUNC}\} \times \{1, \ldots, L\}$

Number of Latent-Head Types $= \begin{cases} L & \text{for Model 1} \\ |POS| \times L & \text{for Model 2} \end{cases}$ (L is a free parameter)

Figure 5: Parse tree with latent heads, and a list of the rules it contains.

Initialization: Generate a randomly initialized distribution p_0 for the rules of G_{LEX} (a head-lexicalized CFG with latent heads as previously defined).

Iterations:

(1) for each $i = 1, 2, 3, \ldots$, *number_of_iterations* do
(2) set $p = p_{i-1}$
(3) **E step**: Generate a lexicalized tree-bank T_{LEX}, by
- running over all unlexicalized trees t of the original tree-bank
- generating the finite set $G_{\text{LEX}}(t)$ of the lexicalized transforms of t
- allocating the frequency $\text{c}(t') = \text{c}(t) \cdot p(\ t' \mid t\ )$ to the lexicalized trees $t' \in G_{\text{LEX}}(t)$

[Here, $c(t)$ is the frequency of t in the original tree-bank]

(4) **M step**: Read the tree-bank grammar off T_{LEX}, by
- calculating relative frequencies $\hat{p}$ for all rules of G_{LEX} as occurring in T_{LEX}

(5) set $p_i = \hat{p}$
(6) end

Figure 6: Grammar induction algorithm (EM algorithm)

with latent heads cannot be directly estimated from the tree-bank (by counting rules) since the latent heads are not annotated in the trees. Faced with this incomplete-data problem, we apply the Expectation-Maximization (EM) algorithm developed for these type of problems (Dempster et al., 1977). For details of the EM algorithm, we refer to the numerous tutorials on EM (e.g. Prescher (2003)). Here, it suffices to know that it is a sort of meta algorithm, resulting for each incomplete-data problem in an iterative estimation method that aims at maximum-likelihood estimation on the data. Disregarding the fact that we implement a dynamic-programming version for our experiments (running in linear time in the size of the trees in the tree-bank (Prescher, 2005)), the EM algorithm is here as displayed in Figure 6. Beside this pure form of the EM algorithm, we also use a variant where the original tree-bank is annotated with most probable heads only. Here is a characterization of both estimation methods:

Estimation from latent-head distributions: The key steps of the EM algorithm produce a lexicalized tree-bank T_{LEX}, consisting of all lexicalized versions of the original trees (E-step), and calculate the probabilities for the rules of G_{LEX} on the basis of T_{LEX} (M-step). Clearly, all lexicalized trees in $G_{\text{LEX}}(t)$ differ only in the heads of their nodes. Thus, EM estimation uses the original tree-bank, where each node can be thought of as annotated with a *latent-head distribution*.

Estimation from most probable heads: By contrast, a quite different scheme is applied in Klein and Manning (2003): extensive manual annotation enriches the tree-bank with information, but no trees are added to the tree-bank. We borrow from this scheme in that we take the best EM model to calculate the most probable head-lexicalized versions of the trees in the original tree-bank. After collecting this Viterbi-style lexicalized tree-bank, the ordinary tree-bank estimation yields another estimate of G_{LEX}. Clearly, this estimation method uses the original tree-bank, where each node can be thought of annotated with the *most probable latent head*.

4 Experiments

This section presents empirical results across our models and estimation methods.

Data and Parameters

To facilitate comparison with previous work, we trained our models on sections 2-21 of the WSJ section of the Penn tree-bank (Marcus et al., 1993). All trees were modified such that: The empty top node got the category TOP, node labels consisted solely of syntactic category information, empty nodes (i.e. nodes dominating the empty string) were deleted, and words in rules occurring less than 3 times in the tree-bank were replaced by (word-suffix based)

	Estimation from most probable heads				Estimation from head distributions			
	Model 1 (completely latent)		**Model 2** (POS+latent)		**Model 1** (completely latent)		**Model 2** (POS+latent)	
baseline	(15 400)	73.5	(25 000)	78.9	(15 400)	73.5	(25 000)	78.9
L=2	(17 900)	76.3	(32 300)	81.1	(25 900)	76.9	(49 500)	81.6
L=5	(22 800)	80.7	(46 200)	*83.3*	(49 200)	82.0	(116 300)	84.9
L=10	(28 100)	*83.3*	(58 900)	82.6	(79 200)	*84.6*	(224 300)	**85.7**
		Δ=9.8		Δ=4.4		Δ=**11.1**		Δ=6.8

Table 2: Parsing results in LP/LR F_1 (the baseline is $L = 1$)

unknown-word symbols. No other changes were made.

On this tree-bank, we trained several head-lexicalized CFGs with latent-heads as described in Section 3, but smoothed the grammar rules using deleted interpolation; We also performed some preliminary experiments without smoothing, but after observing that about 3000 trees of our training corpus were allocated a zero-probability (resulting from the fact that too many grammar rules got a zero-probability), we decided to smooth all rule probabilities.

We tried to find optimal starting parameters by repeating the whole training process multiple times, but we observed that starting parameters affect final results only up to 0.5%. We also tried to find optimal iteration numbers by evaluating our models after each iteration step on a held-out corpus, and observed that the best results were obtained with 70 to 130 iterations. Within a wide range from 50 to 200 iteration, however, iteration numbers affect final results only up to 0.5%

Empirical Results

We evaluated on a parsing task performed on Section 22 of the WSJ section of the Penn tree-bank. For parsing, we mapped all unknown words to unknown word symbols, and applied the Viterbi algorithm as implemented in Schmid (2004), exploiting its ability to deal with highly-ambiguous grammars. That is, we did not use any pruning or smoothing routines for parsing sentences. We then de-transformed the resulting maximum-probability parses to the format described in the previous sub-section. That is, we deleted the heads, the dependencies, and the starting rules. All grammars were able to exhaustively parse the evaluation corpus. Table 2 displays our results in terms of LP/LR F_1 (Black and al., 1991). The largest number per column is printed in italics. The absolutely largest number is printed in boldface. The numbers in brackets are the number of grammar rules (without counting lexical rules). The gain in LP/LR F_1 per estimation method and per model is also displayed (Δ). Finally, the average training time per iteration ranges from 2 to 4 hours (depending on both L and the type of the model). The average parsing time is 10 seconds per sentence, which is comparable to what is reported in Klein and Manning (2003).

5 Discussion

First of all, all model instances outperform the baseline, i.e., the original grammar (F_1=73.5), and the head-lexicalized grammar with POS tags as heads (F_1=78.9). The only plausible explanation for these significant improvements is that useful head classes have been learned by our method. Moreover, increasing L consistently increases F_1 (except for Model 2 estimated from most probable heads; $L = 10$ is out of the row). We thus argue that the granularity of the current head classes is not fine enough; Further refinement may lead to even better latent-head statistics.

Second, estimation from head distributions consistently outperforms estimation from most probable heads (for both models). Although coarse-grained models clearly benefit from POS information in the heads ($L = 1, 2, 5$), it is surprising that the *best* models with completely latent heads are on a par with or almost as good as the *best* ones using POS

	LP	LR	F_1	Exact	CB
Model 1 (this paper)	84.8	84.4	84.6	26.4	1.37
Magerman (1995)	84.9	84.6			1.26
Model 2 (this paper)	85.7	85.7	85.7	29.3	1.29
Collins (1996)	86.3	85.8			1.14
Matsuzaki etal. (2005)	86.6	86.7			1.19
Klein and Manning (2003)	86.9	85.7	86.3	30.9	1.10
Charniak (1997)	87.4	87.5			1.00
Collins (1997)	88.6	88.1			0.91

Table 3: Comparison with other parsers (sentences of length ≤ 40)

as head information.

Finally, our absolutely best model (F_1=85.7) combines POS tags with latent extra-information ($L = 10$) and is estimated from latent-head distributions. Although it also has the largest number of grammar rules (about 224 300), it is still much smaller than fully-lexicalized models. The best model with completely latent heads, however, leads to almost the same performance (F_1=84.6), and has the further advantage of having significantly fewer rules (only about 79 200). Moreover, it is the model which leads to the largest gain compared to the baseline ($\Delta = 11.1$).

In the rest of the section, we compare our method to related methods. To start with performance values, Table 3 displays previous results on parsing Section 23 of the WSJ section of the Penn tree-bank. Comparison indicates that our best model is already better than the early lexicalized model of Magerman (1995). It is a bit worse than the unlexicalized PCFGs of Klein and Manning (2003) and Matsuzaki et al. (2005), and of course, it is also worse than state-of-the-art lexicalized parsers (experience shows that evaluation results on sections 22 and 23 do not differ much).

Beyond performance values, we believe our formalism and methodology have the following attractive features: first, our models incorporate context and lexical information collected from the whole tree-bank. Information is bundled into abstract heads of higher-order information, which results in a drastically reduced parameter space. In terms of Section 2, our approach does not aim at improving the approximation of rule probabilities $p(r|\mathrm{C}, h)$ and dependency probabilities $p(d|\mathrm{D}, \mathrm{C}, h)$ by smoothing. Rather, our approach induces head classes for the words h and d from the tree-bank and aims at a exact calculation of rule probabilities $p(r|\mathrm{C}, \mathrm{class}(h))$ and dependency probabilities $p(\mathrm{class}(d)|\mathrm{D}, \mathrm{C}, \mathrm{class}(h))$. This is in sharp contrast to the smoothed fixed-word statistics in most lexicalized parsing models derived from sparse data (Magerman (1995), Collins (1996), Charniak (1997), etc.). Particularly, class-based dependency probabilities $p(\mathrm{class}(d)|\mathrm{D}, \mathrm{C}, \mathrm{class}(h))$ induced from the tree-bank are not exploited by most of these parsers.

Second, our method results in an *automatic* linguistic mark-up of tree-bank grammars. In contrast, manual linguistic mark-up of the tree-bank like in Klein and Manning (2003) is based on individual linguistic intuition and might be cost and time intensive.

Third, our method can be thought of as a new lexicalization scheme of CFG based on the notion of latent head-information, or as a successful attempt to incorporate lexical classes into parsers, combined with a new word clustering method based on the context represented by tree structure. It thus complements and extends the approach of Chiang and Bikel (2002), who aim at discovering latent head *markers* in tree-banks to improve manually written head-percolation rules.

Finally, the method can also be viewed as an extension of *factorial HMMs* (Ghahramani and Jordan, 1995) to PCFGs: the node labels on trees are enriched with a latent variable and the latent variables are learned by EM. Matsuzaki et al. (2005) independently introduce a similar approach and present empirical results that rival ours. In contrast to us,

they do not use an *explicit linguistic grammar*, and they do not attempt to *constrain* the space of latent variables *by linguistic principles*. As a consequence, our best models are three orders of magnitude more space efficient than theirs (with about 30 000 000 parameters). Therefore, parsing with their models requires sophisticated smoothing and pruning, whereas parsing with ours does not. Moreover, we calculate the most probable latent-head-decorated parse and delete the latent heads in a post-processing step. This is comparable to what they call 'Viterbi complete tree' parsing. Under this regime, our parser is on a par with theirs (F_1=85.5). This suggests that both models have learned a comparable degree of information, which is surprising, because we learn latent heads only, whereas they aim at learning general features. Crucially, a final 1% improvement comes from selecting most-probable parses by bagging all complete parses with the same incomplete skeleton beforehand; Clearly, a solution to this NP-Complete problem (Sima'an, 2002) can/should be also incorporated into our parser.

6 Conclusion

We introduced a method for inducing a head-driven PCFG with latent-head statistics from a tree-bank. The automatically trained parser is time and space efficient and achieves a performance already better than early lexicalized ones. This result suggests that our grammar-induction method can be successfully applied across domains, languages, and tree-bank annotations.

Acknowledgment

This work was supported by the Netherlands Organization for Scientific Research, NWO project no. 612.000.312, 'Learning Stochastic Tree-Grammars from Tree-banks'. I also would like to thank Yoav Seginer and Jelle Zuidema and the anonymous reviewers. A special thanks goes to Khalil Sima'an.

References

Ezra Black and al. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proc. of DARPA-91*.

Joan Bresnan and Ronald M. Kaplan. 1982. Lexical functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*. MIT Press.

Glenn Carroll and Mats Rooth. 1998. Valence induction with a head-lexicalized PCFG. In *Proc. of EMNLP-3*.

Eugene Charniak. 1997. Parsing with a context-free grammar and word statistics. In *Proc. of AAAI-97*.

David Chiang and D. Bikel. 2002. Recovering latent information in treebanks. In *Proc. of COLING-02*.

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proc. of ACL-96*.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of ACL-97*.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the *EM* algorithm. *J. Royal Statist. Soc.*, 39(B).

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proc. of ACL-03*.

Zoubin Ghahramani and Michael Jordan. 1995. Factorial Hidden Markov Models. Technical report, MIT.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. of ACL-03*.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proc. of ACL-95*.

Mitch Marcus, Beatrice Santorini, and Mary Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2).

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proc. of ACL-05*.

Detlef Prescher. 2003. A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars. Presented at the *15th European Summer School in Logic, Language and Information (ESSLLI)*.

Detlef Prescher. 2005. Inducing Head-Driven PCFGs with Latent Heads: Refining a Tree-bank Grammar for Parsing. In *Proc. of the 16th European Conference on Machine Learning*.

Helmut Schmid. 2004. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proc. of COLING-04*.

Khalil Sima'an. 2002. Computational complexity of probabilistic disambiguation. *Grammars*, 5(2).

A Classifier-Based Parser with Linear Run-Time Complexity

Kenji Sagae and **Alon Lavie**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{sagae,alavie}@cs.cmu.edu

Abstract

We present a classifier-based parser that produces constituent trees in linear time. The parser uses a basic bottom-up shift-reduce algorithm, but employs a classifier to determine parser actions instead of a grammar. This can be seen as an extension of the deterministic dependency parser of Nivre and Scholz (2004) to full constituent parsing. We show that, with an appropriate feature set used in classification, a very simple one-path greedy parser can perform at the same level of accuracy as more complex parsers. We evaluate our parser on section 23 of the WSJ section of the Penn Treebank, and obtain precision and recall of 87.54% and 87.61%, respectively.

1 Introduction

Two classifier-based deterministic dependency parsers for English have been proposed recently (Nivre and Scholz, 2004; Yamada and Matsumoto, 2003). Although they use different parsing algorithms, and differ on whether or not dependencies are labeled, they share the idea of greedily pursuing a single path, following parsing decisions made by a classifier. Despite their greedy nature, these parsers achieve high accuracy in determining dependencies. Although state-of-the-art statistical parsers (Collins, 1997; Charniak, 2000) are more accurate, the simplicity and efficiency of deterministic parsers make them attractive in a number of situations requiring fast, light-weight parsing, or parsing of large amounts of data. However, dependency analyses lack important information contained in constituent structures. For example, the tree-path feature has been shown to be valuable in semantic role labeling (Gildea and Palmer, 2002).

We present a parser that shares much of the simplicity and efficiency of the deterministic dependency parsers, but produces both dependency and constituent structures simultaneously. Like the parser of Nivre and Scholz (2004), it uses the basic shift-reduce stack-based parsing algorithm, and runs in linear time. While it may seem that the larger search space of constituent trees (compared to the space of dependency trees) would make it unlikely that accurate parse trees could be built deterministically, we show that the precision and recall of constituents produced by our parser are close to those produced by statistical parsers with higher run-time complexity.

One desirable characteristic of our parser is its simplicity. Compared to other successful approaches to corpus-based constituent parsing, ours is remarkably simple to understand and implement. An additional feature of our approach is its modularity with regard to the algorithm and the classifier that determines the parser's actions. This makes it very simple for different classifiers and different sets of features to be used with the same parser with very minimal work. Finally, its linear run-time complexity allows our parser to be considerably faster than lexicalized PCFG-based parsers. On the other hand, a major drawback of the classifier-based parsing framework is that, depending on

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 125–132,
Vancouver, October 2005.

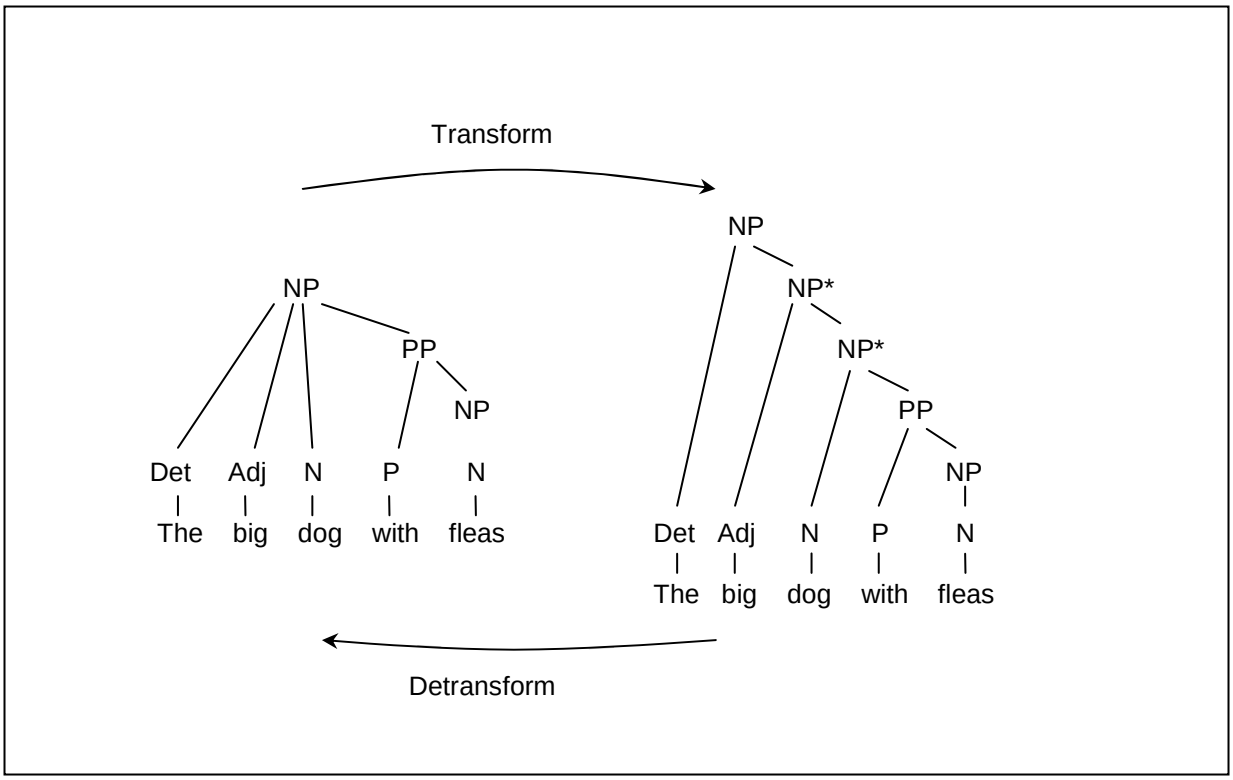


Figure 1: An example of the binarization transform/detransform. The original tree (left) has one node (NP) with four children. In the transformed tree, internal structure (marked by nodes with asterisks) was added to the subtree rooted by the node with more than two children. The word "dog" is the head of the original NP, and it is kept as the head of the transformed NP, as well as the head of each NP* node.

the classifier used, its training time can be much longer than that of other approaches.

Like other deterministic parsers (and unlike many statistical parsers), our parser considers the problem of syntactic analysis separately from part-of-speech (POS) tagging. Because the parser greedily builds trees bottom-up in one pass, considering only one path at any point in the analysis, the task of assigning POS tags to words is done before other syntactic analysis. In this work we focus only on the processing that occurs once POS tagging is completed. In the sections that follow, we assume that the input to the parser is a sentence with corresponding POS tags for each word.

2 Parser Description

Our parser employs a basic bottom-up shift-reduce parsing algorithm, requiring only a single pass over the input string. The algorithm considers only trees with unary and binary branching. In order to use trees with arbitrary branching for training, or generating them with the parser, we employ an instance of the transformation/detransformation process described in (Johnson, 1998). In our case, the transformation step involves simply converting each production with n children (where $n > 2$) into $n - 1$ binary productions. Trees must be lexicalized[1], so that the newly created internal structure of constituents with previous branching of more than two contains only subtrees with the same lexical head as the original constituent. Additional non-terminal symbols introduced in this process are clearly marked. The transformed (or "binarized") trees may then be used for training. Detransformation is applied to trees produced by the parser. This involves the removal of non-terminals intro-

[1] If needed, constituent head-finding rules such as those mentioned in Collins (1996) may be used.

duced in the transformation process, producing trees with arbitrary branching. An example of transformation/detransformation is shown in figure 1.

2.1 Algorithm Outline

The parsing algorithm involves two main data structures: a stack *S*, and a queue *W*. Items in *S* may be terminal nodes (POS-tagged words), or (lexicalized) subtrees of the final parse tree for the input string. Items in *W* are terminals (words tagged with parts-of-speech) corresponding to the input string. When parsing begins, *S* is empty and *W* is initialized by inserting every word from the input string in order, so that the first word is in front of the queue.

Only two general actions are allowed: shift and reduce. A shift action consists only of removing (shifting) the first item (POS-tagged word) from *W* (at which point the next word becomes the new first item), and placing it on top of *S*. Reduce actions are subdivided into unary and binary cases. In a unary reduction, the item on top of *S* is popped, and a new item is pushed onto *S*. The new item consists of a tree formed by a non-terminal node with the popped item as its single child. The lexical head of the new item is the same as the lexical head of the popped item. In a binary reduction, two items are popped from *S* in sequence, and a new item is pushed onto *S*. The new item consists of a tree formed by a non-terminal node with two children: the first item popped from *S* is the right child, and the second item is the left child. The lexical head of the new item is either the lexical head of its left child, or the lexical head of its right child.

If *S* is empty, only a shift action is allowed. If *W* is empty, only a reduce action is allowed. If both *S* and *W* are non-empty, either shift or reduce actions are possible. Parsing terminates when *W* is empty and *S* contains only one item, and the single item in *S* is the parse tree for the input string. Because the parse tree is lexicalized, we also have a dependency structure for the sentence. In fact, the binary reduce actions are very similar to the reduce actions in the dependency parser of Nivre and Scholz (2004), but they are executed in a different order, so constituents can be built. If *W* is empty, and more than one item remain in *S*, and no further reduce actions take place, the input string is rejected.

2.2 Determining Actions with a Classifier

A parser based on the algorithm described in the previous section faces two types of decisions to be made throughout the parsing process. The first type concerns whether to shift or reduce when both actions are possible, or whether to reduce or reject the input when only reduce actions are possible. The second type concerns what syntactic structures are created. Specifically, what new non-terminal is introduced in unary or binary reduce actions, or which of the left or right children are chosen as the source of the lexical head of the new subtree produced by binary reduce actions. Traditionally, these decisions are made with the use of a grammar, and the grammar may allow more than one valid action at any single point in the parsing process. When multiple choices are available, a grammar-driven parser may make a decision based on heuristics or statistical models, or pursue every possible action following a search strategy. In our case, both types of decisions are made by a classifier that chooses a unique action at every point, based on the local context of the parsing action, with no explicit grammar. This type of classifier-based parsing where only one path is pursued with no backtracking can be viewed as greedy or deterministic.

In order to determine what actions the parser should take given a particular parser configuration, a classifier is given a set of features derived from that configuration. This includes, crucially, the two topmost items in the stack *S*, and the item in front of the queue *W*. Additionally, a set of context features is derived from a (fixed) limited number of items below the two topmost items of *S*, and following the item in front of *W*. The specific features are shown in figure 2.

The classifier's target classes are parser actions that specify both types of decisions mentioned above. These classes are:

- SHIFT: a shift action is taken;
- REDUCE-UNARY-*XX*: a unary reduce action is taken, and the root of the new subtree pushed onto *S* is of type *XX* (where *XX* is a non-terminal symbol, typically NP, VP, PP, for example);
- REDUCE-LEFT-*XX*: a binary reduce action is taken, and the root of the new subtree pushed onto *S* is of non-terminal type *XX*.

Let:

S(n) denote the nth item from the top of the stack *S*, and
W(n) denote the nth item from the front of the queue *W*.

Features:

- The head-word (and its POS tag) of: S(0), S(1), S(2), and S(3)
- The head-word (and its POS tag) of: W(0), W(1), W(3) and W(3)
- The non-terminal node of the root of: S(0), and S(1)
- The non-terminal node of the left child of the root of: S(0), and S(1)
- The non-terminal node of the right child of the root of: S(0), and S(1)
- The non-terminal node of the left child of the root of: S(0), and S(1)
- The non-terminal node of the left child of the root of: S(0), and S(1)
- The linear distance (number of words apart) between the head-words of S(0) and S(1)
- The number of lexical items (words) that have been found (so far) to be dependents of the head-words of: S(0), and S(1)
- The most recently found lexical dependent of the head of the head-word of S(0) that is to the left of S(0)'s head
- The most recently found lexical dependent of the head of the head-word of S(0) that is to the right of S(0)'s head
- The most recently found lexical dependent of the head of the head-word of S(0) that is to the left of S(1)'s head
- The most recently found lexical dependent of the head of the head-word of S(0) that is to the right of S(1)'s head

Figure 2: Features used for classification. The features described in items 1 – 7 are more directly related to the lexicalized constituent trees that are built during parsing, while the features described in items 8 – 13 are more directly related to the dependency structures that are built simultaneously to the constituent structures.

Additionally, the head of the new subtree is the same as the head of the left child of the root node;

- REDUCE-RIGHT-*XX*: a binary reduce action is taken, and the root of the new subtree pushed onto *S* is of non-terminal type *XX*. Additionally, the head of the new subtree is the same as the head of the right child of the root node.

2.3 A Complete Classifier-Based Parser than Runs in Linear Time

When the algorithm described in section 2.1 is combined with a trained classifier that determines its parsing actions as described in section 2.2, we have a complete classifier-based parser. Training the parser is accomplished by training its classifier. To that end, we need training instances that consist of sets of features paired with their classes corre-

sponding to the correct parsing actions. These instances can be obtained by running the algorithm on a corpus of sentences for which the correct parse trees are known. Instead of using the classifier to determine the parser's actions, we simply determine the correct action by consulting the correct parse trees. We then record the features and corresponding actions for parsing all sentences in the corpus into their correct trees. This set of features and corresponding actions is then used to train a classifier, resulting in a complete parser.

When parsing a sentence with n words, the parser takes n shift actions (exactly one for each word in the sentence). Because the maximum branching factor of trees built by the parser is two, the total number of binary reduce actions is $n - 1$, if a complete parse is found. If the input string is rejected, the number of binary reduce actions is less than $n - 1$. Therefore, the number of shift and binary reduce actions is linear with the number of words in the input string. However, the parser as described so far has no limit on the number of unary reduce actions it may take. Although in practice a parser properly trained on trees reflecting natural language syntax would rarely make more than $2n$ unary reductions, pathological cases exist where an infinite number of unary reductions would be taken, and the algorithm would not terminate. Such cases may include the observation in the training data of sequences of unary productions that cycle through (repeated) non-terminals, such as *A->B->A->B*. During parsing, it is possible that such a cycle may be repeated infinitely.

This problem can be easily prevented by limiting the number of consecutive unary reductions that may be made to a finite number. This may be the number of non-terminal types seen in the training data, or the length of the longest chain of unary productions seen in the training data. In our experiments (described in section 3), we limited the number of consecutive unary reductions to three, although the parser never took more than two unary reduction actions consecutively in any sentence. When we limit the number of consecutive unary reductions to a finite number m, the parser makes at most $(2n - 1)m$ unary reductions when parsing a sentence of length n. Placing this limit not only guarantees that the algorithm terminates, but also guarantees that the number of actions taken by the parser is $O(n)$, where n is the length of the input string. Thus, the parser runs in linear time, assuming that classifying a parser action is done in constant time.

3 Similarities to Previous Work

As mentioned before, our parser shares similarities with the dependency parsers of Yamada and Matsumoto (2003) and Nivre and Scholz (2004) in that it uses a classifier to guide the parsing process in deterministic fashion. While Yamada and Matsumoto use a quadratic run-time algorithm with multiple passes over the input string, Nivre and Scholz use a simplified version of the algorithm described here, which handles only (labeled or unlabeled) dependency structures.

Additionally, our parser is in some ways similar to the maximum-entropy parser of Ratnaparkhi (1997). Ratnaparkhi's parser uses maximum-entropy models to determine the actions of a shift-reduce-like parser, but it is capable of pursuing several paths and returning the top-K highest scoring parses for a sentence. Its observed time is linear, but parsing is somewhat slow, with sentences of length 20 or more taking more than one second to parse, and sentences of length 40 or more taking more than three seconds. Our parser only pursues one path per sentence, but it is very fast and of comparable accuracy (see section 4). In addition, Ratnaparkhi's parser uses a more involved algorithm that allows it to work with arbitrary branching trees without the need of the binarization transform employed here. It breaks the usual reduce actions into smaller pieces (CHECK and BUILD), and uses two separate passes (not including the POS tagging pass) for determining chunks and higher syntactic structures separately.

Finally, there have been other deterministic shift-reduce parsers introduced recently, but their levels of accuracy have been well below the state-of-the-art. The parser in Kalt (2004) uses a similar algorithm to the one described here, but the classification task is framed differently. Using decision trees and fewer features, Kalt's parser has significantly faster training and parsing times, but its accuracy is much lower than that of our parser. Kalt's parser achieves precision and recall of about 77% and 76%, respectively (with automatically tagged text), compared to our parser's 86% (see section 4). The parser of Wong and Wu (1999) uses a separate NP-chunking step and, like Ratnaparkhi's parser, does not require a binary trans-

	Precision	Recall	Dependency	Time (min)
Charniak	89.5	89.6	92.1	28
Collins	88.3	88.1	91.5	45
Ratnaparkhi	87.5	86.3	Unk	Unk
Y&M	-	-	90.3	Unk
N&S	-	-	87.3	21
MBLpar	80.0	80.2	86.3	127
SVMpar	87.5	87.6	90.3	11

Table 1: Summary of results on labeled precision and recall of constituents, dependency accuracy, and time required to parse the test set. The parsers of Yamada and Matsumoto (Y&M) and Nivre and Scholz (N&S) do not produce constituent structures, only dependencies. "unk" indicates unknown values. Results for MBLpar and SVMpar using correct POS tags (if automatically produced POS tags are used, accuracy figures drop about 1.5% over all metrics).

form. It achieves about 81% precision and 82% recall with gold-standard tags (78% and 79% with automatically tagged text). Wong and Wu's parser is further differentiated from the other parsers mentioned here in that it does not use lexical items, working only from part-of-speech tags.

4 Experiments

We conducted experiments with the parser described in section 2 using two different classifiers: TinySVM (a support vector machine implementation by Taku Kudo)[2], and the memory-based learner TiMBL (Daelemans et al., 2004). We trained and tested the parser on the Wall Street Journal corpus of the Penn Treebank (Marcus et al., 1993) using the standard split: sections 2-21 were used for training, section 22 was used for development and tuning of parameters and features, and section 23 was used for testing. Every experiment reported here was performed on a Pentium IV 1.8GHz with 1GB of RAM.

Each tree in the training set had empty-node and function tag information removed, and the trees were lexicalized using similar head-table rules as those mentioned in (Collins, 1996). The trees were then converted into trees containing only unary and binary branching, using the binarization transform described in section 2. Classifier training instances of features paired with classes (parser actions) were extracted from the trees in the training set, as described in section 2.3. The total number of training instances was about 1.5 million.

The classifier in the SVM-based parser (denoted by SVMpar) uses the polynomial kernel with degree 2, following the work of Yamada and Matsumoto (2003) on SVM-based deterministic dependency parsing, and a one-against-all scheme for multi-class classification. Because of the large number of training instances, we used Yamada and Matsumoto's idea of splitting the training instances into several parts according to POS tags, and training classifiers on each part. This greatly reduced the time required to train the SVMs, but even with the splitting of the training set, total training time was about 62 hours. Training set splitting comes with the cost of reduction in accuracy of the parser, but training a single SVM would likely take more than one week. Yamada and Matsumoto experienced a reduction of slightly more than 1% in de-

[2] http://chasen.org/~taku/software/TinySVM

pendency accuracy due to training set splitting, and we expect that a similar loss is incurred here.

When given perfectly tagged text (gold tags extracted from the Penn Treebank), SVMpar has labeled constituent precision and recall of 87.54% and 87.61%, respectively, and dependency accuracy of 90.3% over all sentences in the test set. The total time required to parse the entire test set was 11 minutes. Out of more than 2,400 sentences, only 26 were rejected by the parser (about 1.1%). For these sentences, partial analyses were created by combining the items in the stack in flat structures, and these were included in the evaluation. Predictably, the labeled constituent precision and recall obtained with automatically POS-tagged sentences were lower, at 86.01% and 86.15%. The part-of-speech tagger used in our experiments was SVMTool (Giménez and Márquez, 2004), and its accuracy on the test set is 97%.

The MBL-based parser (denoted by MBLpar) uses the IB1 algorithm, with five nearest neighbors, and the modified value difference metric (MVDM), following the work of Nivre and Scholz (2004) on MBL-based deterministic dependency parsing. MBLpar was trained with all training instances in under 15 minutes, but its accuracy on the test set was much lower than that of SVMpar, with constituent precision and recall of 80.0% and 80.2%, and dependency accuracy of 86.3% (24 sentences were rejected). It was also much slower than SVMpar in parsing the test set, taking 127 minutes. In addition, the total memory required for running MBLpar (including the classifier) was close to 1 gigabyte (including the trained classifier), while SVMpar required only about 200 megabytes (including all the classifiers).

Table 1 shows a summary of the results of our experiments with SVMpar and MBLpar, and also results obtained with the Charniak (2000) parser, the Bikel (2003) implementation of the Collins (1997) parser, and the Ratnaparkhi (1997) parser. We also include the dependency accuracy from Yamada and Matsumoto's (2003) SVM-based dependency parser, and Nivre and Scholz's (2004) MBL-based dependency parser. These results show that the choice of classifier is extremely important in this task. SVMpar and MBLpar use the same algorithm and features, and differ only on the classifiers used to make parsing decisions. While in many natural language processing tasks different classifiers perform at similar levels of accuracy, we have observed a dramatic difference between using support vector machines and a memory-based learner. Although the reasons for such a large disparity in results is currently the subject of further investigation, we speculate that a relatively small difference in initial classifier accuracy results in larger differences in parser performance, due to the deterministic nature of the parser (certain errors may lead to further errors). We also believe classifier choice to be one major source of the difference in accuracy between Nivre and Scholz's parser and Yamada and Matsumoto's parser.

While the accuracy of SVMpar is below that of lexicalized PCFG-based statistical parsers, it is surprisingly good for a greedy parser that runs in linear time. Additionally, it is considerably faster than lexicalized PCFG-based parsers, and offers a good alternative for when fast parsing is needed. MBLpar, on the other hand, performed poorly in terms of accuracy and speed.

5 Conclusion and Future Work

We have presented a simple shift-reduce parser that uses a classifier to determine its parsing actions and runs in linear time. Using SVMs for classification, the parser has labeled constituent precision and recall higher than 87% when using the correct part-of-speech tags, and slightly higher than 86% when using automatically assigned part-of-speech tags. Although its accuracy is not as high as those of state-of-the-art statistical parsers, our classifier-based parser is considerably faster than several well-known parsers that employ search or dynamic programming approaches. At the same time, it is significantly more accurate than previously proposed deterministic parsers for constituent structures.

We have also shown that much of the success of a classifier-based parser depends on what classifier is used. While this may seem obvious, the differences observed here are much greater than what would be expected from looking, for example, at results from chunking/shallow parsing (Zhang et al., 2001; Kudo and Matsumoto, 2001; Veenstra and van den Bosch, 2000).

Future work includes the investigation of the effects of individual features, the use of additional classification features, and the use of different classifiers. In particular, the use of tree features seems appealing. This may be accomplished with SVMs

using a tree kernel, or the tree boosting classifier BACT described in (Kudo and Matsumoto, 2004). Additionally, we plan to investigate the use of the beam strategy of Ratnaparkhi (1997) to pursue multiple parses while keeping the run-time linear.

References

Charniak, E. 2000. A maximum-entropy-inspired parser. *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics*. Seattle, WA.

Collins, M. 1997. Three generative, lexicalized models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics* (pp. 16-23). Madrid, Spain.

Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. 2004. TiMBL: Tilburg Memory Based Learner, version 5.1, reference guide. *ILK Research Group Technical Report Series* no. 04-02, 2004.

Gildea, D., and Palmer, M. 2002. The necessity of syntactic parsing for predicate argument recognition. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (pp. 239-246). Philadelphia, PA.

Kalt, T. 2004. Induction of greedy controllers for deterministic treebank parsers. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain.

Kudo, T., and Matsumoto, Y. 2004. A boosting algorithm for classification of semi-structured text. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain.

Kudo, T., and Matsumoto, Y. 2001. Chunking with support vector machines. *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*. Pittsburgh, PA.

Johnson, M. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613-632.

Marcus, M. P., Santorini, B., and Marcinkiewics, M. A. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19.

Nivre, J., and Scholz, M. 2004. Deterministic dependency parsing of English text. *Proceedings of the 20th International Conference on Computational Linguistics* (pp. 64-70). Geneva, Switzerland.

Ratnaparkhi, A. 1997. A linear observed time statistical parser based on maximum entropy models. *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*. Providence, Rhode Island.

Veenstra, J., van den Bosch, A. 2000. Single-classifier memory-based phrase chunking. *Proceedings of Fourth Workshop on Computational Natural Language Learning (CoNLL 2000).* Lisbon, Portugal.

Wong, A., and Wu. D. 1999. Learning a lightweight robust deterministic parser. *Proceedings of the Sixth European Conference on Speech Communication and Technology*. Budapest.

Yamada, H., and Matsumoto, Y. 2003. Statistical dependency analysis with support vector machines. *Proceedings of the Eighth International Workshop on Parsing Technologies.* Nancy, France.

Zhang, T., Damerau, F., and Johnson, D. 2002. Text chunking using regularized winnow. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.* Tolouse, France.

Chunk Parsing Revisited

Yoshimasa Tsuruoka[1,2] **and Jun'ichi Tsujii**[2,3,1]
[1] CREST, JST (Japan Science and Technology Corporation)
[2] Department of Computer Science, University of Tokyo
[3] School of Informatics, University of Manchester
{tsuruoka,tsujii}@is.s.u-tokyo.ac.jp

Abstract

Chunk parsing is conceptually appealing but its performance has not been satisfactory for practical use. In this paper we show that chunk parsing can perform significantly better than previously reported by using a simple sliding-window method and maximum entropy classifiers for phrase recognition in each level of chunking. Experimental results with the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs with very high speed (14 msec/sentence). We also present a parsing method for searching the best parse by considering the probabilities output by the maximum entropy classifiers, and show that the search method can further improve the parsing accuracy.

1 Introduction

Chunk parsing (Tjong Kim Sang, 2001; Brants, 1999) is a simple parsing strategy both in implementation and concept. The parser first performs chunking by identifying base phrases, and convert the identified phrases to non-terminal symbols. The parser again performs chunking on the updated sequence and convert the newly recognized phrases into non-terminal symbols. The parser repeats this procedure until there are no phrases to be chunked. After finishing these chunking processes, we can reconstruct the complete parse tree of the sentence from the chunking results.

Although the conceptual simplicity of chunk parsing is appealing, satisfactory performance for practical use has not yet been achieved with this parsing strategy. Sang achieved an f-score of 80.49 on the Penn Treebank by using the IOB tagging method for each level of chunking (Tjong Kim Sang, 2001). However, there is a very large gap between their performance and that of widely-used practical parsers (Charniak, 2000; Collins, 1999).

The performance of chunk parsing is heavily dependent on the performance of phrase recognition in each level of chunking. We show in this paper that the chunk parsing strategy is indeed appealing in that it can give considerably better performance than previously reported by using a different approach for phrase recognition and that it enables us to build a very fast parser that gives high-precision outputs.

This advantage could open up the possibility of using full parsers for large-scale information extraction from the Web corpus and real-time information extraction where the system needs to analyze the documents provided by the users on run-time.

This paper is organized as follows. Section 2 introduces the overall chunk parsing strategy employed in this work. Section 3 describes the sliding-window based method for identifying chunks. Two filtering methods to reduce the computational cost are presented in sections 4 and 5. Section 6 explains the maximum entropy classifier and the feature set. Section 7 describes methods for searching the best parse. Experimental results on the Penn Treebank corpus are given in Section 8. Section 10 offers some concluding remarks.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 133–140,
Vancouver, October 2005.

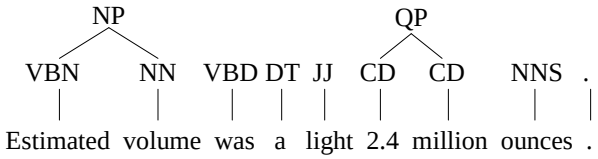


Figure 1: Chunk parsing, the 1st iteration.

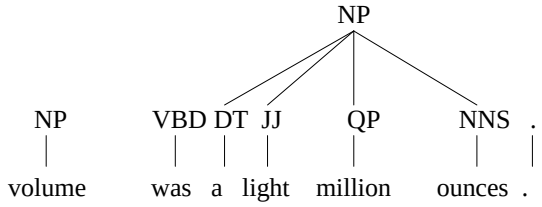


Figure 2: Chunk parsing, the 2nd iteration.

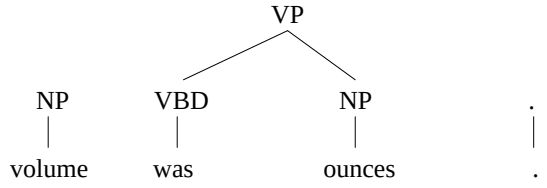


Figure 3: Chunk parsing, the 3rd iteration.

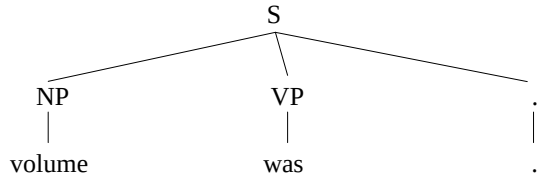


Figure 4: Chunk parsing, the 4th iteration.

2 Chunk Parsing

For the overall strategy of chunk parsing, we follow the method proposed by Sang (Tjong Kim Sang, 2001). Figures 1 to 4 show an example of chunk parsing. In the first iteration, the chunker identifies two base phrases, (NP Estimated volume) and (QP 2.4 million), and replaces each phrase with its non-terminal symbol and head. The head word is identified by using the head-percolation table (Magerman, 1995). In the second iteration, the chunker identifies (NP a light million ounces) and converts this phrase into NP. This chunking procedure is repeated until the whole sentence is chunked at the fourth iteration, and the full parse tree is easily recovered from the chunking history.

This parsing strategy converts the problem of full parsing into smaller and simpler problems, namely, chunking, where we only need to recognize flat structures (base phrases). Sang used the IOB tagging method proposed by Ramshow(Ramshaw and Marcus, 1995) and memory-based learning for each level of chunking and achieved an f-score of 80.49 on the Penn Treebank corpus.

3 Chunking with a sliding-window approach

The performance of chunk parsing heavily depends on the performance of each level of chunking. The popular approach to this shallow parsing is to convert the problem into a tagging task and use a variety of machine learning techniques that have been developed for sequence labeling problems such as Hidden Markov Models, sequential classification with SVMs (Kudo and Matsumoto, 2001), and Conditional Random Fields (Sha and Pereira, 2003).

One of our claims in this paper is that we should not convert the chunking problem into a tagging task. Instead, we use a classical sliding-window method for chunking, where we consider all subsequences as phrase candidates and classify them with a machine learning algorithm. Suppose, for example, we are about to perform chunking on the sequence in Figure 4.

NP-volume VBD-was .-.

We consider the following sub sequences as the phrase candidates in this level of chunking.

1. (NP-volume) VBD-was .-.
2. NP-volume (VBD-was) .-.
3. NP-volume VBD-was (.-.)
4. (NP-volume VBD-was) .-.
5. NP-volume (VBD-was .-.)
6. (NP-volume VBD-was .-.)

The merit of taking the sliding window approach is that we can make use of a richer set of features on recognizing a phrase than in the sequential labeling

approach. We can define arbitrary features on the target candidate (e.g. the whole sequence of non-terminal symbols of the target) and the surrounding context, which are, in general, not available in sequential labeling approaches.

We should mention here that there are some other modeling methods for sequence labeling which allow us to define arbitrary features on the target phrase. Semi-markov conditional random fields (Semi-CRFs) are one of such modeling methods (Sarawagi and Cohen, 2004). Semi-CRFs could give better performance than the sliding-window approach because they can incorporate features on other phrase candidates on the same level of chunking. However, they require additional computational resources for training and parsing, and the use of Semi-CRFs is left for future work.

The biggest disadvantage of the sliding window approach is the cost for training and parsing. Since there are $n(n+1)/2$ phrase candidates when the length of the sequence is n, a naive application of machine learning easily leads to prohibitive consumption of memory and time.

In order to reduce the number of phrase candidates to be considered by machine learning, we introduce two filtering phases into training and parsing. One is done by a rule dictionary. The other is done by a naive Bayes classifier.

4 Filtering with the CFG Rule Dictionary

We use an idea that is similar to the method proposed by Ratnaparkhi (Ratnaparkhi, 1996) for part-of-speech tagging. They used a *Tag Dictionary*, with which the tagger considers only the tag-word pairs that appear in the training sentences as the candidate tags.

A similar method can be used for reducing the number of phrase candidates. We first construct a rule dictionary consisting of all the CFG rules used in the training data. In both training and parsing, we filter out all the sub-sequences that do not match any of the entry in the dictionary.

4.1 Normalization

The rules used in the training data do not cover all the rules in unseen sentences. Therefore, if we take a naive filtering method using the rule dictionary, we substantially lose recall in parsing unseen data.

Original Symbol	Normalized Symbol
NNP, NNS, NNPS, PRP	NN
RBR, RBS	RB
JJR, JJS, PRP$	JJ
VBD, VBZ	VBP
:	,
", "	*NULL*

Table 1: Normalizing preterminals.

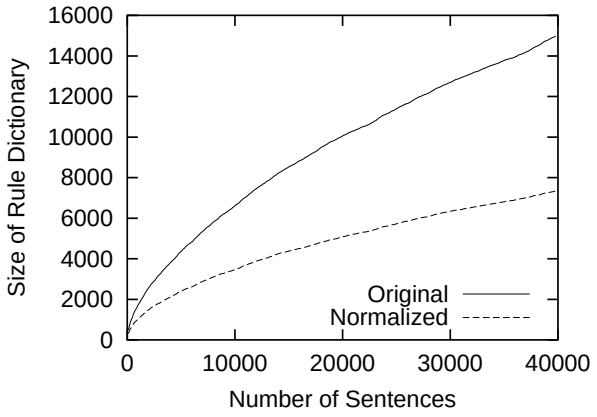


Figure 5: Number of sentences vs the size of the rule dictionary..

To alleviate the problem of the coverage of rules, we conduct normalization of the rules. We first convert preterminal symbols into equivalent sets using the conversion table provided in Table 1. This conversion reduces the sparseness of the rules.

We further normalize the Right-Hand-Side (RHS) of the rules with the following heuristics.

- "X CC X" is converted to "X".
- "X , X" is converted to "X".

Figure 5 shows the effectiveness of this normalization method. The figure illustrates how the number of rules increases in the rule dictionary as we add training sentences. Without the normalization, the number of rules continues to grow rapidly even when the entire training set is read. The normalization methods reduce the growing rate, which considerably alleviates the sparseness problem (i.e. the problems of unknown rules).

5 Filtering with the Naive Bayes classifier

Although the use of the rule dictionary significantly reduced the number of phrase candidates, we still found it difficult to train the parser using the entire training set when we used a rich set of features.

To further reduce the cost required for training and parsing, we propose to use a naive Bayes classifier for filtering the candidates. A naive Bayes classifier is simple and requires little storage and computational cost.

We construct a binary naive Bayes classifier for each phrase type using the entire training data. We considered the following information as the features.

- The Right-Hand-Side (RHS) of the CFG rule
- The left-adjacent nonterminal symbol.
- The right-adjacent nonterminal symbol.

By assuming the conditional independence among the features, we can compute the probability for filtering as follows:

$$\begin{aligned} P(y|c,l,r) &= \frac{P(c,l,r|y)P(y)}{P(c,l,r)} \\ &= \frac{P(c|y)P(l|y)P(r|y)P(y)}{\sum_y P(c|y)P(l|y)P(r|y)P(y)}, \end{aligned}$$

where y is a binary output indicating whether the candidate is a phrase of the target type or not, c is the RHS of the CFG rule, l is the symbol on the left, and r is the symbol on the right. We used the Laplace smoothing method for computing each probability. Note that the information about the result of the rule application, i.e., the LHS symbol, is considered in this filtering scheme because different naive Bayes classifiers are used for different LHS symbols (phrase types).

Table 2 shows the filtering performance in training with sections 02-21 on the Penn Treebank. We set the threshold probability for filtering to be 0.0001 for the experiments reported in this paper. The naive Bayes classifiers effectively reduced the number of candidates with little positive samples that were wrongly filtered out.

6 Phrase Recognition with a Maximum Entropy Classifier

For the candidates which are not filtered out in the above two phases, we perform classification with maximum entropy classifiers (Berger et al., 1996).

We construct a binary classifier for each type of phrases using the entire training set. The training samples for maximum entropy consist of the phrase candidates that have not been filtered out by the CFG rule dictionary and the naive Bayes classifier.

One of the merits of using a maximum entropy classifier is that we can obtain a probability from the classifier in each decision. The probability of each decision represents how likely the candidate is a correct chunk. We accept a chunk only when the probability is larger than the predefined threshold. With this thresholding scheme, we can control the trade-off between precision and recall by changing the threshold value.

Regularization is important in maximum entropy modeling to avoid overfitting to the training data. For this purpose, we use the maximum entropy modeling with inequality constraints (Kazama and Tsujii, 2003). This modeling has one parameter to tune as in Gaussian prior modeling. The parameter is called the *width factor*. We set this parameter to be 1.0 throughout the experiments. For numerical optimization, we used the Limited-Memory Variable-Metric (LMVM) algorithm (Benson and Moré, 2001).

6.1 Features

Table 3 lists the features used in phrase recognition with the maximum entropy classifier. Information about the adjacent non-terminal symbols is important. We use unigrams, bigrams, and trigrams of the adjacent symbols. Head information is also useful. We use unigrams and bigrams of the neighboring heads. The RHS of the CFG rule is also informative. We use the features on RHSs combined with symbol features.

7 Searching the best parse

7.1 Deterministic parsing

The deterministic version of chunk parsing is straight-forward. All we need to do is to repeat chunking until there are no phrases to be chunked.

Symbol	# candidates	# remaining candidates	# positives	# false negative
ADJP	4,043,409	1,052,983	14,389	53
ADVP	3,459,616	1,159,351	19,765	78
NP	7,122,168	3,935,563	313,042	117
PP	3,889,302	1,181,250	94,568	126
S	3,184,827	1,627,243	95,305	99
VP	4,903,020	2,013,229	145,878	144

Table 2: Effectiveness of the naive Bayes filtering on some representative nonterminals.

Symbol Unigrams	s_{i-1}, s_{j+1}
Symbol Bigrams	$s_i s_j$, $s_{i-2}s_{i-1}$, $s_{i-1}s_{j+1}$, $s_{j+1}s_{j+2}$
Symbol Trigrams	$s_{i-3}s_{i-2}s_{i-1}$, $s_{i-2}s_{i-1}s_{j+1}$, $s_{i-1}s_{j+1}s_{j+2}$, $s_{j+1}s_{j+2}s_{j+3}$
Head Unigrams	h_{i-1}, h_{j+1}
Head Bigrams	$h_{i-2}h_{i-1}$, $h_{i-1}s_{i+1}$, $h_{i+1}h_{i+2}$
Symbol-Head Unigrams	$s_i h_i$, $s_j h_j$, $s_k h_k (k \in i...j)$
CFG Rule	RHS
CFG Rule + Symbol Unigram	$s_{i-1}RHS$, $s_{j+1}RHS$
CFG Rule + Symbol Bigram	$s_{i-1}s_{j+1}RHS$

Table 3: Feature templates used in chunking. s_i and s_j represent the non-terminal symbols at the beginning and the ending of the target phrase respectively. h_i and h_j represent the head at the beginning and the ending of the target phrase respectively. RHS represents the Right-Hand-Side of the CFG rule.

If the maximum entropy classifiers give contradictory chunks in each level of chunking, we choose the chunk which has a larger probability than the other ones.

7.2 Parsing with search

We tried to perform searching in chunk parsing in order to investigate whether or not extra effort of searching gives a gain in parsing performance.

The problem is that because the modeling of our chunk parsing provides no explicit probabilistic distribution over the entire parse tree, there is no decisive way to properly evaluate the correctness of each parse. Nevertheless, we can consider the following score on each parse tree.

$$score = \prod_{i \in parse} P_i, \qquad (1)$$

where P_i is the probability of a phrase given by the maximum entropy classifier.

Because exploring all the possibilities of chunking requires prohibitive computational cost, we reduce the search space by focusing only on "uncertain" chunk candidates for the search. In each level of chunking, the chunker provides chunks with their probabilities. We consider only the chunks whose probabilities are within the predefined margin from 0.5. In other words, the chunks whose probabilities are larger than $(0.5 + margin)$ are considered as assured chunks, and thus are fixed when we generate alternative hypotheses of chunking. The chunks whose probabilities are smaller than $(0.5 - margin)$ are simply ignored.

We generate alternative hypotheses in each level of chunking, and search the best parse in a depth-first manner.

7.3 Iterative parsing

We also tried an *iterative parsing* strategy, which was successfully used in probabilistic HPSG parsing (Ninomiya et al., 2005). The parsing strategy is simple. The parser starts with a very low margin and tries to find a successful parse. If the parser cannot find a successful parse, then it increases the margin by a certain step and tries to parse with the wider margin.

8 Experiments

We ran parsing experiments using the Penn Treebank corpus, which is widely used for evaluating parsing algorithms. The training set consists of sections 02-21. We used section 22 as the development data, with which we tuned the feature set and parameters for parsing. The test set consists of section 23 and we report the performance of the parser on the set.

We used the *evalb* script provided by Sekine and Collins for evaluating the labeled recall/precision (LR/LP) of the parser outputs [1]. All the experiments were carried out on a server having a 2.6 GHz AMD Opteron CPU and 16GB memory.

8.1 Speed and Accuracy

First, we show the performance that achieved by deterministic parsing. Table 4 shows the results. We parsed all the sentences in section 23 using gold-standard part-of-speech (POS) tags. The trade-off between precision and recall can be controlled by changing the threshold for recognizing chunks. The fifth row gives the performance achieved with the default threshold (=0.5), where the precision is over 90% but the recall is low (75%). By lowering the threshold, we can improve the recall up to around 81% with 2% loss of precision. The best f-score is 85.06.

The parsing speed is very high. The parser takes only about 34 seconds to parse the entire section. Since this section contains 2,416 sentences, the average time required for parsing one sentence is 14 msec. The parsing speed slightly dropped when we used a lower threshold (0.1).

Table 5 shows the performance achieved when we used the search algorithm described in Section 7.2. We limited the maximum number of the nodes in the search space to 100 because further increase of the nodes had shown little improvement in parsing accuracy.

The search algorithm significantly boosted the precisions and recalls and achieved an f-score of 86.52 when the margin was 0.3. It should be noted that we obtain no gain when we use a tight margin. We need to consider phrases having low probabilities in order for the search to work.

[1] We used the parameter file "COLLINS.prm"

Threshold	LR	LP	F-score	Time (sec)
0.9	47.61	96.43	63.75	30.6
0.8	58.06	94.29	71.87	32.4
0.7	65.33	92.82	76.69	33.2
0.6	70.89	91.67	79.95	33.2
0.5	75.38	90.71	82.34	34.5
0.4	79.11	89.87	84.15	34.2
0.3	80.95	88.80	84.69	33.9
0.2	82.59	87.69	85.06	33.6
0.1	82.32	85.02	83.65	46.9

Table 4: Parsing performance on section 23 (all sentences, gold-standard POS tags) with the deterministic algorithm.

Margin	LR	LP	F-score	Time (sec)
0.0	75.65	90.81	82.54	41.2
0.1	79.63	90.16	84.57	74.4
0.2	82.70	89.57	86.00	94.8
0.3	84.60	88.53	86.52	110.2
0.4	84.91	86.99	85.94	116.3

Table 5: Parsing performance on section 23 (all sentences, gold-standard POS tags) with the search algorithm.

One of the advantages of our chunk parser is its parsing speed. For comparison, we show the trade-off between parsing time and performance in Collins parser (Collins, 1999) and our chunk parser in Figure 6. Collins parser allows the user to change the size of the beam in parsing. We used Model-2 because it gave better performance than Model-3 when the beam size was smaller than 1000. As for the chunk parser, we controlled the trade-off by changing the maximum number of nodes in the search. The uncertainty margin for chunk recognition was 0.3. Figure 6 shows that Collins parser clearly outperforms our chunk parser when the beam size is large. However, the performance significantly drops with a smaller beam size. The break-even point is at around 200 sec (83 msec/sentence).

8.2 Comparison with previous work

Table 6 summarizes our parsing performance on section 23 together with the results of previous studies. In order to make the results directly comparable, we produced POS tags as the input of our parsers by using a POS tagger (Tsuruoka and Tsujii, 2005) which was trained on sections 0-18 in the WSJ corpus.

The table also shows the performance achieved

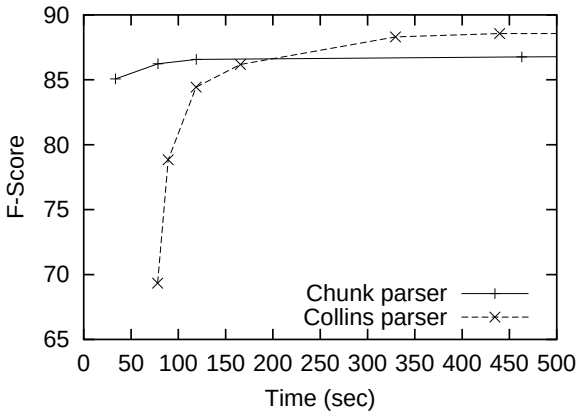


Figure 6: Time vs F-score on section 23. The x-axis represents the time required to parse the entire section. The time required for making a hash table in Collins parser is excluded.

	LR	LP	F-score
Ratnaparkhi (1997)	86.3	87.5	86.9
Collins (1999)	88.1	88.3	88.2
Charniak (2000)	89.6	89.5	89.5
Kudo (2005)	89.3	89.6	89.4
Sang (2001)	78.7	82.3	80.5
Deterministic (tagger-POSs)	81.2	86.5	83.8
Deterministic (gold-POSs)	82.6	87.7	85.1
Search (tagger-POSs)	83.2	87.1	85.1
Search (gold-POSs)	84.6	88.5	86.5
Iterative Search (tagger-POSs)	85.0	86.8	85.9
Iterative Search (gold-POSs)	86.2	88.0	87.1

Table 6: Comparison with other work. Parsing performance on section 23 (all sentences).

with the iterative parsing method presented in section 7.3. Our chunk parser achieved an f-score of 83.8 with the deterministic parsing methods using the POS-tagger tags. This f-score is better than that achieved by the previous study on chunk parsing by 3.3 points (Tjong Kim Sang, 2001). The search algorithms gave an additional 1.3 point improvement. Finally, the iterative parsing method achieved an f-score of 85.9.

Although our chunk parser showed considerably better performance than the previous study on chunk parsing, the performance is still significantly lower than those achieved by state-of-the-art parsers.

9 Discussion

There is a number of possible improvements in our chunk parser. We used a rule dictionary to reduce the cost required for training and parsing. However, the use of the rule dictionary makes the parser fail to identify a correct phrase if the phrase is not contained in the rule dictionary. Although we applied some normalization techniques in order to alleviate this problem, we have not completely solved the problem. Figure 5 indicates that still we will face unknown rules even when we have constructed the rule dictionary using the whole training data (note that the dotted line does not saturate).

Additional feature sets for the maximum entropy classifiers could improve the performance. The bottom-up parsing strategy allows us to use information about sub-trees that have already been constructed. We thus do not need to restrict ourselves to use only head-information of the partial parses. Since many researchers have reported that information on partial parse trees plays an important role for achieving high performance (Bod, 1992; Collins and Duffy, 2002; Kudo et al., 2005), we expect that additional features will improve the performance of chunk parsing.

Also, the methods for searching the best parse presented in sections 7.2 and 7.3 have much room for improvement. the search method does not have the device to avoid repetitive computations on the same nonterminal sequence in parsing. A chart-like structure which effectively stores the partial parse results could enable the parser to explore a broader search space and produce better parses.

Our chunk parser exhibited a considerable improvement in parsing accuracy over the previous study on chunk parsing. However, the reason is not completely clear. We believe that the sliding window approach, which enabled us to exploit a richer set of features than the so-called IOB approach, was the main contributer of the better performance. However, the combination of the IOB approach and a state-of-the-art machine learning algorithm such as support vector machines could produce a similar level of performance. In our preliminary experiments, the IOB tagging method with maximum entropy markov models has not yet achieved a comparable performance to the sliding window method.

10 Conclusion

In this paper we have shown that chunk parsing can perform significantly better than previously reported by using a simple sliding-window method and maximum entropy classifiers in each level of chunking. Experimental results on the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs with very high speed (14 msec/sentence). We also show that searching can improve the performance and the f-score reaches 85.9.

Although there is still a large gap between the accuracy of our chunk parser and the state-of-the-art, our parser can produce better f-scores than a widely-used parser when the parsing speed is really needed. This could open up the possibility of using full-parsing for large-scale information extraction.

References

Steven J. Benson and Jorge Moré. 2001. A limited-memory variable-metric algorithm for bound-constrained minimization. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory. ANL/MCS-P909-0901.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Rens Bod. 1992. Data oriented parsing. In *Proceedings of COLING 1992*.

Thorsten Brants. 1999. Cascaded markov models. In *Proceedings of EACL 1999*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL 2000*, pages 132–139.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL 2002*.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of EMNLP 2003*.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL 2001*.

Taku Kudo, Jun Suzuki, and Hideki Isozaki. 2005. Boosting-based parse reranking with subtree features. In *Proceedings of ACL 2005*.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL 1995*, pages 276–283.

Takashi Ninomiya, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic hpsg parsing. In *Proceedings of IWPT 2005*.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP 1996*, pages 133–142.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of EMNLP 1997*, pages 1–10.

Sunita Sarawagi and William W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *Proceedings of ICML 2004*.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*.

Erik Tjong Kim Sang. 2001. Transforming a chunker to a parser. In J. Veenstra W. Daelemans, K. Sima'an and J. Zavrel, editors, *Computational Linguistics in the Netherlands 2000*, pages 177–188. Rodopi.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of HLT/EMNLP 2005*.

Constituent Parsing by Classification

Joseph Turian and I. Dan Melamed
{lastname}@cs.nyu.edu
Computer Science Department
New York University
New York, New York 10003

Abstract

Ordinary classification techniques can drive a conceptually simple constituent parser that achieves near state-of-the-art accuracy on standard test sets. Here we present such a parser, which avoids some of the limitations of other discriminative parsers. In particular, it does not place any restrictions upon which types of features are allowed. We also present several innovations for faster training of discriminative parsers: we show how training can be parallelized, how examples can be generated prior to training without a working parser, and how independently trained sub-classifiers that have never done any parsing can be effectively combined into a working parser. Finally, we propose a new figure-of-merit for best-first parsing with confidence-rated inferences. Our implementation is freely available at: http://cs.nyu.edu/~turian/software/parser/

1 Introduction

Discriminative machine learning methods have improved accuracy on many NLP tasks, such as POS-tagging (Toutanova et al., 2003), machine translation (Och & Ney, 2002), and relation extraction (Zhao & Grishman, 2005). There are strong reasons to believe the same would be true of parsing. However, only limited advances have been made thus far, perhaps due to various limitations of extant discriminative parsers. In this paper, we present some innovations aimed at reducing or eliminating some of these limitations, specifically for the task of constituent parsing:

- We show how constituent parsing can be performed using standard classification techniques.
- Classifiers for different non-terminal labels can be induced independently and hence training can be parallelized.
- The parser can use arbitrary information to evaluate candidate constituency inferences.
- Arbitrary confidence scores can be aggregated in a principled manner, which allows beam search.

In Section 2 we describe our approach to parsing. In Section 3 we present experimental results.

The following terms will help to explain our work. A *span* is a range over contiguous words in the input sentence. Spans *cross* if they overlap but neither contains the other. An *item* (or *constituent*) is a (span, label) pair. A *state* is a set of parse items, none of which may cross. A parse *inference* is a pair (S, i), given by the current state S and an item i to be added to it. A parse *path* (or *history*) is a sequence of parse inferences over some input sentence (Klein & Manning, 2001). An *item ordering* (*ordering*, for short) constrains the order in which items may be inferred. In particular, if we prescribe a complete item ordering, the parser is *deterministic* (Marcus, 1980) and each state corresponds to a unique parse path. For some input sentence and gold-standard parse, a state is *correct* if the parser can infer zero or more additional items to obtain the gold-standard parse. A parse path is correct if it leads to a correct state. An

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 141–151,
Vancouver, October 2005.

inference is correct if adding its item to its state is correct.

2 Parsing by Classification

Recall that with typical probabilistic parsers, our goal is to output the parse $\hat{P}$ with the highest likelihood for the given input sentence x:

$$\hat{P} = \arg\max_{P \in \mathbf{P}(x)} Pr(P) \quad (1)$$

$$= \arg\max_{P \in \mathbf{P}(x)} \prod_{I \in P} Pr(I) \quad (2)$$

or, equivalently,

$$= \arg\max_{P \in \mathbf{P}(x)} \sum_{I \in P} \log(Pr(I)) \quad (3)$$

where each I is a constituency inference in the parse path P.

In this work, we explore a generalization in which each inference I is assigned a real-valued *confidence* score $Q(I)$ and individual confidences are aggregated using some function $\mathcal{A}$, which need not be a sum or product:

$$\hat{P} = \arg\max_{P \in \mathbf{P}(x)} \mathcal{A}_{I \in P} Q(I) \quad (4)$$

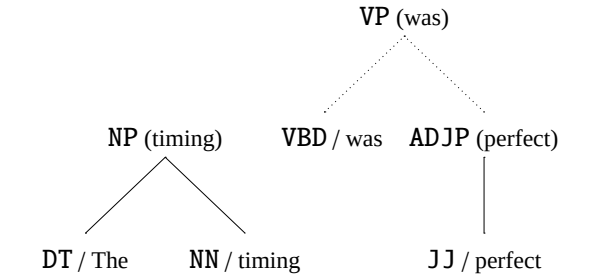
In Section 2.1 we describe how we induce scoring function $Q(I)$. In Section 2.2 we discuss the aggregation function $\mathcal{A}$. In Section 2.3 we describe the method used to restrict the size of the search space over $\mathbf{P}(x)$.

2.1 Learning the Scoring Function $Q(I)$

During training, our goal is to induce the scoring function Q, which assigns a real-valued confidence score $Q(I)$ to each candidate inference I (Equation 4). We treat this as a classification task: If inference I is correct, we would like $Q(I)$ to be a positive value, and if inference I is incorrect, we would like $Q(I)$ to be a negative value.

Training discriminative parsers can be computationally very expensive. Instead of having a single classifier score every inference, we parallelize training by inducing 26 sub-classifiers, one for each constituent label λ in the Penn Treebank (Taylor, Marcus, & Santorini, 2003): $Q(I_\lambda) = Q_\lambda(I_\lambda)$, where Q_λ is the λ-classifier and I_λ is an inference that infers a constituent with label λ. For example, the VP-classifier $Q_{\texttt{VP}}$ would score the VP-inference in Figure 1, preferably assigning it a positive confidence.

Figure 1 A candidate VP-inference, with head-children annotated using the rules given in (Collins, 1999).



Each λ-classifier is independently trained on training set E_λ, where each example $e_\lambda \in E_\lambda$ is a tuple (I_λ, y), I_λ is a candidate λ-inference, and $y \in \{\pm 1\}$. $y = +1$ if I_λ is a correct inference and -1 otherwise. This approach differs from that of Yamada and Matsumoto (2003) and Sagae and Lavie (2005), who parallelize according to the POS tag of one of the child items.

2.1.1 Generating Training Examples

Our method of generating training examples does not require a working parser, and can be run prior to any training. It is similar to the method used in the literature by deterministic parsers (Yamada & Matsumoto, 2003; Sagae & Lavie, 2005) with one exception: Depending upon the order constituents are inferred, there may be multiple bottom-up paths that lead to the same final parse, so to generate training examples we choose a *single* random path that leads to the gold-standard parse tree.[1] The training examples correspond to all candidate inferences considered in every state along this path, nearly all of which are incorrect inferences (with $y = -1$). For instance, only 4.4% of candidate NP-inferences are correct.

2.1.2 Training Algorithm

During training, for each label λ we induce scoring function Q_λ to minimize the *loss* over training examples E_λ:

$$Q_\lambda = \arg\min_{Q'_\lambda} \sum_{(I_\lambda, y) \in E_\lambda} L(y \cdot Q'_\lambda(I_\lambda)) \quad (5)$$

[1] The particular training tree paths used in our experiments are included in the aforementioned implementation so that our results can be replicated under the same experimental conditions.

where $y \cdot Q_\lambda(I_\lambda)$ is the margin of example (I_λ, y). Hence, the learning task is to maximize the margins of the training examples, i.e. induce scoring function Q_λ such that it classifies correct inferences with positive confidence and incorrect inferences with negative confidence. In our work, we minimized the logistic loss:

$$L(z) = \log(1 + \exp(-z)) \tag{6}$$

i.e. the negative log-likelihood of the training sample.

Our classifiers are ensembles of decisions trees, which we boost (Schapire & Singer, 1999) to minimize the above loss using the update equations given in Collins, Schapire, and Singer (2002). More specifically, classifier Q_λ^T is an ensemble comprising decision trees $q_\lambda^1, \ldots, q_\lambda^T$, where:

$$Q_\lambda^T(I_\lambda) = \sum_{t=1}^{T} q_\lambda^t(I_\lambda) \tag{7}$$

At iteration t, decision tree q_λ^t is grown, its leaves are confidence-rated, and it is added to the ensemble. The classifier for each constituent label is trained independently, so we henceforth omit λ subscripts.

An example (I, y) is assigned weight $w^t(I, y)$:[2]

$$w^t(I, y) = \frac{1}{1 + \exp(y \cdot Q^{t-1}(I))} \tag{8}$$

The total weight of y-value examples that fall in leaf f is $W_{f,y}^t$:

$$W_{f,y}^t = \sum_{\substack{(I,y') \in E \\ y'=y,\ I \in f}} w^t(I, y) \tag{9}$$

and this leaf has loss Z_f^t:

$$Z_f^t = 2 \cdot \sqrt{W_{f,+}^t \cdot W_{f,-}^t} \tag{10}$$

Growing the decision tree: The loss of the entire decision tree q^t is

$$Z(q^t) = \sum_{\text{leaf } f \in q^t} Z_f^t \tag{11}$$

We will use Z^t as a shorthand for $Z(q^t)$. When growing the decision tree, we greedily choose node splits to minimize this Z (Kearns & Mansour, 1999). In particular, the loss reduction of splitting leaf f using feature ϕ into two children, $f \wedge \phi$ and $f \wedge \neg\phi$, is $\Delta Z_f^t(\phi)$:

$$\Delta Z_f^t(\phi) = Z_f^t - (Z_{f \wedge \phi}^t + Z_{f \wedge \neg\phi}^t) \tag{12}$$

To split node f, we choose the $\hat{\phi}$ that reduces loss the most:

$$\hat{\phi} = \arg\max_{\phi \in \Phi} \Delta Z_f^t(\phi) \tag{13}$$

Confidence-rating the leaves: Each leaf f is confidence-rated as κ_f^t:

$$\kappa_f^t = \frac{1}{2} \cdot \log \frac{W_{f,+}^t + \epsilon}{W_{f,-}^t + \epsilon} \tag{14}$$

Equation 14 is smoothed by the ϵ term (Schapire & Singer, 1999) to prevent numerical instability in the case that either $W_{f,+}^t$ or $W_{f,-}^t$ is 0. In our experiments, we used $\epsilon = 10^{-8}$. Although our example weights are unnormalized, so far we've found no benefit from scaling ϵ as Collins and Koo (2005) suggest. All inferences that fall in a particular leaf node are assigned the same confidence: if inference I falls in leaf node f in the tth decision tree, then $q^t(I) = \kappa_f^t$.

2.1.3 Calibrating the Sub-Classifiers

An important concern is when to stop growing the decision tree. We propose the *minimum reduction in loss* (MRL) stopping criterion: During training, there is a value Θ^t at iteration t which serves as a threshold on the minimum reduction in loss for leaf splits. If there is no splitting feature for leaf f that reduces loss by at least Θ^t then f is not split. Formally, leaf f will not be bisected during iteration t if $\max_{\phi \in \Phi} \Delta Z_f^t(\phi) < \Theta^t$. The MRL stopping criterion is essentially ℓ_0 regularization: Θ^t corresponds to the ℓ_0 penalty parameter and each feature with non-zero confidence incurs a penalty of Θ^t, so to outweigh the penalty each split must reduce loss by at least Θ^t.

Θ^t decreases monotonically during training at the slowest rate possible that still allows training to proceed. We start by initializing Θ^1 to ∞, and at the beginning of iteration t we decrease Θ^t only if the root node $\emptyset$ of the decision tree cannot be split. Otherwise, Θ^t is set to Θ^{t-1}. Formally,

[2] If we were to replace this equation with $w^t(I, y) = \exp(y \cdot Q^{t-1}(I))^{-1}$, but leave the remainder of the algorithm unchanged, this algorithm would be confi dence-rated AdaBoost (Schapire & Singer, 1999), minimizing the exponential loss $L(z) = \exp(-z)$. In preliminary experiments, however, we found that the logistic loss provided superior generalization accuracy.

$\Theta^t = \min(\Theta^{t-1}, \max_{\phi \in \Phi} \Delta Z^t_\emptyset(\phi))$. In this manner, the decision trees are induced in order of decreasing Θ^t.

During training, the constituent classifiers Q_λ never do any parsing *per se*, and they train at different rates: If $\lambda \neq \lambda'$, then Θ^t_λ isn't necessarily equal to $\Theta^t_{\lambda'}$. We *calibrate* the different classifiers by picking some meta-parameter $\hat{\Theta}$ and insisting that the sub-classifiers comprised by a particular parser have all reached some fixed Θ in training. Given $\hat{\Theta}$, the constituent classifier for label λ is Q^t_λ, where $\Theta^t_\lambda \geq \hat{\Theta} > \Theta^{t+1}_\lambda$. To obtain the final parser, we cross-validate $\hat{\Theta}$, picking the value whose set of constituent classifiers maximizes accuracy on a development set.

2.1.4 Types of Features used by the Scoring Function

Our parser operates bottom-up. Let the *frontier* of a state be the top-most items (i.e. the items with no parents). The *children* of a candidate inference are those frontier items below the item to be inferred, the left *context* items are those frontier items to the left of the children, and the right context items are those frontier items to the right of the children. For example, in the candidate `VP`-inference shown in Figure 1, the frontier comprises the `NP`, `VBD`, and `ADJP` items, the `VBD` and `ADJP` items are the children of the `VP`-inference (the `VBD` is its *head* child), the `NP` is the left context item, and there are no right context items.

The design of some parsers in the literature restricts the kinds of features that can be usefully and efficiently evaluated. Our scoring function and parsing algorithm have no such limitations. Q can, in principle, use arbitrary information from the history to evaluate constituent inferences. Although some of our feature types are based on prior work (Collins, 1999; Klein & Manning, 2003; Bikel, 2004), we note that our scoring function uses more history information than typical parsers.

All features check whether an item has some property; specifically, whether the item's label/headtag/headword is a certain value. These features perform binary tests on the state directly, unlike Henderson (2003) which works with an intermediate representation of the history. In our baseline setup, feature set Φ contained five different feature types, described in Table 1.

Table 2 Feature item groups.

- all children
- all non-head children
- all non-leftmost children
- all non-rightmost children
- all children left of the head
- all children right of the head
- head-child and all children left of the head
- head-child and all children right of the head

2.2 Aggregating Confidences

To get the cumulative score of a parse path P, we apply aggregator $\mathcal{A}$ over the confidences $Q(I)$ in Equation 4. Initially, we defined $\mathcal{A}$ in the customary fashion as summing the loss of each inference's confidence:

$$\hat{P} = \arg\max_{P \in \mathbf{P}(x)} \left(- \sum_{I \in P} L\left(Q(I)\right) \right) \tag{15}$$

with the logistic loss L as defined in Equation 6. (We negate the final sum because we want to minimize the loss.) This definition of $\mathcal{A}$ is motivated by viewing L as a negative log-likelihood given by a logistic function (Collins et al., 2002), and then using Equation 3. It is also inspired by the multiclass loss-based decoding method of Schapire and Singer (1999). With this additive aggregator, loss monotonically increases as inferences are added, as in a PCFG-based parser in which all productions decrease the cumulative probability of the parse tree.

In preliminary experiments, this aggregator gave disappointing results: precision increased slightly, but recall dropped sharply. Exploratory data analysis revealed that, because each inference incurs some positive loss, the aggregator very cautiously builds the smallest trees possible, thus harming recall. We had more success by defining $\mathcal{A}$ to maximize the minimum confidence. Essentially,

$$\hat{P} = \arg\max_{P \in \mathbf{P}(x)} \min_{I \in P} Q(I) \tag{16}$$

Ties are broken according to the second lowest confidence, then the third lowest, and so on.

2.3 Search

Given input sentence x, we choose the parse path P in $\mathbf{P}(x)$ with the maximum aggregated score (Equation 4). Since it is computationally intractable to

Table 1 Types of features.

- **Child item features** test if a particular child item has some property. E.g. does the item one right of the head have headword "perfect"? (True in Figure 1)
- **Context item features** test if a particular context item has some property. E.g. does the first item of left context have headtag `NN`? (True)
- **Grandchild item features** test if a particular grandchild item has some property. E.g. does the leftmost child of the rightmost child item have label `JJ`? (True)
- **Exists features** test if a particular group of items contains an item with some property. E.g. does some non-head child item have label `ADJP`? (True) Exists features select one of the groups of items specified in Table 2. Alternately, they can select the *terminals* dominated by that group. E.g. is there some terminal item dominated by non-rightmost children items that has headword "quux"? (False)

consider every possible sequence of inferences, we use beam search to restrict the size of $\mathbf{P}(x)$. As an additional guard against excessive computation, search stopped if more than a fixed maximum number of states were popped from the agenda. As usual, search also ended if the highest-priority state in the agenda could not have a better aggregated score than the best final parse found thus far.

3 Experiments

Following Taskar, Klein, Collins, Koller, and Manning (2004), we trained and tested on ≤ 15 word sentences in the English Penn Treebank (Taylor et al., 2003), 10% of the entire treebank by word count.[3] We used sections 02–21 (9753 sentences) for training, section 24 (321 sentences) for development, and section 23 (603 sentences) for testing, preprocessed as per Table 3. We evaluated our parser using the standard PARSEVAL measures (Black et al., 1991): labelled precision, recall, and F-measure (LPRC, LRCL, and LFMS, respectively), which are computed based on the number of constituents in the parser's output that match those in the gold-standard parse. We tested whether the observed differences in PARSEVAL measures are significant at $p = 0.05$ using a stratified shuffling test (Cohen, 1995, Section 5.3.2) with one million trials.[4]

As mentioned in Section 1, the parser cannot infer any item that crosses an item already in the state. We placed three additional candidacy restrictions on inferences: (a) Items must be inferred under the bottom-up item ordering; (b) To ensure the parser does not enter an infinite loop, no two items in a state can have both the same span and the same label; (c) An item can have no more than $K = 5$ children. (Only 0.24% of non-terminals in the preprocessed development set have more than five children.) The number of candidate inferences at each state, as well as the number of training examples generated by the algorithm in Section 2.1.1, is proportional to K. In our experiment, there were roughly $|E_\lambda| \approx 1.7$ million training examples for each classifier.

3.1 Baseline

In the baseline setting, context item features (Section 2.1.4) could refer to the two nearest items of context in each direction. The parser used a beam width of 1000, and was terminated in the rare event that more than 10,000 states were popped from the agenda. Figure 2 shows the accuracy of the baseline on the development set as training progresses. Cross-validating the choice of $\hat{\Theta}$ against the LFMS (Section 2.1.3) suggested an optimum of $\hat{\Theta} = 1.42$. At this $\hat{\Theta}$, there were a total of 9297 decision tree splits in the parser (summed over all constituent classifiers), LFMS = 87.16, LRCL = 86.32, and LPRC = 88.02.

3.2 Beam Width

To determine the effect of the beam width on the accuracy, we evaluated the baseline on the development set using a beam width of 1, i.e. parsing entirely greedily (Wong & Wu, 1999; Kalt, 2004; Sagae & Lavie, 2005). Table 4 compares the base-

[3] There was insufficient time before deadline to train on all sentences.

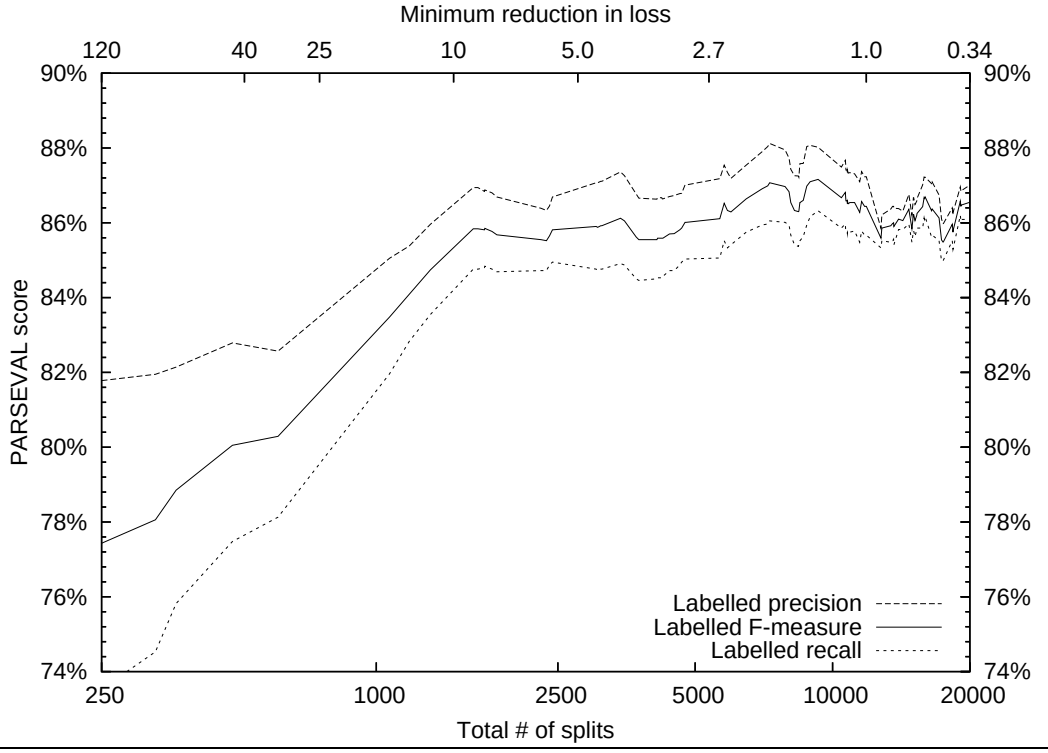
[4] The shuffling test we used was originally implemented by Dan Bikel (`http://www.cis.upenn.edu/~dbikel/software.html`) and subsequently modified to compute p-values for LFMS differences.

Table 3 Steps for preprocessing the data. Starred steps are performed only on input with tree structure.

1. * Strip functional tags and trace indices, and remove traces.
2. * Convert PRT to ADVP. (This convention was established by Magerman (1995).)
3. Remove quotation marks (i.e. terminal items tagged ‘‘ or ’’). (Bikel, 2004)
4. * Raise punctuation. (Bikel, 2004)
5. Remove outermost punctuation.[a]
6. * Remove unary projections to self (i.e. duplicate items with the same span and label).
7. POS tag the text using Ratnaparkhi (1996).
8. Lowercase headwords.
9. Replace any word observed fewer than 5 times in the (lower-cased) training sentences with UNK.

[a] As pointed out by an anonymous reviewer of Collins (2003), removing outermost punctuation may discard useful information. It's also worth noting that Collins and Roark (2004) saw a LFMS improvement of 0.8% over their baseline discriminative parser after adding punctuation features, one of which encoded the sentence-final punctuation.

Figure 2 PARSEVAL scores of the baseline on the ≤ 15 words development set of the Penn Treebank. The top *x*-axis shows accuracy as the minimum reduction in loss $\hat{\Theta}$ decreases. The bottom shows the corresponding number of decision tree splits in the parser, summed over all classifiers.



line results on the development set with a beam width of 1 and a beam width of 1000.[5] The wider beam seems to improve the PARSEVAL scores of the parser, although we were unable to detect a statistically significant improvement in LFMS on our relatively small development set.

[5] Using a beam width of 100,000 yielded output identical to using a beam width of 1000.

3.3 Context Size

Table 5 compares the baseline to parsers that could not examine as many context items. A significant portion of the baseline's accuracy is due to contextual clues, as evidenced by the poor accuracy of the no context run. However, we did not detect a significant difference between using one context item or two.

Table 4 PARSEVAL results on the ≤ 15 words development set of the baseline, varying the beam width. Also, the MRL that achieved this LFMS and the total number of decision tree splits at this MRL.

	Dev LFMS	Dev LRCL	Dev LPRC	MRL $\hat{\Theta}$	#splits total
Beam=1	86.36	86.20	86.53	2.03	7068
Baseline	87.16	86.32	88.02	1.42	9297

Table 5 PARSEVAL results on the ≤ 15 words development set, given the amount of context available. is statistically significant. The score differences between "context 0" and "context 1" are significant, whereas the differences between "context 1" and the baseline are not.

	Dev LFMS	Dev LRCL	Dev LPRC	MRL $\hat{\Theta}$	#splits total
Context 0	75.15	75.28	75.03	3.38	3815
Context 1	86.93	85.78	88.12	2.45	5588
Baseline	87.16	86.32	88.02	1.42	9297

Table 6 PARSEVAL results of decision stumps on the ≤ 15 words development set, through 8200 splits. The differences between the stumps run and the baseline are statistically significant.

	Dev LFMS	Dev LRCL	Dev LPRC	MRL $\hat{\Theta}$	#splits total
Stumps	85.72	84.65	86.82	2.39	5217
Baseline	87.07	86.05	88.12	1.92	7283

3.4 Decision Stumps

Our features are of relatively fine granularity. To test if a less powerful machine could provide accuracy comparable to the baseline, we trained a parser in which we boosted decisions *stumps*, i.e. decision trees of depth 1. Stumps are equivalent to learning a linear discriminant over the atomic features. Since the stumps run trained quite slowly, it only reached 8200 splits total. To ensure a fair comparison, in Table 6 we chose the best baseline parser with at most 8200 splits. The LFMS of the stumps run on the development set was 85.72%, significantly less accurate than the baseline.

For example, Figure 3 shows a case where NP classification better served by the informative conjunction $\phi_1 \wedge \phi_2$ found by the decision trees. Given the sentence "The man left", at the initial state there are six candidate NP-inferences, one for each span, and "(NP The man)" is the only candidate inference that is correct. ϕ_1 is true for the correct inference and two of the incorrect inferences ("(NP The)" and "(NP The man left)"). $\phi_1 \wedge \phi_2$, on the other hand, is true only for the correct inference, and so it is better at discriminating NPs over this sample.

Figure 3 An example of a decision (a) stump and (b) tree for scoring NP-inferences. Each leaf's value is the confidence assigned to all inferences that fall in this leaf. ϕ_1 asks "does the first child have a determiner headtag?". ϕ_2 asks "does the last child have a noun label?". NP classification is better served by the informative conjunction $\phi_1 \wedge \phi_2$ found by the decision trees.

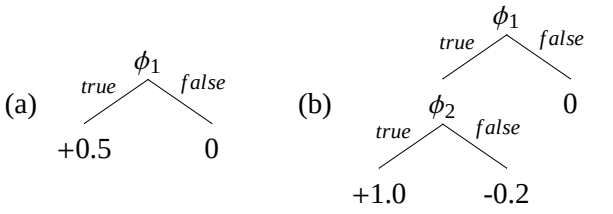


Table 7 PARSEVAL results of deterministic parsers on the ≤ 15 words development set through 8700 splits. A shaded cell means that the difference between this value and that of the baseline is statistically significant. All differences between l2r and r2l are significant.

	Dev LFMS	Dev LRCL	Dev LPRC	MRL $\hat{\Theta}$	#splits total
l2r	83.61	82.71	84.54	3.37	2157
r2l	85.76	85.37	86.15	3.39	1881
Baseline	87.07	86.05	88.12	1.92	7283

3.5 Deterministic Parsing

Our baseline parser simulates a non-deterministic machine, as at any state there may be several correct decisions. We trained deterministic variations of the parser, for which we imposed strict left-to-right (l2r) and right-to-left (r2l) item orderings. For these variations we generated training examples using the corresponding unique path to each gold-standard training tree. The r2l run reached only 8700 splits total, so in Table 7 we chose the best baseline and l2r

Table 8 PARSEVAL results of the full vocabulary parser on the ≤ 15 words development set. The differences between the full vocabulary run and the baseline are not statistically significant.

	Dev LFMS	Dev LRCL	Dev LPRC	MRL Θ	#splits total
Baseline	87.16	86.32	88.02	1.42	9297
Full vocab	87.50	86.85	88.15	1.27	10711

parser with at most 8700 splits.

r2l parsing is significantly more accurate than l2r. The reason is that the deterministic runs (l2r and r2l) must avoid prematurely inferring items that come later in the item ordering. This puts the l2r parser in a tough spot. If it makes far-right decisions, it's more likely to prevent correct subsequent decisions that are earlier in the l2r ordering, i.e. to the left. But if it makes far-left decisions, then it goes against the right-branching tendency of English sentences. In contrast, the r2l parser is more likely to be correct when it infers far-right constituents.

We also observed that the accuracy of the deterministic parsers dropped sharply as training progressed (See Figure 4). This behavior was unexpected, as the accuracy curve levelled off in every other experiment. In fact, the accuracy of the deterministic parsers fell even when parsing the training data. To explain this behavior, we examined the margin distributions of the r2l NP-classifier (Figure 5). As training progressed, the NP-classifier was able to reduce loss by driving up the margins of the incorrect training examples, at the expense of incorrectly classifying a slightly increased number of correct training examples. However, this is detrimental to parsing accuracy. The more correct inferences with negative confidence, the less likely it is at some state that the highest confidence inference is correct. This effect is particularly pronounced in the deterministic setting, where there is only one correct inference per state.

3.6 Full Vocabulary

As in traditional parsers, the baseline was smoothed by replacing any word that occurs fewer than five times in the training data with the special token UNK (Table 3.9). Table 8 compares the baseline to a full vocabulary run, in which the vocabulary contained all words observed in the training data. As evidenced by the results therein, controlling for lexical sparsity did not significantly improve accuracy in our setting. In fact, the full vocabulary run is slightly more accurate than the baseline on the development set, although this difference was not statistically significant. This was a late-breaking result, and we used the full vocabulary condition as our final parser for parsing the test set.

3.7 Test Set Results

Table 9 shows the results of our best parser on the ≤ 15 words test set, as well as the accuracy reported for a recent discriminative parser (Taskar et al., 2004) and scores we obtained by training and testing the parsers of Charniak (2000) and Bikel (2004) on the same data. Bikel (2004) is a "clean room" reimplementation of the Collins parser (Collins, 1999) with comparable accuracy. Both Charniak (2000) and Bikel (2004) were trained using the gold-standard tags, as this produced higher accuracy on the development set than using Ratnaparkhi (1996)'s tags.

3.8 Exploratory Data Analysis

To gain a better understanding of the weaknesses of our parser, we examined a sample of 50 development sentences that the full vocabulary parser did not get entirely correct. Besides noise and cases of genuine ambiguity, the following list outlines all error types that occurred in more than five sentences, in roughly decreasing order of frequency. (Note that there is some overlap between these groups.)

- **ADVPs and ADJPs** A disproportionate amount of the parser's error was due to ADJPs and ADVPs. Out of the 12.5% total error of the parser on the development set, an absolute 1.0% was due to ADVPs, and 0.9% due to ADJPs. The parser had LFMS = 78.9%, LPRC = 82.5%, LRCL = 75.6% on ADVPs, and LFMS = 68.0%, LPRC = 71.2%, LRCL = 65.0% on ADJPs.
  These constructions can sometimes involve tricky attachment decisions. For example, in the fragment "to get fat in times of crisis", the parser's output was "(VP to (VP get (ADJP fat (PP in (NP (NP times) (PP of (NP crisis)))))))" instead of the correct construction "(VP to (VP get (ADJP fat) (PP in (NP (NP times) (PP of (NP crisis)))))))".

Figure 4 LFMS of the baseline and the deterministic runs on the ≤ 15 words development set of the Penn Treebank. The *x*-axis shows the LFMS as training progresses and the number of decision tree splits increases.

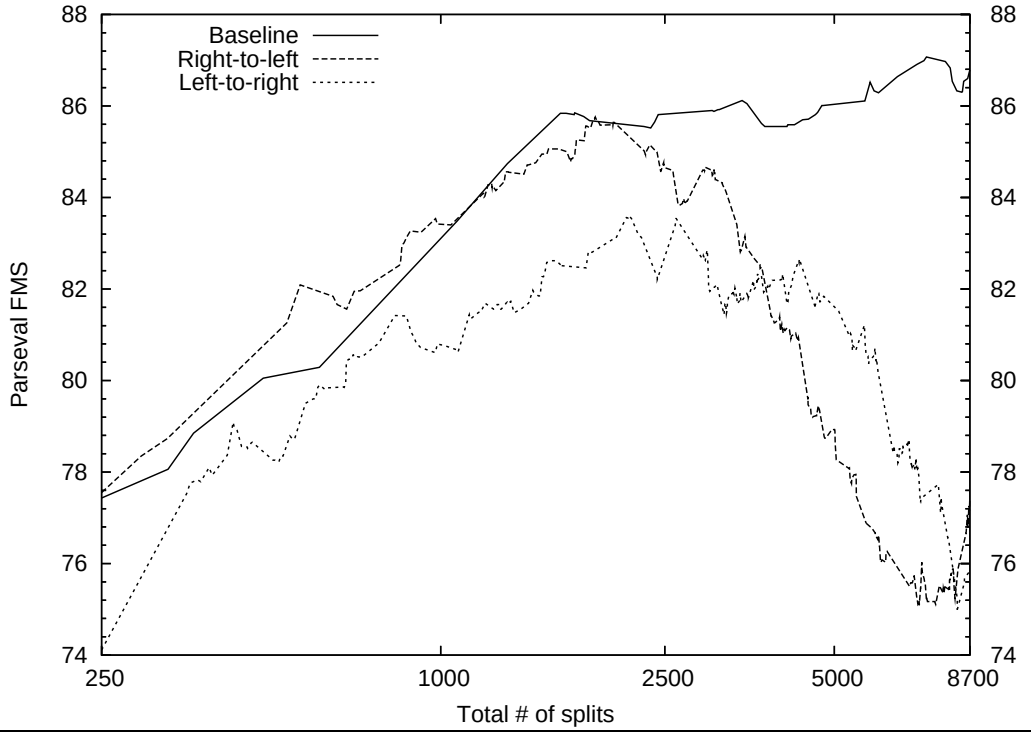
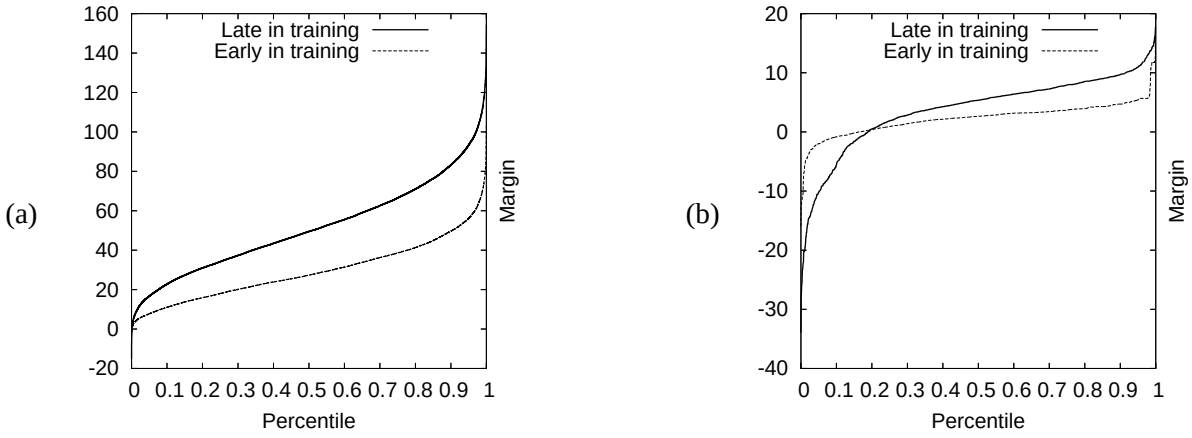


Figure 5 The margin distributions of the r2l NP-classifier, early in training and late in training, (a) over the incorrect training examples and (b) over the correct training examples.



The amount of noise present in ADJP and ADVP annotations in the PTB is unusually high. Annotation of ADJP and ADVP unary projections is particularly inconsistent. For example, the development set contains the sentence "The dollar was trading sharply lower in Tokyo .", with "sharply lower" bracketed as "(ADVP (ADVP sharply) lower)". "sharply lower" appears 16 times in the complete training section, every time bracketed as "(ADVP sharply lower)", and "sharply higher" 10 times, always as "(ADVP sharply higher)". Because of the high number of negative examples, the classifiers'

Table 9 PARSEVAL results of on the ≤ 15 words test set of various parsers in the literature. The differences between the full vocabulary run and Bikel or Charniak are significant. Taskar et al. (2004)'s output was unavailable for significance testing, but presumably its differences from the full vocab parser are also significant.

	Test LFMS	Test LRCL	Test LPRC	Dev LFMS	Dev LRCL	Dev LPRC
Full vocab	87.13	86.47	87.80	87.50	86.85	88.15
Bikel (2004)	88.85	88.31	89.39	86.82	86.43	87.22
Taskar et al. (2004)	89.12	89.10	89.14	89.98	90.22	89.74
Charniak (2000)	90.09	90.01	90.17	89.50	89.69	89.32

bias is to cope with the noise by favoring negative confidences predictions for ambiguous ADJP and ADVP decisions, hence their abysmal labelled recall. One potential solution is the weight-sharing strategy described in Section 3.5.

- **Tagging Errors** Many of the parser's errors were due to poor tagging. Preprocessing sentence "Would service be voluntary or compulsory ?" gives "would/MD service/VB be/VB voluntary/JJ or/CC UNK/JJ" and, as a result, the parser brackets "service ... compulsory" as a VP instead of correctly bracketing "service" as an NP. We also found that the tagger we used has difficulties with completely capitalized words, and tends to tag them NNP. By giving the parser access to the same features used by taggers, especially rich lexical features (Toutanova et al., 2003), the parser might learn to compensate for tagging errors.
- **Attachment decisions** The parser does not detect affinities between certain word pairs, so it has difficulties with bilexical dependency decisions. In principle, bilexical dependencies can be represented as conjunctions of feature given in Section 2.1.4. Given more training data, the parser might learn these affinities.

4 Conclusions

In this work, we presented a near state-of-the-art approach to constituency parsing which overcomes some of the limitations of other discriminative parsers. Like Yamada and Matsumoto (2003) and Sagae and Lavie (2005), our parser is driven by classifiers. Even though these classifiers themselves never do any parsing during training, they can be combined into an effective parser. We also presented a beam search method under the objective function of maximizing the minimum confidence.

To ensure efficiency, some discriminative parsers place stringent requirements on which types of features are permitted. Our approach requires no such restrictions and our scoring function can, in principle, use arbitrary information from the history to evaluate constituent inferences. Even though our features may be of too fine granularity to discriminate through linear combination, discriminatively trained decisions trees determine useful feature combinations automatically, so adding new features requires minimal human effort.

Training discriminative parsers is notoriously slow, especially if it requires generating examples by repeatedly parsing the treebank (Collins & Roark, 2004; Taskar et al., 2004). Although training time is still a concern in our setup, the situation is ameliorated by generating training examples in advance and inducing one-vs-all classifiers in parallel, a technique similar in spirit to the POS-tag parallelization in Yamada and Matsumoto (2003) and Sagae and Lavie (2005).

This parser serves as a proof-of-concept, in that we have not fully exploited the possibilities of engineering intricate features or trying more complex search methods. Its flexibility offers many opportunities for improvement, which we leave to future work.

Acknowledgments

The authors would like to thank Dan Bikel, Mike Collins, Ralph Grishman, Adam Meyers, Mehryar Mohri, Satoshi Sekine, and Wei Wang, as well as the anonymous reviewers, for their helpful comments

and constructive criticism. This research was sponsored by an NSF CAREER award, and by an equipment gift from Sun Microsystems.

References

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics*, *30*(4), 479–511.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., et al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language* (pp. 306–311).

Charniak, E. (2000). A maximum-entropy-inspired parser. In *NAACL* (pp. 132–139).

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. MIT Press.

Collins, M. (1999). *Head-driven statistical models for natural language parsing*. Unpublished doctoral dissertation, UPenn.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, *29*(4), 589–637.

Collins, M., & Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, *31*(1), 25–69.

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *ACL*.

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, *48*(1-3), 253–285.

Henderson, J. (2003). Inducing history representations for broad coverage statistical parsing. In *HLT/NAACL*.

Kalt, T. (2004). Induction of greedy controllers for deterministic treebank parsers. In *EMNLP* (pp. 17–24).

Kearns, M. J., & Mansour, Y. (1999). On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and Systems Sciences*, *58*(1), 109–128.

Klein, D., & Manning, C. D. (2001). Parsing and hypergraphs. In *IWPT* (pp. 123–134).

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *ACL* (pp. 423–430).

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *ACL* (pp. 276–283).

Marcus, M. P. (1980). *Theory of syntactic recognition for natural languages*. MIT Press.

Och, F. J., & Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *ACL*.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP* (pp. 133–142).

Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *IWPT*.

Schapire, R. E., & Singer, Y. (1999). Improved boosting using confidence-rated predictions. *Machine Learning*, *37*(3), 297–336.

Taskar, B., Klein, D., Collins, M., Koller, D., & Manning, C. (2004). Max-margin parsing. In *EMNLP* (pp. 1–8).

Taylor, A., Marcus, M., & Santorini, B. (2003). The Penn Treebank: an overview. In A. Abeillé (Ed.), *Treebanks: Building and using parsed corpora* (pp. 5–22).

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT/NAACL* (pp. 252–259).

Wong, A., & Wu, D. (1999). Learning a lightweight robust deterministic parser. In *EUROSPEECH*.

Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *IWPT*.

Zhao, S., & Grishman, R. (2005). Extracting relations with integrated information using kernel methods. In *ACL*.

Strictly Lexical Dependency Parsing

Qin Iris Wang and **Dale Schuurmans**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2E8
{wqin,dale}@cs.ualberta.ca

Dekang Lin
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California, USA, 94043
lindek@google.com

Abstract

We present a strictly lexical parsing model where all the parameters are based on the words. This model does not rely on part-of-speech tags or grammatical categories. It maximizes the conditional probability of the parse tree given the sentence. This is in contrast with most previous models that compute the joint probability of the parse tree and the sentence. Although the maximization of joint and conditional probabilities are theoretically equivalent, the conditional model allows us to use distributional word similarity to generalize the observed frequency counts in the training corpus. Our experiments with the Chinese Treebank show that the accuracy of the conditional model is 13.6% higher than the joint model and that the strictly lexicalized conditional model outperforms the corresponding unlexicalized model based on part-of-speech tags.

1 Introduction

There has been a great deal of progress in statistical parsing in the past decade (Collins, 1996; Collins, 1997; Chaniak, 2000). A common characteristic of these parsers is their use of lexicalized statistics. However, it was discovered recently that bi-lexical statistics (parameters that involve two words) actually played much smaller role than previously believed. It was found in (Gildea, 2001) that the removal of bi-lexical statistics from a state-of-the-art PCFG parser resulted very small change in the output. Bikel (2004) observed that the bi-lexical statistics accounted for only 1.49% of the bigram statistics used by the parser. When considering only bigram statistics involved in the highest probability parse, this percentage becomes 28.8%. However, even when the bi-lexical statistics do get used, they are remarkably similar to their back-off values using part-of-speech tags. Therefore, the utility of bi-lexical statistics becomes rather questionable. Klein and Manning (2003) presented an unlexicalized parser that eliminated all lexicalized parameters. Its performance was close to the state-of-the-art lexicalized parsers.

We present a statistical dependency parser that represents the other end of spectrum where all statistical parameters are lexical and the parser does not require part-of-speech tags or grammatical categories. We call this strictly lexicalized parsing.

A part-of-speech lexicon has always been considered to be a necessary component in any natural language parser. This is true in early rule-based as well as modern statistical parsers and in dependency parsers as well as constituency parsers. The need for part-of-speech tags arises from the sparseness of natural language data. They provide generalizations of words that are critical for parsers to deal with the sparseness. Words belonging to the same part-of-speech are expected to have the same syntactic behavior.

Instead of part-of-speech tags, we rely on distributional word similarities computed automatically from a large unannotated text corpus. One of the benefits of strictly lexicalized parsing is that

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 152–159,
Vancouver, October 2005.

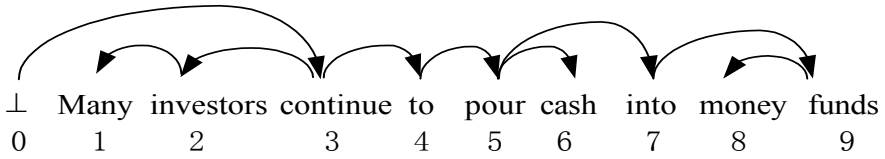


Figure 1. An Example Dependency Tree.

the parser can be trained with a treebank that only contains the dependency relationships between words. The annotators do not need to annotate parts-of-speech or non-terminal symbols (they don't even have to know about them), making the construction of the treebank easier.

Strictly lexicalized parsing is especially beneficial for languages such as Chinese, where parts-of-speech are not as clearly defined as English. In Chinese, clear indicators of a word's part-of-speech such as suffixes -ment, -ous or function words such as the, are largely absent. In fact, monolingual Chinese dictionaries that are mainly intended for native speakers almost never contain part-of-speech information.

In the next section, we present a method for modeling the probabilities of dependency trees. Section 3 applies similarity-based smoothing to the probability model to deal with data sparseness. We then present experimental results with the Chinese Treebank in Section 4 and discuss related work in Section 5.

2 A Probabilistic Dependency Model

Let S be a sentence. The dependency structure T of S is a directed tree connecting the words in S. Each link in the tree represents a dependency relationship between two words, known as the head and the modifier. The direction of the link is from the head to the modifier. We add an artificial root node ($\perp$) at the beginning of each sentence and a dependency link from $\perp$ to the head of the sentence so that the head of the sentence can be treated in the same way as other words. Figure 1 shows an example dependency tree.

We denote a dependency link l by a triple (u, v, d), where u and v are the indices ($u < v$) of the words connected by l, and d specifies the direction of the link l. The value of d is either L or R. If $d = L$, v is the index of the head word; otherwise, u is the index of the head word.

Dependency trees are typically assumed to be projective (without crossing arcs), which means that if there is an arc from h to m, h is an ancestor of all the words between h and m. Let $F(S)$ be the set of possible directed, projective trees spanning on S. The parsing problem is to find

$$\arg\max_{T \in F(S)} P(T \mid S)$$

Generative parsing models are usually defined recursively from top down, even though the decoders (parsers) for such models almost always take a bottom-up approach. The model proposed here is a bottom-up one. Like previous approaches, we decompose the generation of a parse tree into a sequence of steps and define the probability of each step. The probability of the tree is simply the product of the probabilities of the steps involved in the generation process. This scheme requires that different sequences of steps must not lead to the same tree. We achieve this by defining a canonical ordering of the links in a dependency tree. Each generation step corresponds to the construction of a dependency link in the canonical order.

Given two dependency links l and l' with the heads being h and h' and the modifiers being m and m', respectively, the order between l and l' are determined as follows:

- If $h \neq h'$ and there is a directed path from one (say h) to the other (say h'), then l' precedes l.
- If $h \neq h'$ and there does not exist a directed path between h and h', the order between l and l' is determined by the order of h and h' in the sentence (h precedes $h' \Rightarrow l$ precedes l').
- If $h = h'$ and the modifiers m and m' are on different sides of h, the link with modifier on the right precedes the other.
- If $h = h'$ and the modifiers m and m' are on the same side of the head h, the link with its modifier closer to h precedes the other one.

For example, the canonical order of the links in the dependency tree in Figure 1 is: (1, 2, L), (5, 6, R), (8, 9, L), (7, 9, R), (5, 7, R), (4, 5, R), (3, 4, R), (2, 3, L), (0, 3, L).

The generation process according to the canonical order is similar to the head outward generation process in (Collins, 1999), except that it is bottom-up whereas Collins' models are top-down.

Suppose the dependency tree T is constructed in steps G_1, …, G_N in the canonical order of the dependency links, where N is the number of words in the sentence. We can compute the probability of T as follows:

$$\begin{aligned} & P(T \mid S) \\ &= P(G_1, G_2, ..., G_N \mid S) \\ &= \prod_{i=1}^{N} P(G_i \mid S, G_1, ..., G_{i-1}) \end{aligned}$$

Following (Klein and Manning, 2004), we require that the creation of a dependency link from head h to modifier m be preceded by placing a left STOP and a right STOP around the modifier m and ¬STOP between h and m.

Let E_w^L (and E_w^R) denote the event that there are no more modifiers on the left (and right) of a word w. Suppose the dependency link created in the step i is (u, v, d). If $d = L$, G_i is the conjunction of the four events: E_u^R, E_u^L, $\neg E_v^L$ and $link_L(u, v)$. If $d = R$, G_i consists of four events: E_v^L, E_v^R, $\neg E_u^R$ and $link_R(u, v)$.

The event G_i is conditioned on $S, G_1, ..., G_{i-1}$, which are the words in the sentence and a forest of trees constructed up to step i-1. Let C_w^L (and C_w^R) be the number of modifiers of w on its left (and right). We make the following independence assumptions:

- Whether there is any more modifier of w on the d side depends only on the number of modifiers already found on the d side of w. That is, E_w^d depends only on w and C_w^d.
- Whether there is a dependency link from a word h to another word m depends only on the words h and m and the number of modifiers of h between m and h. That is,
  - $link_R(u,v)$ depends only on u, v, and C_u^R.
  - $link_L(u,v)$ depends only on u, v, and C_v^L.

Suppose G_i corresponds to a dependency link (u, v, L). The probability $P(G_i \mid S, G_1, ..., G_{i-1})$ can be computed as:

$$\begin{aligned} & P(G_i \mid S, G_1, ..., G_{i-1}) \\ &= P(E_u^L, E_u^R, \neg E_v^L, link_L(u,v) \mid S, G_1, ..., G_{i-1}) \\ &= P(E_u^L \mid u, C_u^L) \times P(E_u^R \mid u, C_u^R) \times \\ &\quad (1 - P(E_v^L \mid v, C_v^L)) \times P(link_L(u,v) \mid u, v, C_v^L) \end{aligned}$$

The events E_w^R and E_w^L correspond to the STOP events in (Collins, 1999) and (Klein and Manning, 2004). They are crucial for modeling the number of dependents. Without them, the parse trees often contain some 'obvious' errors, such as determiners taking arguments, or prepositions having arguments on their left (instead of right).

Our model requires three types of parameters:

- $P(E_w^d \mid w, C_w^d)$, where w is a word, d is a direction (left or right). This is the probability of a STOP after taking C_w^d modifiers on the d side.
- $P(link_R(u,v) \mid u, v, C_u^R)$ is the probability of v being the $(C_u^R + 1)$'th modifier of u on the right.
- $P(link_L(u,v) \mid u, v, C_v^L)$ is the probability of u being the $(C_v^L + 1)$'th modifier of v on the left.

The Maximum Likelihood estimations of these parameters can be obtained from the frequency counts in the training corpus:

- C(w, c, d): the frequency count of w with c modifiers on the d side.
- C(u, v, c, d): If $d = L$, this is the frequency count words u and v co-occurring in a sentence and v has c modifiers between itself and u. If $d = R$, this is the frequency count words u and v co-occurring in a sentence and u has c modifiers between itself and v.
- K(u, v, c, d): similar to C(u, v, c, d) with an additional constraint that $link_d(u, v)$ is true.

$$P\left(E_w^d \mid w, C_w^d\right) = \frac{C(w,c,d)}{\sum_{c' \geq c} C(w,c',d)}$$, where $c = C_w^d$;

$$P\left(link_R(u,v) \mid u, v, C_u^R\right) = \frac{K(u,v,c,R)}{C(u,v,c,R)},$$

where $c = C_u^R$;

$$P\left(link_L(u,v) \mid u, v, C_v^L\right) = \frac{K(u,v,c,L)}{C(u,v,c,L)},$$

where $c = C_v^L$.

We compute the probability of the tree conditioned on the words. All parameters in our model are conditional probabilities where the left sides of the conditioning bar are binary variables. In contrast, most previous approaches compute joint probability of the tree and the words in the tree. Many of their model parameters consist of the probability of a word in a given context.

We use a dynamic programming algorithm similar to chart parsing as the decoder for this model. The algorithm builds a packed parse forest from bottom up in the canonical order of the parser trees. It attaches all the right children before attaching the left ones to maintain the canonical order as required by our model.

3 Similarity-based Smoothing

3.1 Distributional Word Similarity

Words that tend to appear in the same contexts tend to have similar meanings. This is known as the Distributional Hypothesis in linguistics (Harris, 1968). For example, the words test and exam are similar because both of them follow verbs such as administer, cancel, cheat on, conduct, ... and both of them can be preceded by adjectives such as academic, comprehensive, diagnostic, difficult, ...

Many methods have been proposed to compute distributional similarity between words (Hindle, 1990; Pereira et al., 1993; Grefenstette, 1994; Lin, 1998). Almost all of the methods represent a word by a feature vector where each feature corresponds to a type of context in which the word appeared. They differ in how the feature vectors are constructed and how the similarity between two feature vectors is computed.

We define the features of a word *w* to be the set of words that occurred within a small context window of *w* in a large corpus. The context window of an instance of *w* consists of the closest non-stop-word on each side of *w* and the stop-words in between. In our experiments, the set of stop-words are defined as the top 100 most frequent words in the corpus. The value of a feature *w′* is defined as the point-wise mutual information between the *w′* and *w*:

$$PMI(w,w') = -\log\left(\frac{P(w,w')}{P(w)P(w')}\right)$$

where *P*(*w*, *w*') is the probability of *w* and *w*' co-occur in a context window.

The similarity between two vectors is computed as the cosine of the angle between the vectors. The following are the top similar words for the word *keystone* obtained from the English Gigaword Corpus:

centrepiece 0.28, figment 0.27, fulcrum 0.21, culmination 0.20, albatross 0.19, bane 0.19, pariahs 0.18, lifeblood 0.18, crux 0.18, redoubling 0.17, apotheosis 0.17, cornerstones 0.17, perpetuation 0.16, forerunners 0.16, shirking 0.16, cornerstone 0.16, birthright 0.15, hallmark 0.15, centerpiece 0.15, evidenced 0.15, germane 0.15, gist 0.14, reassessing 0.14, engrossed 0.14, Thorn 0.14, biding 0.14, narrowness 0.14, linchpin 0.14, enamored 0.14, formalised 0.14, tenths 0.13, testament 0.13, certainties 0.13, forerunner 0.13, re-evaluating 0.13, antithetical 0.12, extinct 0.12, rarest 0.12, imperiled 0.12, remiss 0.12, hindrance 0.12, detriment 0.12, prouder 0.12, upshot 0.12, cosponsor 0.12, hiccups 0.12, premised 0.12, perversion 0.12, destabilisation 0.12, prefaced 0.11, ……

3.2 Similarity-based Smoothing

The parameters in our model consist of conditional probabilities P(*E*|*C*) where *E* is the binary variable $link_d(u, v)$ or E_w^d and the context *C* is either $\left[w, C_w^d\right]$ or $\left[u, v, C_w^d\right]$, which involves one or two words in the input sentence. Due to the sparseness of natural language data, the contexts observed in the training data only covers a tiny fraction of the contexts whose probability distribution are needed during parsing. The standard approach is to back off the probability to word classes (such as part-of-speech tags). We have taken a different approach. We search in the train-

ing data to find a set of similar contexts to C and estimate the probability of E based on its probabilities in the similar contexts that are observed in the training corpus.

Similarity-based smoothing was used in (Dagan et al., 1999) to estimate word co-occurrence probabilities. Their method performed almost 40% better than the more commonly used back-off method. Unfortunately, similarity-based smoothing has not been successfully applied to statistical parsing up to now.

In (Dagan et al., 1999), the bigram probability $P(w_2|w_1)$ is computed as the weighted average of the conditional probability of w_2 given similar words of w_1.

$$P_{SIM}(w_2 \mid w_1) = \sum_{w'_1 \in S(w_1)} \frac{sim(w_1, w'_1)}{norm(w_1)} P_{MLE}(w_2 \mid w'_1)$$

where $sim(w_1, w'_1)$ denotes the similarity (or an increasing function of the similarity) between w_1 and w'_1, $S(w_1)$ denote the set of words that are most similar to w_1 and $norm(w_1)$ is the normalization factor $norm(w_1) = \sum_{w'_1 \in S(w_1)} sim(w_1, w'_1)$.

The underlying assumption of this smoothing scheme is that a word is more likely to occur after w_1 if it tends to occur after similar words of w_1.

We make a similar assumption: the probability $P(E|C)$ of event E given the context C is computed as the weight average of $P(E|C')$ where C' is a similar context of C and is attested in the training corpus:

$$P_{SIM}(E \mid C) = \sum_{C' \in S(C) \cap O} \frac{sim(C, C')}{norm(C)} P_{MLE}(E \mid C')$$

where $S(C)$ is the set of top-K most similar contexts of C (in the experiments reported in this paper, K = 50); O is the set of contexts observed in the training corpus, $sim(C,C')$ is the similarity between two contexts and $norm(C)$ is the normalization factor.

In our model, a context is either $\left[w, C_w^d\right]$ or $\left[u, v, C_w^d\right]$. Their similar contexts are defined as:

$$S\left(\left[w, C_w^d\right]\right) = \left\{\left[w', C_{w'}^d\right] \middle| w' \in S(w)\right\}$$
$$S\left(\left[u, v, C_w^d\right]\right) = \left\{\left[u', v', C_w^d\right] \middle| u' \in S(u), v' \in S(v)\right\}$$

where $S(w)$ is the set of top-K similar words of w (K = 50).

Since all contexts used in our model contain at least one word, we compute the similarity between two contexts, sim(C, C'), as the geometric average of the similarities between corresponding words:

$$sim\left(\left[w, C_w^d\right], \left[w', C_{w'}^d\right]\right) = sim(w, w')$$
$$sim\left(\left[u, v, C_w^d\right], \left[u', v', C_{w'}^d\right]\right) = \sqrt{sim(u, u') \times sim(v, v')}$$

Similarity-smoothed probability is only necessary when the frequency count of the context C in the training corpus is low. We therefore compute

$$P(E \mid C) = \alpha\, P_{MLE}(E \mid C) + (1 - \alpha)\, P_{SIM}(E \mid C)$$

where the smoothing factor $\alpha = \frac{|C| + 1}{|C| + 5}$ and $|C|$ is the frequency count of the context C in the training data.

A difference between similarity-based smoothing in (Dagan et al., 1999) and our approach is that our model only computes probability distributions of binary variables. Words only appear as parts of contexts on the right side of the conditioning bar. This has two important implications. Firstly, when a context contains two words, we are able to use the cross product of the similar words, whereas (Dagan et al., 1999) can only use the similar words of one of the words. This turns out to have significant impact on the performance (see Section 4).

Secondly, in (Dagan et al., 1999), the distribution $P(\bullet|w'_1)$ may itself be sparsely observed. When $P_{MLE}(w_2 \mid w'_1)$ is 0, it is often due to data sparseness. Their smoothing scheme therefore tends to under-estimate the probability values. This problem is avoided in our approach. If a context did not occur in the training data, we do not include it in the average. If it did occur, the Maximum Likelihood estimation is reasonably accurate even if the context only occurred a few times, since the entropy of the probability distribution is upper-bounded by log 2.

4 Experimental Results

We experimented with our parser on the Chinese Treebank (CTB) 3.0. We used the same data split as (Bikel, 2004): Sections 1-270 and 400-931 as

the training set, Sections 271-300 as testing and Sections 301-325 as the development set. The CTB contains constituency trees. We converted them to dependency trees using the same method and the head table as (Bikel, 2004). Parsing Chinese generally involve segmentation as a preprocessing step. We used the gold standard segmentation in the CTB.

The distributional similarities between the Chinese words are computed using the Chinese Gigaword corpus. We did not segment the Chinese corpus when computing the word similarity.

We measure the quality of the parser by the undirected accuracy, which is defined as the number of correct undirected dependency links divided by the total number of dependency links in the corpus (the treebank parse and the parser output always have the same number of links). The results are summarized in Table 1. It can be seen that the performance of the parser is highly correlated with the length of the sentences.

Max Sentence Length	10	15	20	40
Undirected Accuracy	90.8	85.6	84.0	79.9

Table 1. Evaluation Results on CTB 3.0

We also experimented with several alternative models for dependency parsing. Table 2 summerizes the results of these models on the test corpus with sentences up to 40 words long.

One of the characteristics of our parser is that it uses the similar words of both the head and the modifier for smoothing. The similarity-based smoothing method in (Dagan et al., 1999) uses the similar words of one of the words in a bigram. We can change the definition of similar context as follows so that only one word in a similar context of C may be different from a word in C (see Model (b) in Table 2):

$$S\left(\left[u, v, C_w^d\right]\right) = \left\{\left[u', v, C_w^d\right] \middle| u' \in S(u)\right\} \cup \left\{\left[u, v', C_w^d\right] \middle| v' \in S(v)\right\}$$

where w is either v or u depending on whether d is L or R. This change led to a 2.2% drop in accuracy (compared with Model (a) in Table 2), which we attribute to the fact that many contexts do not have similar contexts in the training corpus.

Since most previous parsing models maximize the joint probability of the parse tree and the sentence P(T, S) instead of P(T | S), we also implemented a joint model (see Model (c) in Table 2):

$$P(T,S) = \prod_{i=1}^{N} P\left(E_{m_i}^L \mid m_i, C_{m_i}^L\right) \times P\left(E_{m_i}^R \mid m_i, C_{m_i}^R\right) \times \left(1 - P\left(E_{h_i}^d \mid h_i, C_{h_i}^d\right)\right) \times P\left(m_i \mid h_i, C_{h_i}^{d_i}\right)$$

where h_i and m_i are the head and the modifier of the i'th dependency link. The probability $P\left(m_i \mid h_i, C_{h_i}^{d_i}\right)$ is smoothed by averaging the probabilities $P\left(m_i \mid h'_i, C_{h_i}^{d_i}\right)$, where h'_i is a similar word of h_i, as in (Dagan et al., 1999). The result was a dramatic decrease in accuracy from the conditional model's 79.9%. to 66.3%.

Our use of distributional word similarity can be viewed as assigning soft clusters to words. In contrast, parts-of-speech can be viewed as hard clusters of words. We can modify both the conditional and joint models to use part-of-speech tags, instead of words. Since there are only a small number of tags, the modified models used MLE without any smoothing except using a small constant as the probability of unseen events. Without smoothing, maximizing the conditional model is equivalent to maximizing the joint model. The accuracy of the unlexicalized models (see Model (d) and Model (e) in Table 2) is 71.1% which is considerably lower than the strictly lexicalized conditional model, but higher than the strictly lexicalized joint model. This demonstrated that soft clusters obtained through distributional word similarity perform better than the part-of-speech tags when used appropriately.

Models		**Accuracy**
(a)	Strictly lexicalized conditional model	**79.9**
(b)	At most one word is different in a similar context	77.7
(c)	Strictly lexicalized joint model	66.3
(d)	Unlexicalized conditional models	71.1
(e)	Unlexicalized joint models	71.1

Table 2. Performance of Alternative Models

5 Related Work

Previous parsing models (e.g., Collins, 1997; Charniak, 2000) maximize the joint probability P(*S*, *T*) of a sentence *S* and its parse tree *T*. We maximize the conditional probability P(*T* | *S*). Although they are theoretically equivalent, the use of conditional model allows us to take advantage of similarity-based smoothing.

Clark et al. (2002) also computes a conditional probability of dependency structures. While the probability space in our model consists of all possible non-projective dependency trees, their probability space is constrained to all the dependency structures that are allowed by a Combinatorial Category Grammar (CCG) and a category dictionary (lexicon). They therefore do not need the STOP markers in their model. Another major difference between our model and (Clark et al., 2002) is that the parameters in our model consist exclusively of conditional probabilities of binary variables.

Ratnaparkhi's maximum entropy model (Ratnaparkhi, 1999) is also a conditional model. However, his model maximizes the probability of the action during each step of the parsing process, instead of overall quality of the parse tree.

Yamada and Matsumoto (2002) presented a dependency parsing model using support vector machines. Their model is a discriminative model that maximizes the differences between scores of the correct parse and the scores of the top competing incorrect parses.

In many dependency parsing models such as (Eisner, 1996) and (MacDonald et al., 2005), the score of a dependency tree is the sum of the scores of the dependency links, which are computed independently of other links. An undesirable consequence of this is that the parser often creates multiple dependency links that are separately likely but jointly improbable (or even impossible). For example, there is nothing in such models to prevent the parser from assigning two subjects to a verb. In the DMV model (Klein and Manning, 2004), the probability of a dependency link is partly conditioned on whether or not there is a head word of the link already has a modifier. Our model is quite similar to the DMV model, except that we compute the conditional probability of the parse tree given the sentence, instead of the joint probability of the parse tree and the sentence.

There have been several previous approaches to parsing Chinese with the Penn Chinese Treebank (e.g., Bikel and Chiang, 2000; Levy and Manning, 2003). Both of these approaches employed phrase-structure joint models and used part-of-speech tags in back-off smoothing. Their results were evaluated with the precision and recall of the bracketings implied in the phrase structure parse trees. In contrast, the accuracy of our model is measured in terms of the dependency relationships. A dependency tree may correspond to more than one constituency trees. Our results are therefore not directly comparable with the precision and recall values in previous research. Moreover, it was argued in (Lin 1995) that dependency based evaluation is much more meaningful for the applications that use parse trees, since the semantic relationships are generally embedded in the dependency relationships.

6 Conclusion

To the best of our knowledge, all previous natural language parsers have to rely on part-of-speech tags. We presented a strictly lexicalized model for dependency parsing that only relies on word statistics. We compared our parser with an unlexicalized parser that employs the same probabilistic model except that the parameters are estimated using gold standard tags in the Chinese Treebank. Our experiments show that the strictly lexicalized parser significantly outperformed its unlexicalized counter-part.

An important distinction between our statistical model from previous parsing models is that all the parameters in our model are conditional probability of binary variables. This allows us to take advantage of similarity-based smoothing, which has not been successfully applied to parsing before.

Acknowledgements

The authors would like to thank Mark Steedman for suggesting the comparison with unlexicalized parsing in Section 4 and the anonymous reviewers for their comments. This work was supported in part by NSERC, the Alberta Ingenuity Centre for Machine Learning and the Canada Research

Chairs program. Qin Iris Wang was also supported by iCORE Scholarship.

References

Daniel M. Bikel. 2004. Intricacies of Collins' Parsing Model. *Computational Linguistics*, 30(4), pp. 479-511.

Daniel M. Bikel and David Chiang. 2000. Two Statistical Parsing Models applied to the Chinese Treebank. In *Proceedings of the second Chinese Language Processing Workshop*, pp. 1-6.

Eugene Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the Second Meeting of North American Chapter of Association for Computational Linguistics (NAACL-2000)*, pp. 132-139.

Stephen Clark, Julia Hockenmaier and Mark Steedman. 2002. Building Deep Dependency Structures with a Wide-Coverage CCG Parser. In *Proceedings of the 40th Annual Meeting of the ACL*, pp. 327-334.

Michael Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pp. 184-191. Santa Cruz.

Michael Collins. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the ACL (jointly with the 8th Conference of the EACL)*, pp. 16-23. Madrid.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. PhD Dissertation, University of Pennsylvania.

Ido Dagan, Lillian Lee and Fernando Pereira. 1999. Similarity-based models of cooccurrence probabilities. *Machine Learning, Vol. 34(1-3) special issue on Natural Language Learning*, pp. 43-69.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-96*, pp. 340-345, Copenhagen.

Daniel Gildea. 2001. Corpus Variation and Parser Performance. In Proceedings of *EMNLP-2001*, pp. 167-202. Pittsburgh, PA.

Gregory Grefenstette. 1994. *Explorations in Automatic Thesaurus Discovery.* Kluwer Academic Press, Boston, MA.

Zelig S. Harris. 1968. *Mathematical Structures of Language*. Wiley, New York.

Donald Hindle. 1990. Noun Classification from Predicate-Argument Structures. In *Proceedings of ACL-90*, pp. 268-275. Pittsburg, Pennsylvania.

Dan Klein and Chris Manning. 2002. Fast exact inference with a factored model for natural language parsing. In *Proceedings of Neural Information Processing Systems*.

Dan Klein and Chris Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the ACL,* pp. 423-430.

Dan Klein and Chris Manning. 2004. Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency. In *Proceedings of the 42nd Annual Meeting of the ACL,* pp. 479-486.

Roger Levy and Chris Manning. 2003. Is it harder to parse Chinese, or the Chinese Treebank? In *Proceedings of the 41st Annual Meeting of the ACL,* pp. 439-446.

Dekang Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of IJCAI-95*, pp.1420-1425.

Dekang Lin. 1998. Automatic Retrieval and Clustering of Similar Words. In *Proceeding of COLING-ACL98*, pp. 768-774. Montreal, Canada.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL-2005*, pp. 91-98.

Fernando Pereira, Naftali Z. Tishby, and Lillian Lee. 1993. Distributional clustering of English words. In *Proceedings of ACL-1993*, pp. 183-190, Columbus, Ohio.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pp.195-206.

Efficient extraction of grammatical relations

Rebecca Watson, John Carroll† and Ted Briscoe
Computer Laboratory, University of Cambridge, Cambridge, CB3 OFD, UK
firstname.lastname@cl.cam.ac.uk
†Department of Informatics, University of Sussex, Brighton BN1 9QH, UK
J.A.Carroll@sussex.ac.uk

Abstract

We present a novel approach for applying the Inside-Outside Algorithm to a packed parse forest produced by a unification-based parser. The approach allows a node in the forest to be assigned multiple inside and outside probabilities, enabling a set of 'weighted GRs' to be computed directly from the forest. The approach improves on previous work which either loses efficiency by unpacking the parse forest before extracting weighted GRs, or places extra constraints on which nodes can be packed, leading to less compact forests. Our experiments demonstrate substantial increases in parser accuracy and throughput for weighted GR output.

1 Introduction

RASP is a robust statistical analysis system for English developed by Briscoe and Carroll (2002). It contains a syntactic parser which can output analyses in a number of formats, including (n-best) syntactic trees, robust minimal recursion semantics (Copestake, 2003), grammatical relations (GRs), and weighted GRs. The weighted GRs for a sentence comprise the set of grammatical relations in all parses licensed for that sentence, each GR is weighted based on the probabilities of the parses in which it occurs. This weight is normalised to fall within the range $[0,1]$ where 1.0 indicates that all parses contain the GR. Therefore, high precision GR sets can be determined by thresholding on the GR weight (Carroll and Briscoe, 2002). Carroll and Briscoe compute weighted GRs by first unpacking all parses or the n-best subset from the parse forest. Hence, this approach is either (a) inefficient (and for some examples impracticable) if a large number of parses are licensed by the grammar, or (b) inaccurate if the number of parses unpacked is less than the number licensed by the grammar.

In this paper, we show how to obviate the need to trade off efficiency and accuracy by extracting weighted GRs directly from the parse forest using a dynamic programming approach based on the Inside-Outside algorithm (IOA) (Baker, 1979; Lari and Young, 1990). This approach enables efficient calculation of weighted GRs over *all parses* and substantially improves the throughput and memory usage of the parser. Since the parser is unification-based, we also modify the parsing algorithm so that local ambiguity packing is based on feature structure equivalence rather than subsumption.

Similar dynamic programming techniques that are variants of the IOA have been applied for related tasks, such as parse selection (Johnson, 2001; Schmid and Rooth, 2001; Geman and Johnson, 2002; Miyao and Tsujii, 2002; Kaplan et al., 2004; Taskar et al., 2004). The approach we take is similar to Schmid and Rooth's (2001) adaptation of the algorithm, where 'expected governors' (similar to our 'GR specifications') are determined for each tree, and alternative nodes in the parse forest have the *same lexical head*. Initially, they create a packed parse forest and during a second pass the parse forest nodes are split if multiple lexical heads occur. The IOA is applied over this split data structure. Similarly, Clark and Curran (2004) alter their packing algorithm so that nodes in the packed chart have the same semantic head and 'unfilled' GRs. Our ap-

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 160–170,
Vancouver, October 2005.

proach is novel in that while calculating inside probabilities we allow any node in the parse forest to have *multiple semantic heads*.

Clark and Curran (2004) apply Miyao and Tsujii's (2002) dynamic programming approach to determine weighted GRs. They outline an alternative parse selection method based on the resulting weighted GRs: select the (consistent) GR set with the highest average weighted GR score. We apply this parse selection approach and achieve 3.01% relative reduction in error. Further, the GR set output by this approach is a *consistent set* whereas the high precision GR sets outlined in (Carroll and Briscoe, 2002) are neither consistent nor coherent.

The remainder of this paper is organised as follows: Section 2 gives details of the RASP system that are relevant to this work. Section 3 describes our test suite and experimental environment. Changes required to the current parse forest creation algorithm are discussed in Section 4, while Section 5 outlines our dynamic programming approach for extracting weighted GRs (EWG). Section 6 presents experimental results showing (a) improved efficiency achieved by EWG, (b) increased upper bounds of precision and recall achieved using EWG, and (c) increased accuracy achieved by a parse selection algorithm that would otherwise be too inefficient to consider. Finally, Section 7 outlines our conclusions and future lines of research.

2 The RASP System

RASP is based on a pipelined modular architecture in which text is pre-processed by a series of components including sentence boundary detection, tokenisation, part of speech tagging, named entity recognition and morphological analysis, before being passed to a statistical parser[1]. A brief overview of relevant aspects of syntactic processing in RASP is given below; for full details of system components, see Briscoe and Carroll (1995; 2002; 2005)[2].

[1]Processing times given in this paper do not include these pre-processing stages, since they take negligible time compared with parsing.

[2]RASP is freely available for research use; visit http://www.informatics.susx.ac.uk/research/nlp/rasp/

2.1 The Grammar

Briscoe and Carroll (2005) describe the (manually-written) feature-based unification grammar and the rule-to-rule mapping from local trees to GRs. The mapping specifies for each grammar rule the semantic head(s) of the rule (henceforth, head), and one or more GRs that should be output (optionally depending on feature values instantiated at parse time). For example, Figure 1 shows a grammar rule analysing a verb phrase followed by a prepositional phrase modifier. The rule identifies the first daughter (`1`) as the semantic head, and specifies that one of five possible GRs is to be output, depending on the value of the `PSUBCAT` syntactic feature; so, for example, if the feature has the value `NP`, then the relation is `ncmod` (non-clausal modifier), with slots filled by the semantic heads of the first and second daughters (the `1` and `2` arguments).

Before parsing, a context free backbone is derived automatically from the grammar, and an LALR(1) parse table is computed from this backbone (Carroll, 1993, describes the procedure in detail). Probabilities are associated with actions in the parse table, by training on around 4K sentences from the Susanne corpus (Sampson, 1995), each sentence having been semi-automatically converted from a treebank bracketing to a tree conforming to the unification grammar (Briscoe and Carroll, 1995).

2.2 The Parse Forest

When parsing, the LALR table action probabilities are used to assign a score to each newly derived (sub-)analysis. Additionally, on each reduce action (i.e. complete application of a rule), the rule's daughters are unified with the sequence of sub-analyses being consumed. If unification fails then the reduce action is aborted. Local ambiguity packing (packing, henceforth) is performed on the basis of feature structure subsumption. Thus, the parser builds and returns a compact structure that efficiently represents all parses licensed by the grammar: the parse forest. Since unification often fails it is not possible to apply beam or best first search strategies during construction of the parse forest; statistically high scoring paths often end up in unification failure. Hence, the parse forest represents all parses licensed by the grammar.

```
V1/vp_pp : V1[MOD +] --> H1 P2[ADJ -, WH -] :
    1 :
    2 = [PSUBCAT NP], (ncmod _ 1 2) :
    2 = [PSUBCAT NONE], (ncmod prt 1 2) :
    2 = [PSUBCAT (VP, VPINF, VPING, VPPRT, AP)], (xmod _ 1 2) :
    2 = [PSUBCAT (SFIN, SINF, SING)], (cmod _ 1 2) :
    2 = [PSUBCAT PP], (pmod 1 2).
```

Figure 1: Example grammar rule, showing the rule name and syntactic specification (on the first line), identification of daughter `1` as the semantic head (second line), and possible GR outputs depending on the parse-time value of the `PSUBCAT` feature of daughter `2` (subsequent lines).

Figure 2 shows a simplified parse forest containing three parses generated for the following preprocessed text[3]:

```
I_PPIS1 see+ed_VVD the_AT man_NN1
in_II the_AT park_NN1
```

The GR specifications shown are instantiated based on the values of syntactic features at daughter nodes, as discussed in Section 2.1 above. For example, the `V1/vp_pp` sub-analysis (towards the left hand side of the Figure) contains the instantiated GR specification <1, (ncmod _ 1 2)> since its second daughter has the value `NP` for its `PSUBCAT` feature.

Henceforth, we will use the term 'node' to refer to data structures in our parse forest corresponding to a rule instantiation: a sub-analysis resulting from application of a reduce action. Back pointers are stored in nodes, indicating which daughters were used to create the sub-analysis. These pointers provide a means to traverse the parse forest during subsequent processing stages. A 'packed node' is a node representing a sub-analysis that is subsumed by, and hence packed into, another node. Packing is considered for nodes only if they are produced in the same LR state and represent sub-analyses with the same word span. A parse forest can have a number of root nodes, each one dominating analyses spanning the whole sentence with the specified top category.

2.3 Parser Output

From the parse forest, RASP unpacks the 'n-best'[4] syntactic trees using a depth-first beam search (Carroll, 1993). There are a number of types of analysis output available, including syntactic tree, grammatical relations (GRs) and robust minimal recursion semantics (RMRS). Each of these is computed from the n-best trees.

Another output possibility is weighted GRs (Carroll and Briscoe, 2002); this is the unique set of GRs from the n-best GRs, each GR weighted according to the sum of the probabilities of the parses in which it occurs. Therefore, a number of processing stages determine this output: unpacking the n-best syntactic trees, determining the corresponding n-best GR sets and finding the unique set of GRs and corresponding weights.

The GRs for each parse are computed from the set of GR specifications at each node, passing the (semantic) head of each sub-analysis up to the next higher level in the parse tree (beginning from word nodes). GR specifications for nodes (which, if required, have been instantiated based on the features of daughter nodes) are referred to as 'unfilled' until the slots containing numbers are 'filled' with the corresponding heads of daughter nodes. For example, the grammar rule named `NP/det_n` has the unfilled GR specification <2, (det 2 1)>. Therefore, if an `NP/det_n` local tree has two daughters with heads *the* and *cat* respectively, the resulting filled GR specification will be <cat, (det cat the)>, i.e. the head of the local tree is *cat* and the GR output is (det cat the).

Figure 3 illustrates the n-best GRs and the corresponding (non-normalised and normalised) weighted GRs for the sentence *I saw the man in the park*. The corresponding parse forest for this example is shown in Figure 2. Weights on the GRs are normalised probabilities representing the weighted proportion of parses in which the GR occurs. This weighting is in practice calculated as the sum of parse probabilities for parses con-

[3]The part of speech tagger uses a subset of the Lancaster CLAWS2 tagset – http://www.comp.lancs.ac.uk/computing/research/ucrel/claws2tags.html

[4]This number n is specified by the user, and represents the maximal number of parses to be unpacked.

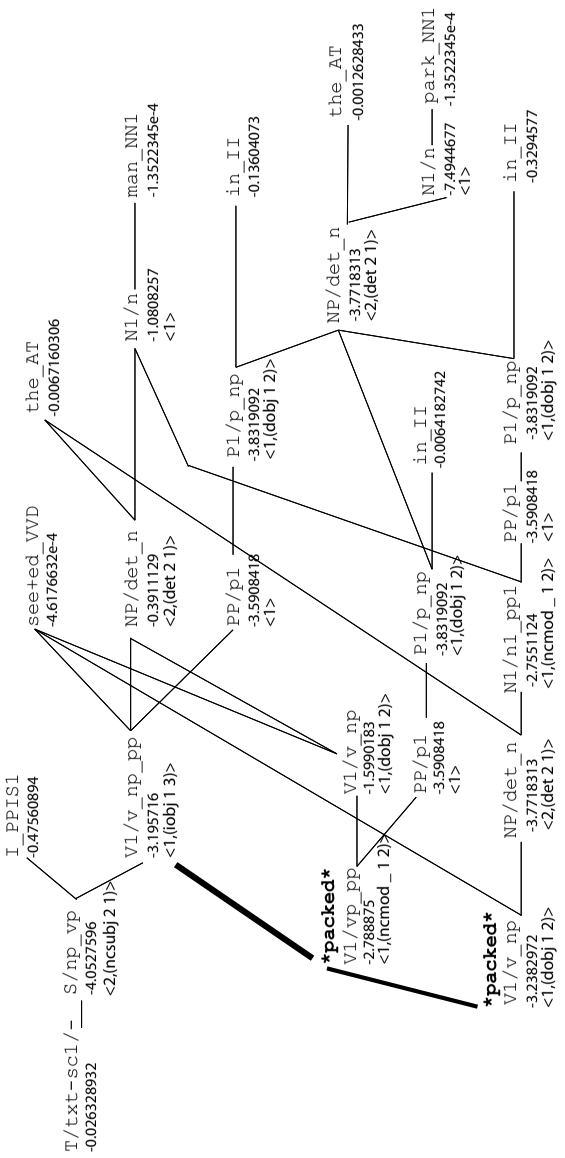


Figure 2: Simplified parse forest for *I saw the man in the park*. Each element in the directed acyclic graph represents a node in the parse forest and is shown with the sub-analysis' rule name, reduce probability (or shift probability at word nodes) and (instantiated) GR specifications. Two nodes are packed into the `V1/v_np_pp` node, so there will be three alternative parses for the sentence. Nodes with multiple in-going pointers on their left are shared. All thin lines indicate pointers from left to right, i.e. from mother to daughter nodes.

taining the specific GR, normalised by the sum of all parse probabilities. For example, the GR (iobj see+ed in) is in one parse with probability -28.056154, the non-normalised score. The sum of all parse probabilities is -28.0200896. Therefore, the normalised probability (and final weight) of the GR is $10^{(-28.056154--28.0200896)} = 0.920314$[5].

3 Data and Methods

King et al. (2003) outline the development of the PARC 700 Dependency Bank (henceforth, DepBank), a gold-standard set of relational dependencies for 700 sentences (originally from the Wall Street Journal) drawn at random from Section 23 of the Penn Treebank. Briscoe and Carroll (2005) extended DepBank with a set of gold-standard RASP GRs that we use to measure parser accuracy.

We use the same 560 sentence subset from the DepBank utilised by Kaplan et al. (2004) in their study of parser accuracy and efficiency. All experimental results are obtained using this test suite on an AMD Opteron 2.5GHz CPU with 1GB of Ram on a 64 bit version of Linux. The parser's output is evaluated using a relational dependency evaluation scheme (Carroll et al., 1998; Lin, 1998) and standard evaluation measures: precision, recall and F_1.

4 Local Ambiguity Packing

Oepen and Carroll (2000) note that when using subsumption-based packing with a unification-based grammar, the parse forest may implicitly represent some parses that are not actually licensed by the grammar; these will have values for one or more features that are locally but not globally consistent. This is not a problem when computing GRs from trees that have already been unpacked, since the relevant unifications will have been checked during the unpacking process, and will have caused the affected trees to be filtered out. Unification fails for at least one packed tree in approximately 10% of the sentences in the test suite. However, such inconsistent trees are a problem for any approach to probability computation over the parse forest that is based on the Inside-Outside algorithm (IOA). For our efficient weighted GR extraction technique we therefore modify the parsing algorithm so that packing is based on feature structure *equality* rather than subsumption.

Oepen and Carroll give definitions and implementation details for subsumption and equality operations, which we adopt. In the experiments below, we refer to versions of the parser with subsumption and equality based packing as SUB-PACKING and EQ-PACKING respectively.

5 Extracting Weighted GRs

Parse forest unpacking consumes larger amounts of CPU time and memory as the number of parses to unpack (n-best) increases. Carroll and Briscoe (2002) demonstrate that increasing the size of the n-best list increases the upper bound on precision (i.e. when low-weighted GRs are filtered out). Therefore, if practicable, it is preferable to include all possible parses when calculating weighted GRs. We describe below a dynamic programming approach (EWG) based on the IOA to efficiently extract weighted GRs directly from the parse forest. EWG calculates weighted GRs over all parses represented in the parse forest.

Inside and outside probabilities are analogous to the forward and backward probabilities of markov model algorithms. The inside probability represents the probability of all possible sub-analyses of a node. Conversely, the outside probability represents the probability of all analyses for which the node is a sub-analysis.

The IOA is ideal for our task, as the product of inside and outside probabilities for a sub-analysis constitutes part of the sum for the non-normalised weight of each GR (arising from the GR specification in the sub-analysis). Further, we can apply the sum of inside probabilities for each root-node, to normalise the weighted GRs.

5.1 Implementation

Three processing stages are required to determine weighted GRs over the parse forest, calculating (1) filled GRs and corresponding inside probabili-

[5] As we are dealing with log probabilities, summation and subtraction of these probabilities is not straightforward. Multiplication of probabilities X and Y, with log probabilities x and y respectively is determined using the formula $X \times Y = x + y$, division using $X \div Y = x - y$, summation using $X + Y = x + log_{10}(1 + 10^{(y-x)})$ and subtraction using $X - Y = x + log_{10}(1 - 10^{(y-x)})$.

N−BEST GRS

parse (log) probability:−28.056154
(ncsubj see+ed I _)
(iobj see+ed in)
(dobj see+ed man)
(dobj in park)
(det park the)
(det man the)

parse (log) probability:−29.11871
(ncsubj see+ed I _)
(ncmod _ see+ed in)
(dobj in park)
(det park the)
(dobj see+ed man)
(det man the)

parse (log) probability:−35.159805
(ncsubj see+ed I _)
(dobj see+ed man)
(det man the)
(ncmod _ man in)
(dobj in park)
(det park the)

(NON NORMALISED) WEIGHTED GRS

−28.0201	(ncsubj see+ed_VVD I_PPIS1 _)
−35.1598	(ncmod _ man_NN1 in_II)
−28.0201	(det park_NN1 the_AT)
−29.1187	(ncmod _ see+ed_VVD in_II)
−28.0562	(iobj see+ed_VVD in_II)
−28.0201	(dobj see+ed_VVD man_NN1)
−28.0201	(dobj in_II park_NN1)
−28.0201	(det man_NN1 the_AT)

(NORMALISED) WEIGHTED GRS

1.0	(det park the)
1.0	(det man the)
1.0	(dobj see+ed man)
1.0	(dobj in park)
0.920314	(iobj see+ed in)
7.249102e−8	(ncmod _ man in)
7.968584e−2	(ncmod _ see+ed in)
1.0	(ncsubj see+ed I _)

Total Probability (log−sum of all parses): −28.0200896

Figure 3: The n-best GRs, and non-normalised/normalised weighted GRs determined from three parses for the sentence *I saw the man in the park*. Parse probabilities and non-normalised weights are shown as log probabilities. Weights and parse probabilities are shown with differing precision, however RASP stores all probabilities in log (base 10) form with double float precision.

ties, (2) outside (and non-normalised) probabilities of weighted GRs, and (3) normalised probabilities of weighted GRs.[6] The first two processing stages are covered in detail in the following sections, while the final stage simply entails normalising the probabilities by dividing each weight by the sum of all the parse probabilities (the sum of root-nodes' inside probabilities).

5.1.1 Inside probability and GR

To determine inside probabilities over the nodes in the parse forest, we need to propagate the head and corresponding inside probability upwards after filling the node's GR specification. The inside probability of node n is usually calculated over the parse forest by multiplying the inside probability of the node's daughters and the probability $r(n)$ of the node itself (i.e. the probability of the shift or reduce action that caused the node to be created). Therefore, if a node has daughters D_1 and D_2, then the inside probability E_n is calculated using:

$$E_n = E_{D_1} \times E_{D_2} \times r(n) \tag{1}$$

However, packed nodes each correspond to an alternative filled GR specification. Inside probabilities for these GR specifications need to be combined. If packed analyses n_p occur in node n then the inside probability of node n is:

$$E_n = \sum_{i \in (n,n_p)} E_i \tag{2}$$

Further, the alternative GR specifications may not necessarily specify the same head as the node's GR specification and *multiple heads* may be passed up by the node. Hence, the summation in equation 2 needs to be conditioned on the possible heads of a node $H(node)$, where E_n^h is the inside probability of each head h for node n:

$$E_n^h = \sum_{i \in (n,n_p), H(i)=h} E_i \tag{3}$$

When multiple heads are passed up by daughter nodes, *multiple filled GR specifications* are found for the node. We create one filled GR specification for each possible combination of daughters' heads[7]. For example, consider the case where a node has daughters D_1 and D_2 with semantic heads $\{$*dog, cat*$\}$ and $\{$*an*$\}$ respectively. Here, we need to fill the GR specification <2, (det 2 1)> with two sets of daughters' heads: <dog, (det dog an)> and <cat, (det cat an)>.

As a node can have multiple filled GR specifications $GR(node)$, we alter equation 3 to:

$$E_n^h = \sum_{i \in (n,n_p)} \Big(\sum_{j \in GR(i), H(j)=h} E_j \Big) \tag{4}$$

Here, E_j (the inside probability of filled GR specification j) is determined by multiplying the inside probabilities of daughters' heads (that filled the GR specification) and the reduce probability of the node itself, i.e. using a modification of equation 1. Returning to the previous example, the inside probabilities of <dog, (det dog an)> and <cat, (det cat an)> will be equal to the reduce probability of the node multiplied by (a) the inside probability of head *an*, and (b) the inside probabilities of the heads *dog* and *cat*, respectively.

Hence, (a) calculation of inside probabilities takes into account multiple semantic heads, and (b) GR specifications are filled using every possible combination of daughters' heads. Each node n is processed in full as follows:

- Process each of the node's packed nodes n_p to determine the packed node's list of filled GR specifications and corresponding inside probabilities.
- Process the node n, with daughters D_n:
  - Instantiate n's GR specifications based on features of D_n.
  - Process each daughter in D_n to determine a list of possible semantic heads and corresponding inside probabilities for each.
  - Fill the GR specification of n with each possible combination of daughters' heads.

[6] Note that the IOA is not applied iteratively; a single pass only is required.

[7] The same word can appear as a head for more than one daughter of a node. This occurs if competing analyses have daughters with different word spans and, therefore, particular words can be considered in the span of either daughter. As the grammar permits both pre- and post- modifiers, it is possible for words in the 'overlapping' span to be passed up as heads for both daughters. Therefore, semantic heads are not combined unless they are different words.

Calculate the inside probability of each filled GR specification.

- Combine the alternative filled GR specifications of n and n_p, determining the list of unique semantic heads and corresponding inside probabilities using equation 4.

For each node, we propagate up a set of data structures $\{S_h\}$ that each contain one possible head h and corresponding inside probability. At word nodes, we simply return the word and the reduce score of the word as the semantic head and inside probability, respectively. Back pointers are also included to store the list of alternative filled GR specifications and corresponding inside probabilities, the reduce score for the node and the daughters' data structures (used to fill the GR specifications).

5.1.2 Outside probability determination

After the inside probabilities have been computed (bottom-up) the resulting data structure at the root-node is traversed to compute outside probabilities. The data structure created is split into alternative semantic heads for each node and, therefore, traversal to determine outside probabilities is relatively trivial: the outside probability of a filled GR specification is equal to the outside probability of the corresponding unique head of the node. Therefore, once we have created the new data structure, outside probabilities for each node can be determined over this structure in the regular fashion.

We calculate the outside probabilities (top-down) and, when we find filled GR specifications, we incrementally store the non-normalised weight of each GR. Each data structure S_h for head h, with outside probability F_h, is processed in full as follows:

- Process each of the GR specifications $GR(S_h)$. For each $j \in GR(S_h)$:
  - Let $F_j = F_h$ and calculate the probability of j, $I_j = E_j \times F_j$.
  - Add I_j to the (non-normalised) probability for j (in a hash table).
  - Process the data structure for each child head in j, $C(j)$. That is, the daughters' heads that filled the GR specification (resulting in j). For each $k \in C(j)$:
    - Calculate the outside probability of k (using the reduce probability of the node $r(n)$, stored in the data structure S_h):

$$F_k = F_j \times r(n) \times \prod_{i \in C(j), i \neq k} E_i \quad (5)$$

    - Queue the data structure k and corresponding outside probability F_k.[8]

6 Experimentation

6.1 Efficiency and Accuracy

The dynamic programming algorithm outlined in Section 5, EWG, provides an efficient and accurate method of determining weighted GRs directly from the parse forest. Figures 5 and 6 compare the efficiency of EWG to the EQ-PACKING and SUB-PACKING methods in terms of CPU time and memory, respectively[9]. Note that EWG applies equality-based packing to ensure only parses licensed by the grammar are considered (see Section 4).

As the maximum number of (n-best) parses increases, EQ-PACKING requires more time and memory than SUB-PACKING. However, if we compare these systems with an n-best value of 1, the difference in time and memory is negligible, suggesting that it is the unpacking stage which is responsible for the decreased throughput. For EWG we are forced to use equality-based packing, but these results suggest that using equality is not hurting the throughput of EWG.

Both figures illustrate that the time and memory required by EWG are static because the algorithm considers all parses represented in the parse forest regardless of the value of n-best specified. Therefore, the 'cross-over points' are of particular interest: at which n-best value is EWG's efficiency the same as that of the current system? This value is

[8] We apply a breadth first search (FIFO queue) to minimise multiple processing of shared data structures. If an outside probability is determined for a data structure already queued, then the probability is appended to the queued item. The steps are modified to enable multiple outside probabilities, i.e. summation over each outside probability when calculating I_j and F_k.

[9] CPU time and memory usage are as reported using the `time` function in Allegro Common Lisp 7.0 and do not include system start-up overheads or the time required for garbage collection.

approximately 580 and 100 for time and memory, respectively (comparing EWG to EQ-PACKING). Given that there are on average around 9000 parses per sentence in the test suite, these results indicate a substantial improvement in both efficiency and accuracy for weighted GR calculation. However, the median number of parses per sentence is around 50, suggesting that large parse numbers for a small subset of the test suite are skewing the arithmetic mean. Therefore, the complexity of this subset will significantly decrease throughput and EWG will improve efficiency for these sentences more so than for others.

The general relationship between sentence length and number of parses suggests that the EWG will be more beneficial for longer sentences. Figure 4 shows the distribution of number of parses over sentence length. The figure illustrates that the number of parses can not be reliably predicted from sentence length. Considering the cross-over points for time and memory, the number of sentences with more than 580 and 100 parses were 216 and 276, respectively. Thus, the EWG out-performs the current algorithm for around half of the sentences in the data set. The relative gain achieved reflects that a subset of sentences can significantly decrease throughput. Hence, the EWG is expected to improve the efficiency if a) longer sentences are present in the data set and b) n-best is set to a value greater than the cross-over point(s).

Upper bounds on precision and recall can be determined using weight thresholds over the GRs of 1.0 and 0.0, respectively[10]. Upper bounds of precision and recall provided by EWG are 79.57 and 82.02, respectively, giving an F_1 upper bound of 81.22%. However, considering the top 100 parses only, we achieve upper bounds on precision and recall of 78.77% and 81.18% respectively, resulting in an F_1 upper bound of 79.96%. Therefore, using EWG, we are able to achieve a relative increase of 6.29% for the F_1 upper bound on the task. Similarly, Carroll and Briscoe (2002) demonstrate (on an earlier, different test suite) that increasing the number of parses (n-best) from 100 to 1000 increases precision of weighted GR sets from 89.59% to 90.24%,

[10]In fact, in these experiments we use a threshold of $1 - \epsilon$ (with $\epsilon = 0.0001$) instead of a threshold of 1.0 to reduce the influence of very low ranked parses.

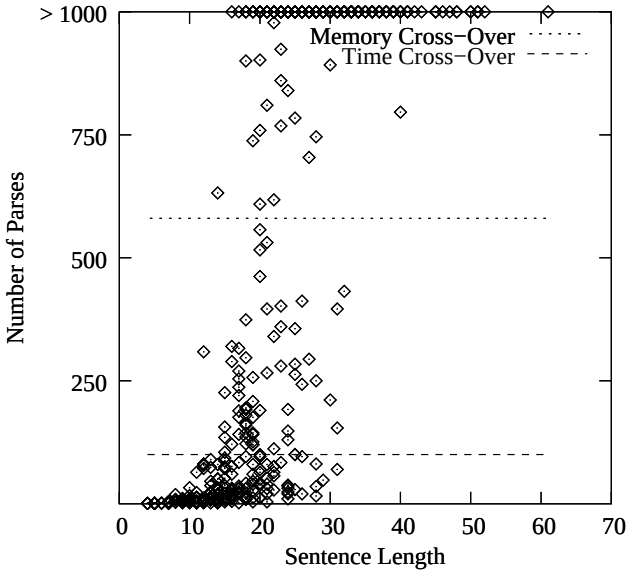


Figure 4: Scatter graph of number of parses to sentence length (one point per sentence). The cross-over points are illustrated for time and memory. The maximum number of parses shown is 1000, points plotted at 1000 correspond to equal to or greater than 1000 parses.

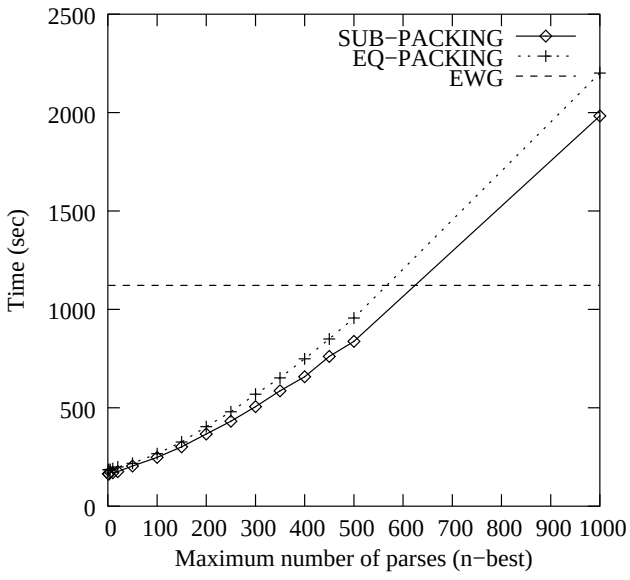


Figure 5: Comparison of total CPU time required by the different versions of the parsing system for calculation of weighted GRs over the n-best parses.

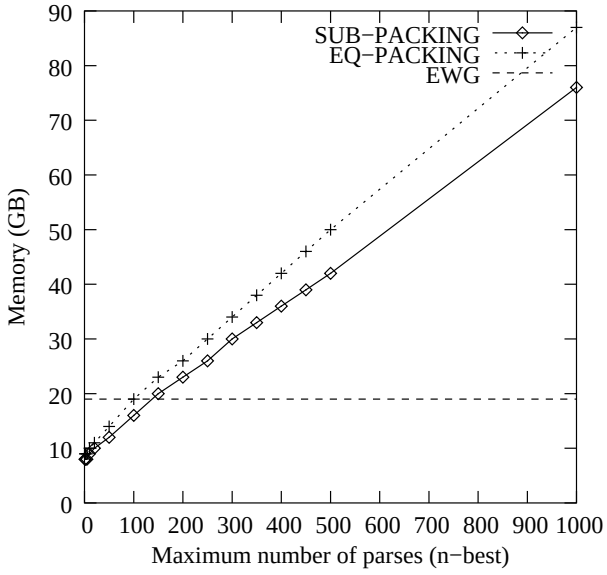


Figure 6: Comparison of total memory required by the different versions of the system for calculation of weighted GRs over the n-best parses.

a relative error reduction (RER) of 6.8%. Therefore, EWG achieves a substantial improvement in both efficiency and accuracy for weighted GR calculation; providing increased precision for thresholded GR sets and an increased F_1 upper bound on the task.

6.2 Parse Selection

Section 6.1 illustrated the increased level of efficiency achieved by EWG compared to the current system's method for calculating weighted GRs. This section briefly considers a parse selection algorithm using EWG that would otherwise be too inefficient to apply.

Clark and Curran (2004) determine weighted GRs directly from a packed chart using Miyao and Tsujii's (2002) dynamic programming algorithm. They outline a parse selection algorithm which maximises the expected recall of dependencies by selecting the n-best GR set with the highest average GR score based on the weights from the weighted GRs. We can apply this parse selection algorithm in two ways: either (a) re-rank the n-best GR sets based on the average weight of GRs and select the highest ranking set, or (b) apply a simple variant of the Viterbi algorithm to select the GR set with the highest average weighted score over the data structure built during EWG. The latter approach, based on the parse selection algorithm in Clark and Curran (2004), takes into account *all possible parses* and effectively re-ranks all parses using weights output by EWG. These approaches will be referred to as *RE-RANK* (over the top 1000 parses) and *BEST-AVG*, respectively.

The GR set corresponding to the system's top parse achieves an F_1 of 71.24%. By applying BEST-AVG and RE-RANK parse selection, we achieve a relative error reduction of 3.01% and 0.90%, respectively. Therefore, BEST-AVG achieves higher accuracy and is more efficient than RE-RANK. It is also worth noting that these parse selection schemes are able to output a *consistent* set of GRs unlike the set corresponding to high precision GR output.

7 Conclusions

We have described a dynamic programming approach based on the Inside Outside Algorithm for producing weighted grammatical relation output directly from a unification-based parse forest. In an evaluation on a standard test suite the approach achieves substantial improvements in accuracy and parser throughput over a previous implementation. The approach is novel as it allows multiple heads (and inside probabilities) per parse forest node instead of manipulating the parse forest so that each node represents only a single head.

We intend to extend this work to develop more sophisticated parse selection schemes based on weighted GR output. Re-ranking the n-best GR sets results in a consistent but not necessarily a coherent set of GRs. Given the increased upper bound on precision for the high precision GR output, we hope to boost the corresponding recall measure by determining a consistent and coherent set of GRs *from the weighted GR set*.

Acknowledgements

This work is in part funded by the Overseas Research Students Awards Scheme and the Poynton Scholarship appointed by the Cambridge Australia Trust in collaboration with the Cambridge Commonwealth Trust. We would like to thank four anonymous reviewers who provided many useful suggestions for improvement.

References

J. K. Baker. 1979. Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America*, pages 557–550.

Ted Briscoe and John Carroll. 1995. Developing and evaluating a probabilistic LR parser of part-of-speech and punctuation labels. In *Proceedings of the ACL/SIGPARSE 4th International Workshop on Parsing Technologies*, pages 48–58, Prague / Karlovy Vary, Czech Republic.

Ted Briscoe and John Carroll. 2002. Robust accurate statistical annotation of general text. In *Proceedings of the Conference on Language Resources and Evaluation (LREC 2002)*, pages 1499–1504, Palmas, Canary Islands, May.

Ted Briscoe and John Carroll. 2005. Evaluating the speed and accuracy of an unlexicalized statistical parser on the PARC Depbank. Under review.

John Carroll and Ted Briscoe. 2002. High precision extraction of grammatical relations. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

John Carroll, Ted Briscoe, and Antonio Sanfi lippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 447–454, Granada.

John Carroll. 1993. *Practical unifi cation-based parsing of natural language*. Ph.D. thesis, Computer Laboratory, University of Cambridge. Technical Report No. 314.

Stephen Clark and James Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Barcelona, Spain.

Ann Copestake. 2003. Report on the design of RMRS. DeepThought Project Deliverable D1.1a, University of Cambridge, UK.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unifi cation-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA.

Mark Johnson. 2001. Joint and conditional estimation of tagging and parsing models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, Toulouse, France, July.

Ronald Kaplan, Stephen Riezler, Tracy King, John Maxwell, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology conference / North American chapter of the Association for Computational Linguistics annual meeting*, pages 97–113, Boston, Massachusetts, May.

Tracy King, Richard Crouch, Stephen Riezler, Mary Dalrymple, and Ronald Kaplan. 2003. The PARC700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*.

Karim Lari and Steve Young. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 2(4):35–56.

Dekang Lin. 1998. Dependency-based evaluation of MINIPAR. In *Proceedings of the Workshop on The Evaluation of Parsing Systems at the 1st International Conference on Language Resources and Evaluation*, Granada, Spain.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*, San Diego, California, March.

Stephan Oepen and John Carroll. 2000. Ambiguity packing in constraint-based parsing - practical results. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 162–169, Seattle, WA.

Geoffrey Sampson. 1995. *English for the Computer*. Oxford University Press.

Helmut Schmid and Mats Rooth. 2001. Parse forest computation of expected governors. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 458–465.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Improving Parsing Accuracy by Combining Diverse Dependency Parsers

Daniel Zeman and **Zdeněk Žabokrtský**
Ústav formální a aplikované lingvistiky, Univerzita Karlova
Malostranské náměstí 25, CZ-11800 Praha
{zeman|zabokrtsky}@ufal.mff.cuni.cz

Abstract

This paper explores the possibilities of improving parsing results by combining outputs of several parsers. To some extent, we are porting the ideas of Henderson and Brill (1999) to the world of dependency structures. We differ from them in exploring context features more deeply. All our experiments were conducted on Czech but the method is language-independent. We were able to significantly improve over the best parsing result for the given setting, known so far. Moreover, our experiments show that even parsers far below the state of the art can contribute to the total improvement.

1 Introduction

Difficult and important NLP problems have the property of attracting whole range of researchers, which often leads to the development of several different approaches to the same problem. If these approaches are independent enough in terms of not producing the same kinds of errors, there is a hope that their combination can bring further improvement to the field. While improving any single approach gets more and more difficult once some threshold has been touched, exploring the potential of approach combination should never be omitted, provided three or more approaches are available.

Combination techniques have been successfully applied to part of speech tagging (van Halteren et al., 1998; Brill and Wu, 1998; van Halteren et al., 2001). In both cases the investigators were able to achieve significant improvements over the previous best tagging results. Similar advances have been made in machine translation (Frederking and Nirenburg, 1994), speech recognition (Fiscus, 1997), named entity recognition (Borthwick et al., 1998), partial parsing (Inui and Inui, 2000), word sense disambiguation (Florian and Yarowsky, 2002) and question answering (Chu-Carroll et al., 2003).

Brill and Hladká (Hajič et al., 1998) have first explored committee-based dependency parsing. However, they generated multiple parsers from a single one using bagging (Breiman, 1994). There have not been more sufficiently good parsers available. A successful application of voting and of a stacked classifier to constituent parsing followed in (Henderson and Brill, 1999). The authors have investigated two combination techniques (constituent voting and naïve Bayes), and two ways of their application to the (full) parsing: parser switching, and similarity switching. They were able to gain 1.6 constituent F-score, using their most successful technique.

In our research, we focused on dependency parsing. One of the differences against Henderson and Brill's situation is that a dependency parser has to assign exactly one governing node (parent word) to each word. Unlike the number of constituents in constituency-based frameworks, the number of dependencies is known in advance, the parser only has to assign a link (number 0 through N) to each word. In that sense, a dependency parser is similar to classifiers like POS taggers. Unless it deliberately fails to assign a parent to a word (or assigns

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 171–178,
Vancouver, October 2005.

Parser	Author	Brief description	Accuracy	
			Tune	Test
ec	Eugene Charniak	A maximum-entropy inspired parser, home in constituency-based structures. English version described in Charniak (2000), Czech adaptation 2002 – 2003, unpublished.	83.6	85.0
mc	Michael Collins	Uses a probabilistic context-free grammar, home in constituency-based structures. Described in (Hajič et al., 1998; Collins et al., 1999).	81.7	83.3
zž	Zdeněk Žabokrtský	Purely rule-based parser, rules are designed manually, just a few lexical lists are collected from the training data. 2002, unpublished.	74.3	76.2
dz	Daniel Zeman	A statistical parser directly modeling syntactic dependencies as word bigrams. Described in (Zeman, 2004).	73.8	75.5
thr	Tomáš Holan	Three parsers. Two of them use a sort of push-down automata and differ from each other only in the way they process the sentence (left-to-right or right-to-left). Described in (Holan, 2004).	71.0	72.3
thl			69.5	70.3
thp			62.0	63.5

Table 1. A brief description of the tested parsers. Note that the Tune data is *not* the data used to train the individual parsers. Higher numbers in the right column reflect just the fact that the Test part is slightly easier to parse.

several alternate parents to a word), there is no need for precision & recall. Instead, a single metric called accuracy is used.

On the other hand, a dependency parser is not a *real* classifier: the number of its "classes" is theoretically unlimited (natural numbers), and no generalization can be drawn about objects belonging to the same "class" (words that – sometimes – appeared to find their parent at the position *i*).

A combination of dependency parsers does not necessarily grant the resulting dependency structure being cycle-free. (This contrasts to not introducing crossing brackets in constituent parsing, which is granted according to Henderson and Brill.) We address the issue in 4.4.

The rest of this paper is organized as follows: in Sections 2 and 3 we introduce the data and the component parsers, respectively. In Section 4 we discuss several combining techniques, and in Section 5 we describe the results of the corresponding experiments. We finally compare our results to the previous work and conclude.

2 The data

To test our parser combination techniques, we use the Prague Dependency Treebank 1.0 (PDT; Hajič et al. 2001). All the individual parsers have been trained on its analytical-level training section (73,088 sentences; 1,255,590 tokens).

The PDT analytical d-test section has been partitioned into two data sets, Tune (last 77 files; 3646 sentences; 63,353 tokens) and Test (first 76 files; 3673 sentences; 62,677 tokens). We used the Tune set to train the combining classifiers if needed. The Test data were used to evaluate the approach. Neither the member parsers, nor the combining classifier have seen this data set during their respective learning runs.

3 Component parsers

The parsers involved in our experiments are summarized in Table 1. Most of them use unique strategies, the exception being *thl* and *thr*, which differ only in the direction in which they process the sentence.

The table also shows individual parser accuracies on our Test data. There are two state-of-the art parsers, four not-so-good parsers, and one quite poor parser. We included the two best parsers (ec+mc) in all our experiments, and tested the contributions of various selections from the rest.

The necessary assumption for a meaningful combination is that the outputs of the individual parsers are sufficiently uncorrelated, i.e. that the parsers do not produce the same errors. If some

Parsers compared		All 7	4 best	3 best	ec+mc+dz	2 best	3 worst
Who is correct		**How many times correct**					
a single parser (all other wrong)	ec	1.7 %	3.0 %	4.1 %	4.5 %	8.1 %	
	zž	1.2 %	2.0 %	3.3 %			
	mc	0.9 %	1.7 %	2.7 %	2.9 %	6.2 %	
	thr	0.4 %					4.9 %
	thp	0.4 %					4.4 %
	dz	0.3 %	1.0 %		2.2 %		
	thl	0.3 %					4.3 %
all seven parsers		42.5 %					
at least six		58.1 %					
at least five		68.4 %					
at least four		76.8 %	58.0 %				
at least three		84.0 %	75.1 %	63.6 %	64.7 %		50.6 %
at least two		90.4 %		82.9 %	82.4 %	75.5 %	69.2 %
at least one		95.8 %	94.0 %	93.0 %	92.0 %	89.8 %	82.7 %

Table 2: Comparison of various groups of parsers. All percentages refer to the share of the total words in test data, attached correctly. The "single parser" part shows shares of the data where a single parser is the only one to know how to parse them. The sizes of the shares should correlate with the uniqueness of the individual parsers' strategies and with their contributions to the overall success. The "at least" rows give clues about what can be got by majority voting (if the number represents over 50 % of parsers compared) or by hypothetical oracle selection (if the number represents 50 % of the parsers or less, an oracle would generally be needed to point to the parsers that know the correct attachment).

parsers produced too similar results, there would be the danger that they push all their errors through, blocking any meaningful opinion of the other parsers.

To check the assumption, we counted (on the Tune data set) for each parser in a given parser selection the number of dependencies that only this parser finds correctly. We show the results in Table 2. They demonstrate that all parsers are independent on the others at least to some extent.

4 Combining techniques

Each dependency structure consists of a number of dependencies, one for each word in the sentence. Our goal is to tell for each word, which parser is the most likely to pick its dependency correctly. By combining the selected dependencies we aim at producing a better structure. We call the complex system (of component parsers plus the selector) the *superparser*.

Although we have shown how different strategies lead to diversity in the output of the parsers, there is little chance that any parser will be able to push through the things it specializes in. It is very difficult to realize that a parser is right if most of the others reject its proposal. Later in this section we assess this issue; however, the real power is in majority of votes.

4.1 Voting

The simplest approach is to let the member parsers vote. At least three parsers are needed. If there are exactly three, only the following situations really matter: 1) two parsers outvote the third one; 2) a tie: each parser has got a unique opinion. It would be democratic in the case of a tie to select randomly. However, that hardly makes sense once we know the accuracy of the involved parsers on the Tune set. Especially if there is such a large gap between the parsers' performance, the best parser (here ec) should get higher priority whenever there

is no clear majority of votes. Van Halteren et al. (1998) have generalized this approach for higher number of classifiers in their TotPrecision voting method. The vote of each classifier (parser) is weighted by their respective accuracy. For instance, mc + zž would outvote ec + thr, as 81.7 + 74.3 = 156 > 154.6 = 83.6 + 71.0.

4.2 Stacking

If the world were ideal, we would have an oracle, able to *always* select the right parser. In such situation our selection of parsers would grant the accuracy as high as 95.8 %. We attempt to imitate the oracle by a second-level classifier that learns from the Tune set, which parser is right in which situations. Such technique is usually called *classifier stacking*. Parallel to (van Halteren et al., 1998), we ran experiments with two stacked classifiers, Memory-Based, and Decision-Tree-Based. This approach roughly corresponds to (Henderson and Brill, 1999)'s Naïve Bayes parse hybridization.

4.3 Unbalanced combining

For applications preferring precision to recall, unbalanced combination — introduced by Brill and Hladká in (Hajič et al., 1998) — may be of interest. In this method, all dependencies proposed by at least half of the parsers are included. The term *unbalanced* reflects the fact that now precision is *not* equal to recall: some nodes lack the link to their parents. Moreover, if the number of member parsers is even, a node may get two parents.

4.4 Switching

Finally, we develop a technique that considers the whole dependency structure rather than each dependency alone. The aim is to check that the resulting structure is a tree, i.e. that the dependency-selecting procedure does not introduce cycles.[1] Henderson and Brill prove that under certain conditions, their parse hybridization approach cannot introduce crossing brackets. This might seem an analogy to our problem of introducing cycles — but unfortunately, no analogical lemma holds. As a workaround, we have investigated a crossbreed approach between Henderson and Brill's Parser Switching, and the voting methods described above. After each step, all dependencies that would introduce a cycle are banned. The algorithm is greedy — we do not try to search the space of dependency combinations for other paths. If there are no allowed dependencies for a word, the whole structure built so far is abandoned, and the structure suggested by the best component parser is used instead.[2]

5 Experiments and results

5.1 Voting

We have run several experiments where various selections of parsers were granted the voting right. In all experiments, the TotPrecision voting scheme of (van Halteren et al., 1998) has been used. The voting procedure is only very moderately affected by the Tune set (just the accuracy figures on that set are used), so we present results on both the Test and the Tune sets.

Voters	Accuracy	
	Tune	Test
ec (baseline)	83.6	85.0
all seven	84.0	85.4
ec+mc+dz	84.9	86.2
all but thp	84.9	86.3
ec+mc+zž+dz+thr	85.1	86.5
ec+mc+zž	85.2	86.7
ec+mc+zž+dz	85.6	87.0

Table 3: Results of voting experiments.

According to the results, the best voters pool consists of the two best parsers, accompanied by

[1] One may argue that "treeness" is not a necessary condition for the resulting structure, as the standard accuracy measure does not penalize non-trees in any way (other than that there is at least one bad dependency). Interestingly enough, even some of the component parsers do not produce correct trees at all times. However, non-trees are both linguistically and technically problematic, and it is good to know how far we can get with the condition in force.

[2] We have not encountered such situation in our test data. However, it indeed is possible, even if all the component parsers deliver correct trees, as can be seen from the following example. Assume we have a sentence #ABCD and parsers P1 (85 votes), P2 (83 votes), P3 (76 votes). P1 suggests the tree A→D→B→C→#, P2 suggests B→D→A→C→#, P3 suggests B→D→A→#, C→#. Then the superparser P gradually introduces the following dependencies: 1. A→D; 2. B→D; 3. C→#; 4. D→A or D→B possible but both lead to a cycle.

the two average parsers. The table also suggests that number of diverse strategies is more important than keeping high quality standard with all the parsers. Apart from the worst parser, all the other together do better than just the first two and the fourth. (On the other hand, the first three parsers are much harder to beat, apparently due to the extreme distance of the strategy of zž parser from all the others.)

Even the worst performing parser combination (all seven parsers) is significantly[3] better than the best component parser alone.

We also investigated some hand-invented voting schemes but no one we found performed better than the ec+mc+zž+dz combination above.

Some illustrative results are given in the Table 4. Votes were not weighted by accuracy in these experiments, but accuracy is reflected in the priority given to ec and mc by the human scheme inventor.

Voters	Selection scheme	Accuracy	
		Tune	Test
all seven	most votes or ec	82.8	84.3
all seven	at least half, or ec if there is no absolute majority	84.4	85.8
all seven	absolute majority, or ec+2, or mc+2, or ec	84.6	85.9

Table 4: Voting under hand-invented schemes.

5.2 Stacking – using context

We explored several ways of using context in pools of three parsers.[4] If we had only three parsers we could use context to detect two kinds of situations:

1. Each parser has its own proposal and a parser other than ec shall win.
2. Two parsers agree on a common proposal but even so the third one should win. Most likely the only reasonable instance is that ec wins over mc + the third one.

"Context" can be represented by a number of features, starting at morphological tags and ending up at complex queries on structural descriptions. We tried a simple memory-based approach, and a more complex approach based on decision trees.

Within the memory-based approach, we use just the core features the individual parsers themselves train on: the POS tags (morphological tags or m-tags in PDT terminology). We consider the m-tag of the dependent node, and the m-tags of the governors proposed by the individual parsers.

We learn the context-based strengths and weaknesses of the individual parsers on their performance on the Tune data set. In the following table, there are some examples of contexts in which ec is better than the common opinion of mc + dz.

Dep. tag	Gov. tag (ec)	Context occurrences	No. of times ec was right	Percent cases ec was right
J^	#	67	44	65.7
Vp	J^	53	28	52.8
VB	J^	46	26	56.5
N1	Z,	38	21	55.3
Rv	Vp	25	13	52.0
Z,	Z,	15	8	53.3
A1	N1	15	8	53.3
Vje	J^	14	9	64.3
N4	Vf	12	9	75.0

Table 5: Contexts where ec is better than mc+dz. J^ are coordination conjunctions, # is the root, V* are verbs, N*n* are nouns in case *n*, R* are prepositions, Z* are punctuation marks, A*n* are adjectives.

For the experiment with decision trees, we used the C5 software package, a commercial version of the well-known C4.5 tool (Quinlan, 1993). We considered the following features:

For each of the four nodes involved (the dependent and the three governors suggested by the three component parsers):

[3] All significance claims refer to the Wilcoxon Signed Ranks Test at the level of $p = 0.001$.

[4] Similar experiments could be (and have been) run for sets of more parsers as well. However, the number of possible features is much higher and the data sparser. We were not able to gain more accuracy on context-sensitive combination of more parsers.

```
agreezzmc = yes: zz (3041/1058)
agreezzmc = no:
:...agreemcec = yes: ec (7785/1026)
    agreemcec = no:
    :...agreezzec = yes: ec (2840/601)
        agreezzec = no:
        :...zz_case = 6: zz (150/54)
            zz_case = 3: zz (34/10)
            zz_case = X: zz (37/20)
            zz_case = undef: ec (2006/1102)
            zz_case = 7: zz (83/48)
            zz_case = 2: zz (182/110)
            zz_case = 4: zz (108/57)
            zz_case = 1: ec (234/109)
            zz_case = 5: mc (1)
            zz_case = root:
            :...ec_negat = A: mc (117/65)
                ec_negat = undef: ec (139/65)
                ec_negat = N: ec (1)
                ec_negat = root: ec (2)
```

Figure 1. The decision tree for ec+mc+zž, learned by C5. Besides pairwise agreement between the parsers, only morphological case and negativeness matter.

- 12 attributes derived from the morphological tag (part of speech, subcategory, gender, number, case, inner gender, inner number, person, degree of comparison, negativeness, tense and voice)
- 4 semantic attributes (such as ProperName, Geography etc.)

For each of the three governor-dependent pairs involved:

- mutual position of the two nodes (LeftNeighbor, RightNeighbor, LeftFar, RightFar)
- mutual position expressed numerically
- for each parser pair a binary flag whether they do or do not share opinions

The decision tree was trained only on situations where at least one of the three parsers was right *and* at least one was wrong.

Voters	Scheme	Accuracy
ec+mc+dz	context free	86.2
ec+mc+dz	memory-based	86.3
ec+mc+zž	context free	86.7
ec+mc+zž	decision tree	86.9

Table 6: Context-sensitive voting. Contexts trained on the Tune data set, accuracy figures apply to the Test data set. Context-free results are given for the sake of comparison.

It turns out that there is very low potential in the context to improve the accuracy (the improvement is significant, though). The behavior of the parsers is too noisy as to the possibility of formulating some rules for prediction, when a particular parser is right. C5 alone provided a supporting evidence for that hypothesis, as it selected a very simple tree from all the features, just 5 levels deep (see Figure 1).

Henderson and Brill (1999) also reported that context did not help them to outperform simple voting. Although it is risky to generalize these observations for other treebanks and parsers, our environment is quite different from that of Henderson and Brill, so the similarity of the two observations is at least suspicious.

5.3 Unbalanced combining

Finally we compare the balanced and unbalanced methods. Expectedly, precision of the unbalanced combination of odd number of parsers rose while recall dropped slightly. A different situation is observed if even number of parsers vote and more than one parent can be selected for a node. In such case, precision drops in favor of recall.

Method	Precision	Recall	F-measure
ec only (baseline)	85.0		
balanced (all seven)	85.4		
unbalanced (all seven)	**90.7**	78.6	84.2
balanced (best four)	**87.0**		
unbalanced (best four)	85.4	**87.7**	86.5
balanced (ec+mc+dz)	86.2		
unbalanced	89.5	84.0	86.7

Method	Precision	Recall	F-measure
(ec+mc+dz)			
balanced (ec+mc+zž)	86.7		
unbalanced (ec+mc+zž)	90.2	84.7	**87.3**

Table 7: Unbalanced vs. balanced combining. All runs ignored the context. Evaluated on the Test data set.

5.4 Switching

Out of the 3,673 sentences in our Test set, 91.6 % have been rendered as correct trees in the balanced decision-tree based stacking of ec+mc+zž+dz (our best method).

After we banned cycles, the accuracy dropped from 97.0 to 96.9 %.

6 Comparison to related work

Brill and Hladká in (Hajič et al., 1998) were able to improve the original accuracy of the mc parser on PDT 0.5 e-test data from 79.1 to 79.9 (a nearly 4% reduction of the error rate). Their unbalanced[5] voting pushed the F-measure from 79.1 to 80.4 (6% error reduction). We pushed the balanced accuracy of the ec parser from 85.0 to 87.0 (13% error reduction), and the unbalanced F-measure from 85.0 to 87.7 (18% reduction). Note however that there were different data and component parsers (Hajič et al. found bagging the best parser better than combining it with other that-time-available parsers). This is the first time that several strategically different dependency parsers have been combined.

(Henderson and Brill, 1999) improved their best parser's F-measure of 89.7 to 91.3, using their naïve Bayes voting on the Penn TreeBank constituent structures (16% error reduction). Here, even the framework is different, as has been explained above.

7 Conclusion

We have tested several approaches to combining of dependency parsers. Accuracy-aware voting of the four best parsers turned out to be the best method, as it significantly improved the accuracy of the best component from 85.0 to 87.0 % (13 % error rate reduction). The unbalanced voting lead to the precision as high as 90.2 %, while the F-measure of 87.3 % outperforms the best result of balanced voting (87.0).

At the same time, we found that employing context to this task is very difficult even with a well-known and widely used machine-learning approach.

The methods are language independent, though the amount of accuracy improvement may vary according to the performance of the available parsers.

Although voting methods are themselves not new, as far as we know we are the first to propose and evaluate their usage in full dependency parsing.

8 Acknowledgements

Our thanks go to the creators of the parsers used here for making their systems available.

The research has been supported by the Czech Academy of Sciences, the "Information Society" program, project No. 1ET101470416.

References

Andrew Borthwick, John Sterling, Eugene Agichtein, Ralph Grishman. 1998. *Exploiting Diverse Knowledge Sources via Maximum Entropy in Named Entity Recognition.* In: Eugene Charniak (ed.): Proceedings of the 6th Workshop on Very Large Corpora, pp. 152–160. Université de Montréal, Montréal, Québec.

Leo Breiman. 1994. *Bagging Predictors.* Technical Report 421, Department of Statistics, University of California at Berkeley, Berkeley, California.

Eric Brill, Jun Wu. 1998. *Classifier Combination for Improved Lexical Combination.* In: Proceedings of the 17th International Conference on Computational Linguistics (COLING-98), pp. 191–195. Université de Montréal, Montréal, Québec.

Eugene Charniak. 2000. *A Maximum-Entropy-Inspired Parser*. In: Proceedings of NAACL. Seattle, Washington.

Jennifer Chu-Carroll, Krzysztof Czuba, John Prager, Abraham Ittycheriah. 2003. *In Question Answering, Two Heads Are Better Than One.* In: Proceedings of the HLT-NAACL. Edmonton, Alberta.

Michael Collins, Jan Hajič, Eric Brill, Lance Ramshaw, Christoph Tillmann. 1999. *A Statistical Parser of Czech.* In: Proceedings of the 37th Meeting of the

[5] Also alternatively called *unrestricted*.

ACL, pp. 505–512. University of Maryland, College Park, Maryland.

Jonathan G. Fiscus. 1997. *A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER).* In: EuroSpeech 1997 Proceedings, vol. 4, pp. 1895–1898. Rodos, Greece.

Radu Florian, David Yarowsky. 2002. *Modeling Consensus: Classifier Combination for Word Sense Disambiguation.* In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 25–32. Philadelphia, Pennsylvania.

Robert Frederking, Sergei Nirenburg. 1994. *Three Heads Are Better Than One.* In: Proceedings of the 4th Conference on Applied Natural Language Processing, pp. 95–100. Stuttgart, Germany.

Jan Hajič, Eric Brill, Michael Collins, Barbora Hladká, Douglas Jones, Cynthia Kuo, Lance Ramshaw, Oren Schwartz, Christoph Tillmann, Daniel Zeman. 1998. *Core Natural Language Processing Technology Applicable to Multiple Languages. The Workshop 98 Final Report.* http://www.clsp.jhu.edu/ws98/projects/nlp/report/. Johns Hopkins University, Baltimore, Maryland.

Jan Hajič, Barbora Vidová Hladká, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas. 2001. *Prague Dependency Treebank 1.0 CD-ROM.* Catalog # LDC2001T10, ISBN 1-58563-212-0. Linguistic Data Consortium, Philadelphia, Pennsylvania.

Hans van Halteren, Jakub Zavřel, Walter Daelemans. 1998. *Improving Data-Driven Wordclass Tagging by System Combination.* In: Proceedings of the 17th International Conference on Computational Linguistics (COLING-98), pp. 491–497. Université de Montréal, Montréal, Québec.

Hans van Halteren, Jakub Zavřel, Walter Daelemans. 2001. *Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems.* In: Computational Linguistics, vol. 27, no. 2, pp. 199–229. MIT Press, Cambridge, Massachusetts.

John C. Henderson, Eric Brill. 1999. *Exploiting Diversity in Natural Language Processing: Combining Parsers.* In: Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP-99), pp. 187–194. College Park, Maryland.

Tomáš Holan. 2004. *Tvorba závislostního syntaktického analyzátoru.* In: David Obdržálek, Jana Tesková (eds.): MIS 2004 Josefův Důl, Sborník semináře. Matfyzpress, Praha, Czechia.

Inui Takashi, Inui Kentaro. 2000. *Committee-Based Decision Making in Probabilistic Partial Parsing.* In: Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000), pp. 348–354. Universität des Saarlandes, Saarbrücken, Germany.

J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California.

Daniel Zeman. 2004. *Parsing with a Statistical Dependency Model (PhD thesis).* Univerzita Karlova, Praha, Czechia.

Exploring Features for Identifying Edited Regions in Disfluent Sentences

Qi Zhang
Department of Computer Science
Fudan University
Shanghai, P.R.China 200433
qi_zhang@fudan.edu.cn

Fuliang Weng
Research and Technology Center
Robert Bosch Corp.
Palo Alto, CA 94304
fuliang.weng@rtc.bosch.com

Abstract

This paper describes our effort on the task of edited region identification for parsing disfluent sentences in the Switchboard corpus. We focus our attention on exploring feature spaces and selecting good features and start with analyzing the distributions of the edited regions and their components in the targeted corpus. We explore new feature spaces of a part-of-speech (POS) hierarchy and relaxed for rough copy in the experiments. These steps result in an improvement of 43.98% percent relative error reduction in F-score over an earlier best result in edited detection when punctuation is included in both training and testing data [Charniak and Johnson 2001], and 20.44% percent relative error reduction in F-score over the latest best result where punctuation is excluded from the training and testing data [Johnson and Charniak 2004].

1 Introduction

Repairs, hesitations, and restarts are common in spoken language, and understanding spoken language requires accurate methods for identifying such disfluent phenomena. Processing speech repairs properly poses a challenge to spoken dialog systems. Early work in this field is primarily based on small and proprietary corpora, which makes the comparison of the proposed methods difficult [Young and Matessa 1991, Bear et al. 1992, Heeman & Allen 1994]. Because of the availability of the Switchboard corpus [Godfrey et al. 1992] and other conversational telephone speech (CTS) corpora, there has been an increasing interest in improving the performance of identifying the edited regions for parsing disfluent sentences [Charniak and Johnson 2001, Johnson and Charniak 2004, Ostendorf et al. 2004, Liu et al. 2005].

In this paper we describe our effort towards the task of edited region identification with the intention of parsing disfluent sentences in the Switchboard corpus. A clear benefit of having accurate edited regions for parsing has been demonstrated by a concurrent effort on parsing conversational speech [Kahn et al 2005]. Since different machine learning methods provide similar performances on many NLP tasks, in this paper, we focus our attention on exploring feature spaces and selecting good features for identifying edited regions. We start by analyzing the distributions of the edited regions and their components in the targeted corpus. We then design several feature spaces to cover the disfluent regions in the training data. In addition, we also explore new feature spaces of a part-of-speech hierarchy and extend candidate pools in the experiments. These steps result in a significant improvement in F-score over the earlier best result reported in [Charniak and Johnson 2001], where punctuation is included in both the training and testing data of the Switchboard corpus, and a significant error reduction in F-score over the latest best result [Johnson and Charniak 2004], where punctuation is ignored in both the training and testing data of the Switchboard corpus.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 179–185,
Vancouver, October 2005.

In this paper, we follow the definition of [Shriberg 1994] and others for speech repairs: A speech repair is divided into three parts: the *reparandum*, the part that is repaired; the *interregnum*, the part that can be either empty or fillers; and the *repair/repeat*, the part that replaces or repeats the reparandum. The definition can also be exemplified via the following utterance:

$$\underbrace{\textit{This is}}_{\text{reparanda}},\ \underbrace{\textit{you know}}_{\text{interregnum}},\ \underbrace{\text{this is}}_{\text{repeat}}\ \text{a big problem.}$$

This paper is organized as follows. In section 2, we examine the distributions of the editing regions in Switchboard data. Section 3, then, presents the Boosting method, the baseline system and the feature spaces we want to explore. Section 4 describes, step by step, a set of experiments that lead to a large performance improvement. Section 5 concludes with discussion and future work.

2 Repair Distributions in Switchboard

We start by analyzing the speech repairs in the Switchboard corpus. Switchboard has over one million words, with telephone conversations on prescribed topics [Godfrey et al. 1992]. It is full of disfluent utterances, and [Shriberg 1994, Shriberg 1996] gives a thorough analysis and categorization of them. [Engel et al. 2002] also showed detailed distributions of the interregnum, including interjections and parentheticals. Since the majority of the disfluencies involve all the three parts (reparandum, interregnum, and repair/repeat), the distributions of all three parts will be very helpful in constructing patterns that are used to identify edited regions.

For the reparandum and repair types, we include their distributions with and without punctuation. We include the distributions with punctuation is to match with the baseline system reported in [Charniak and Johnson 2001], where punctuation is included to identify the edited regions. Resent research showed that certain punctuation/prosody marks can be produced when speech signals are available [Liu et al. 2003]. The interregnum type, by definition, does not include punctuation.

The length distributions of the reparanda in the training part of the Switchboard data with and without punctuation are given in Fig. 1. The reparanda with lengths of less than 7 words make up 95.98% of such edited regions in the training data. When we remove the punctuation marks, those with lengths of less than 6 words reach roughly 96%. Thus, the patterns that consider only reparanda of length 6 or less will have very good coverage.

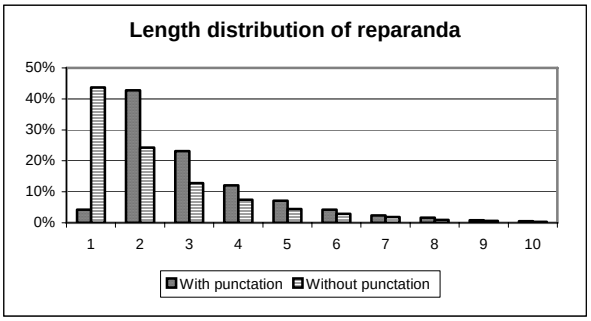


Figure 1. Length distribution of reparanda in Switchboard training data.

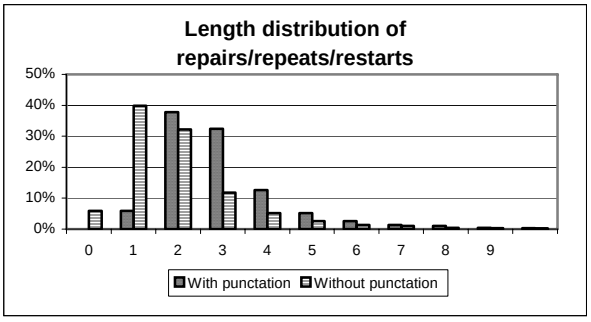


Figure 2. Length distribution of repairs/repeats/restarts in Switchboard training data.

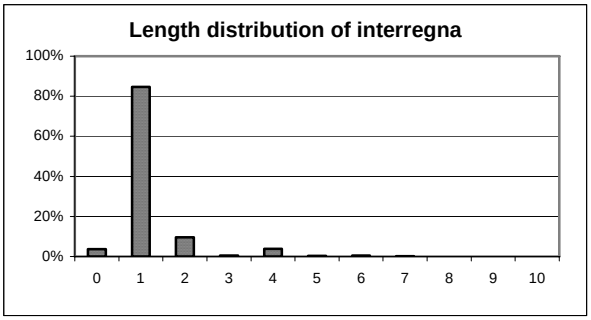


Figure 3. Length distribution of interregna in Switchboard training data.

The two repair/repeat part distributions in the training part of the Switchboard are given in Fig. 2. The repairs/repeats with lengths less than 7 words

make 98.86% of such instances in the training data. This gives us an excellent coverage if we use 7 as the threshold for constructing repair/repeat patterns.

The length distribution of the interregna of the training part of the Switchboard corpus is shown in Fig. 3. We see that the overwhelming majority has the length of one, which are mostly words such as "uh", "yeah", or "uh-huh".

In examining the Switchboard data, we noticed that a large number of reparanda and repair/repeat pairs differ on less than two words, i.e. "*as to*, you know, when to"[1], and the amount of the pairs differing on less than two POS tags is even bigger. There are also cases where some of the pairs have different lengths. These findings provide a good base for our feature space.

3 Feature Space Selection for Boosting

We take as our baseline system the work by [Charniak and Johnson 2001]. In their approach, *rough copy* is defined to produce candidates for any potential pairs of reparanda and repairs. A boosting algorithm [Schapire and Singer 1999] is used to detect whether a word is edited. A total of 18 variables are used in the algorithm. In the rest of the section, we first briefly introduce the boosting algorithm, then describe the method used in [Charniak and Johnson 2001], and finally we contrast our improvements with the baseline system.

3.1 Boosting Algorithm

Intuitively, the boosting algorithm is to combine a set of simple learners iteratively based on their classification results on a set of training data. Different parts of the training data are scaled at each iteration so that the parts of the data previous classifiers performed poorly on are weighted higher. The weighting factors of the learners are adjusted accordingly.

We re-implement the boosting algorithm reported by [Charniak and Johnson 2001] as our baseline system in order to clearly identify contributing factors in performance. Each word token is characterized by a finite tuple of random variables

$$(Y, X_1, ..., X_m).$$

Y is the *conditioned variables* and ranges from {-1,+1}, with Y = +1 indicating that the word is edited. $X_1, ..., X_m$ are the *conditioning variables;* each variable X_j ranges over a finite set χ_j. The goal of the classifer is to predict the value of Y given a value for $X_1, ..., X_m$.

A boosting classifier is a linear combination of n features to define the *prediction variable Z.*

$$Z = \sum_{i=1}^{n} \alpha_i F_i \qquad (1)$$

where α_i is the weight to be estimated for feature ϕ_i. ϕ_i is a set of variable-value pairs, and each F_i has the form of:

$$F_i = \prod_{<X_j, x_j> \in \phi_i} (X_j = x_j) \qquad (2)$$

with X's being conditioning variables and x's being values.

Each component in the production for F_i is defined as:

$$(X_j = x_j) = \begin{cases} 1 & < X_j = x_j > \in \phi_i \\ 0 & otherwise \end{cases} \qquad (3)$$

In other words, F_i is 1 if and only if all the variable-value pairs for the current position belong to ϕ_i.

The prediction made by the classifier is $\text{sign}(Z) = Z/|Z|$. Intuitively, our goal is to adjust the vector of feature weights $\vec{\alpha} = (\alpha_1, ..., \alpha_n)$ to minimize the expected *misclassification rate* $E[\text{sign}(Z) \neq Y]$. This function is difficult to minimize, so our boosting classifier minimizes the expected *boost loss* $\hat{E}_t[(\exp(-YZ)]$ as in [Collins 2000], where $\hat{E}_t[\cdot]$ is the expectation on the empirical training corpus distribution. In our implementation, each learner contains only one variable. The feature weights are adjusted iteratively, one weight per iteration. At each iteration, it reduces the boost loss on the training corpus. In our experiments, $\vec{\alpha}$ is obtained after

[1] "*as to*" is the edited region. Italicized words in the examples are edited words

1500 iterations, and contains around 1350 non-zero feature weights.

3.2 Charniak-Johnson approach

In [Charniak and Johnson 2001], identifying edited regions is considered as a classification problem, where each word is classified either as edited or normal. The approach takes two steps. The first step is to find *rough copy*. Then, a number of variables are extracted for the boosting algorithm. In particular, a total of 18 different conditioning variables are used to predict whether the current word is an edited word or a non-edited word. The 18 different variables listed in Table 1 correspond to the 18 different dimensions/factors for the current word position. Among the 18 variables, six of them, N_m, N_u, N_i, N_l, N_r *and* T_f, depend on the identification of a *rough copy*.

For convenience, their definition of a *rough copy* is repeated here. A rough copy in a string of tagged words has the form of $\partial_1 \beta \lambda \partial_2$, where:

1. ∂_1 (the source) and ∂_2 (the copy) both begin with non-punctuation,
2. the strings of non-punctuation POS tag of ∂_1 and ∂_2 are identical,
3. β (the free final) consists of zero or more sequences of a free final word (see below) followed by optional punctuation,
4. λ (the interregnum) consists of sequences of an interregnum string (see below) followed by optional punctuation.

The set of *free final words* includes all partial words and a small set of conjunctions, adverbs and miscellanea. The set of *interregnum strings* consists of a small set of expressions such as *uh, you know, I guess, I mean,* etc.

3.3 New Improvements

Our improvements to the Charniak-Johnson method can be classified into three categories with the first two corresponding to the twp steps in their method. The three categories of improvements are described in details in the following subsections.

3.3.1 Relaxing Rough Copy

We relax the definition for rough copy, because more than 94% of all edits have both reparandum and repair, while the rough copy defined in [Charniak and Johnson 2001] only covers 77.66% of such instances.

Two methods are used to relax the rough copy definition. The first one is to adopt a hierarchical POS tag set: all the Switchboard POS tags are further classified into four major categories: N (noun related), V (verb related), Adj (noun modifiers), Adv (verb modifiers). Instead of requiring the exact match of two POS tag sequences, we also consider two sequences as a

Variables	Name	Short description
X_1	W_0	The current orthographic word.
$X_2 - X_5$	P_0,P_1,P_2,P_f	Partial word flags for the current position, the next two to the right, and the first one in a sequence of free-final words (partial, conjunctions, etc.) to the right of the current position.
$X_6 - X_{10}$	T_{-1},T_0,T_1,T_2,T_f	Part of speech tags for the left position, the current position, the next two positions to the right, and the first free-final word position to the right of the current position.
X_{11}	N_m	Number of words in common in reparandum and repair
X_{12}	N_n	Number of words in reparandum but not repair
X_{13}	N_i	Number of words in interregnum
X_{14}	N_l	Number of words to the left edge of reparandum
X_{15}	N_r	Number of words to the right edge of reparandum
X_{16}	C_t	The first non-punctuation tag to the right of the current position
X_{17}	C_w	The first non-punctuation word to the right of the current position
X_{18}	T_i	The tag of the first word right after the interregnum that is right after the current word.

Table 1. Descriptions of the 18 conditioning variables from [Charniak and Johnson 2001]

rough copy if their corresponding major categories match. This relaxation increases the *rough copy coverage*, (the percent of words in edited regions found through the definition of rough copy), from 77.66% to 79.68%.

The second is to allow one mismatch in the two POS sequences. The mismatches can be an addition, deletion, or substitution. This relaxation improves the coverage from 77.66% to 85.45%. Subsequently, the combination of the two relaxations leads to a significantly higher coverage of 87.70%. Additional relaxation leads to excessive candidates and worse performance in the development set.

3.3.2 Adding New Features

We also include new features in the feature set: one is the shortest *distance* (the number of words) between the current word and a word of the same orthographic form to the right, if that repeated word exists; another is the words around the current position. Based on the distributional analysis in section 2, we also increase the *window sizes* for POS tags ($T_{-5},...,T_5$) and words ($W_{-5},...,W_5$) to ±5 and partial words ($P_{-3},...,P_3$) to ±3, extending T_i and P_j.

3.3.3 Post Processing Step

In addition to the two categories, we try to use contextual patterns to address the independency of variables in the features. The patterns have been extracted from development and training data, to deal with certain sequence-related errors, e.g.,

E N E → E E E,

which means that if the neighbors on both sides of a word are classified into EDITED, it should be classified into EDITED as well.

4 Experimental Results

We conducted a number of experiments to test the effectiveness of our feature space exploration. Since the original code from [Charniak and Johnson 2001] is not available, we conducted our first experiment to replicate the result of their baseline system described in section 3. We used the exactly same training and testing data from the Switchboard corpus as in [Charniak and Johnson 2001]. The training subset consists of all files in the sections 2 and 3 of the Switchboard corpus. Section 4 is split into three approximately equal size subsets. The first of the three, i.e., files sw4004.mrg to sw4153.mrg, is the testing corpus. The files sw4519.mrg to sw4936.mrg are the development corpus. The rest files are reserved for other purposes. When punctuation is included in both training and testing, the re-established baseline has the precision, recall, and F-score of 94.73%, 68.71% and 79.65%, respectively. These results are comparable with the results from [Charniak & Johnson 2001], i.e., 95.2%, 67.8%, and 79.2% for precision, recall, and f-score, correspondingly.

In the subsequent experiments, the set of additional feature spaces described in section 3 are added, step-by-step. The first addition includes the shortest distance to the same word and window size increases. This step gives a 2.27% improvement on F-score over the baseline. The next addition is the introduction of the POS hierarchy in finding rough copies. This also gives more than 3% absolute improvement over the baseline and 1.19% over the expanded feature set model. The addition of the feature spaces of relaxed matches for words, POS tags, and POS hierarchy tags all give additive improvements, which leads to an overall of 8.95% absolute improvement over the re-implemented baseline, or 43.98% relative error reduction on F-score.

When compared with the latest results from [Johnson and Charniak 2004], where no punctuations are used for either training or testing data, we also observe the same trend of the improved results. Our best result gives 4.15% absolute improvement over their best result, or 20.44% relative error reduction in f-scores. As a sanity check, when evaluated on the training data as a cheating experiment, we show a remarkable consistency with the results for testing data.

For error analysis, we randomly selected 100 sentences with 1673 words total from the test sentences that have at least one mistake. Errors can be divided into two types, *miss* (should be edited) and *false alarm* (should be noraml). Among the 207 misses, about 70% of them require some phrase level analysis or acoustic cues for phrases.

Method codes	Results on training data with punctuation			Results on testing data					
				Punctuation on both			No punctuation on both		
	Precision	Recall	f-score	Precision	Recall	f-score	Precision	Recall	f-score
CJ'01				95.2	67.8	79.2			
JC'04 p							82.0	77.8	79.7
R CJ'01	94.9	71.9	81.81	94.73	68.71	79.65	91.46	64.42	75.59
+d	94.56	78.37	85.71	94.47	72.31	81.92	91.79	68.13	78.21
+d+h	94.23	81.32	87.30	94.58	74.12	83.11	91.56	71.33	80.19
+d+rh	94.12	82.61	87.99	92.61	77.15	84.18	89.92	72.68	80.39
+d+rw	96.13	82.45	88.77	94.79	75.43	84.01	92.17	70.79	80.08
+d+rw+rh	94.42	84.67	89.28	94.57	77.93	85.45	92.61	73.46	81.93
+d+rw+rt+wt	94.43	84.79	89.35	94.65	76.61	84.68	92.08	72.61	81.19
+d+rw+rh+wt	94.58	85.21	89.65	94.72	79.22	86.28	92.69	75.30	83.09
+d+rw+rh+wt+ps	93.69	88.62	**91.08**	93.81	83.94	**88.60**	89.70	78.71	**83.85**

Table 2. Result summary for various feature spaces.

Method codes	Method description
CJ'01	Charniak and Johnson 2001
JC'04 p	Johnson and Charniak 2004, parser results
R CJ'01	Duplicated results for Charniak and Johnson 2001
+d	Distance + window sizes
+d+h	Distance + window sizes + POS hierarchy in rough copy
+d+rh	Distance + window sizes + relaxed POS hierarchy in rough copy
+d+rw	Distance + window sizes + relaxed word in rough copy
+d+rw+rh	Distance + window sizes + relaxed word and POS hierarchy in rough copy
+d+rw+rt+wt	Distance + window sizes + word & tag pairs + relaxed word and POS in rough copy
+d+rw+rh+wt	Distance + window sizes + word & tag pairs + relaxed word and POS hierarchy in rough copy
+d+rw+rh+wt+ps	Distance + window sizes + word & tag pairs + relaxed word and POS hierarchy in rough copy + pattern substitution

Table 3. Description of method codes used in the result table.

For example, one miss is "*because of the friends* because of many other things", an error we would have a much better chance of correct identification, if we were able to identify prepositional phrases reliably. Another example is "*most of* all my family". Since it is grammatical by itself, certain prosodic information in between "most of" and "all my family" may help the identification. [Ostendorf et al. 2004] reported that interruption point could help parsers to improve results. [Kahn et al. 2005] also showed that prosody information could help parse disfluent sentences. The second major class of the misses is certain short words that are not labeled consistently in the corpus. For example, "so", "and", and "or", when they occur in the beginning of a sentence, are sometimes labeled as edited, and sometimes just as normal. The last category of the misses, about 5.3%, contains the ones where the distances between reparanda and repairs are often more than 10 words.

Among the 95 false alarms, more than three quarters of misclassified ones are related to certain grammatical constructions. Examples include cases like, "the more … the more" and "I think I should …". These cases may be fixable if more elaborated grammar-based features are used.

5 Conclusions

This paper reports our work on identifying edited regions in the Switchboard corpus. In addition to a

distributional analysis for the edited regions, a number of feature spaces have been explored and tested to show their effectiveness. We observed a 43.98% relative error reduction on F-scores for the baseline with punctuation in both training and testing [Charniak and Johnson 2001]. Compared with the reported best result, the same approach produced a 20.44% of relative error reduction on F-scores when punctuation is ignored in training and testing data [Johnson and Charniak 2004]. The inclusion of both hierarchical POS tags and the relaxation for rough copy definition gives large additive improvements, and their combination has contributed to nearly half of the gain for the test set with punctuation and about 60% of the gain for the data without punctuation.

Future research would include the use of other features, such as prosody, and the integration of the edited region identification with parsing.

6 Acknowledgement

This work has been done while the first author is working at the Research and Technology Center of Robert Bosch Corp. The research is partly supported by the NIST ATP program. The authors would also like to express their thanks to Tess Hand-Bender for her proof-reading and Jeremy G. Kahn for many useful comments. Nevertheless, all the remaining errors are ours.

References

John Bear, John Dowding and Elizabeth Shriberg. 1992. *Integrating Multiple Knowledge Sources for Detection and Correction of Repairs in Human-Computer Dialog.* Proc. Annual Meeting of the Association for Computational Linguistics. 1992.

Charniak, Eugene and Mark Johnson. 2001. *Edit Detection and Parsing for Transcribed Speech.* Proc. of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics, pp 118-126.

Collins, M. 2000. *Discriminative reranking for natural language parsing*. Proc. ICML 2000.

Engel, Donald, Eugene Charniak, and Mark Johnson. 2002. *Parsing and Disfluency Placement.* Proc. EMNLP, pp 49-54, 2002.

Godfrey, J.J., Holliman, E.C. and McDaniel, J. *SWITCHBOARD: Telephone speech corpus for research and development*, Proc. ICASSP, pp 517-520, 1992.

Heeman, Peter, and James Allen. 1994. *Detecting and Correcting Speech Repairs.* Proc. of the annual meeting of the Association for Computational Linguistics. Las Cruces, New Mexico, pp 295-302, 1994.

Johnson, Mark, and Eugene Charniak. 2004. *A TAG-based noisy-channel model of speech repairs*. Proc. of the 42nd Annual Meeting of the Association for Computational Linguistics.

Kahn, Jeremy G., Mari Ostendorf, and Ciprian Chelba. 2004. *Parsing Conversational Speech Using Enhanced Segmentation*. Proc. of HLT-NAACL, pp 125-138, 2004.

Kahn, Jeremy G., Matthew Lease, Eugene Charniak, Mark Johnson and Mari Ostendorf 2005. *Effective Use of Prosody in Parsing Conversational Speech*. Proc. EMNLP, 2005.

Liu, Yang, Elizabeth Shriberg, Andreas Stolcke, Barbara Peskin, Jeremy Ang, Dustin Hillard, Mari Ostendorf, Marcus Tomalin, Phil Woodland, Mary Harper. 2005. *Structural Metadata Research in the EARS Program.* Proc. ICASSP, 2005.

Liu, Yang, Elizabeth Shriberg, Andreas Stolcke. 2003. *Automatic disfluency identification in conversational speech using multiple knowledge sources* Proc. Eurospeech, 2003

Ostendorf, Mari, Jeremy G. Kahn, Darby Wong, Dustin Hillard, and William McNeill. *Leveraging Structural MDE in Language Processing.* EARS RT04 Workshop, 2004.

Robert E. Schapire and Yoram Singer, 1999. *Improved Boosting Algorithms Using Confidence-rated Predictions.* Machine Learning 37(3): 297-336, 1999.

Shriberg, Elizabeth. 1994. *Preliminaries to a Theory of Speech Disfluencies.* Ph.D. Thesis. UC Berkeley,1994.

Shriberg, Elizabeth. 1996. *Disfluencies in Switchboard*. Proc. of ICSLP. 1996.

Young, S. R. and Matessa, M. (1991). *Using pragmatic and semantic knowledge to correct parsing of spoken language utterances.* Proc. Eurospeech 91, Genova, Italy.

Statistical Shallow Semantic Parsing despite Little Training Data

Rahul Bhagat
Information Sciences Institute
University of Southern California
Marina del Rey, CA, 90292, USA
rahul@isi.edu

Anton Leuski
Institute for Creative Technologies
University of Southern California
Marina del Rey, CA, 90292, USA
leuski@ict.usc.edu

Eduard Hovy
Information Sciences Institute
University of Southern California
Marina del Rey, CA, 90292, USA
hovy@isi.edu

1 Introduction and Related Work

Natural language understanding is an essential module in any dialogue system. To obtain satisfactory performance levels, a dialogue system needs a semantic parser/natural language understanding system (NLU) that produces accurate and detailed dialogue oriented semantic output. Recently, a number of semantic parsers trained using either the FrameNet (Baker et al., 1998) or the PropBank (Kingsbury et al., 2002) have been reported. Despite their reasonable performances on general tasks, these parsers do not work so well in specific domains. Also, where these general purpose parsers tend to provide case-frame structures, that include the standard core case roles (Agent, Patient, Instrument, etc.), dialogue oriented domains tend to require additional information about addressees, modality, speech acts, etc. Where general-purpose resources such as PropBank and Framenet provide invaluable training data for general case, it tends to be a problem to obtain enough training data in a specific dialogue oriented domain.

We in this paper propose and compare a number of approaches for building a statistically trained domain specific parser/NLU for a dialogue system. Our NLU is a part of Mission Rehearsal Exercise (MRE) project (Swartout et al., 2001). MRE is a large system that is being built to train experts, in which a trainee interacts with a Virtual Human using voice input. The purpose of our NLU is to convert the sentence strings produced by the speech recognizer into internal shallow semantic frames composed of slot-value pairs, for the dialogue module.

2 Parsing Methods

2.1 Voting Model

We use a simple conditional probability model $P(f \mid W)$ for parsing. The model represents the probability of producing slot-value pair f as an output given that we have seen a particular word or n-gram W as input. Our two-stage procedure for generating a frame for a given input sentence is: (1) Find a set of all slot-value that correspond with each word/ngram (2) Select the top portion of these candidates to form the final frame (Bhagat et al., 2005; Feng and Hovy, 2003).

2.2 Maximum Entropy

Our next approach is the Maximum Entropy (Berger et al., 1996) classification approach. Here, we cast our problem as a problem of ranking using a classifier where each slot-value pair in the training data is considered a class and feature set consists of the unigrams, bigrams and trigrams in the sentences (Bhagat et al., 2005).

2.3 Support Vector Machines

We use another commonly used classifier, Support Vector Machine (Burges, 1998), to perform the same task (Bhagat et al., 2005). Approach is similar to Section 2.2.

2.4 Language Model

As a fourth approach to the problem, we use the Statistical Language Model (Ponte and Croft, 1997). We estimate the language model for the slot-value pairs, then we construct our target interpretation as

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 186–187,
Vancouver, October 2005.

Method	*Precison*	*Recall*	*F-score*
$Voting$	0.82	0.78	0.80
ME	0.77	0.80	0.78
SVM	0.79	0.72	0.75
$LM1$	0.80	0.84	0.82
$LM2$	0.82	0.84	0.83

Table 1: Performance of different systems on test data.

a set of the most likely slot-value pairs. We use unigram-based and trigram-based language models (Bhagat et al., 2005).

3 Experiments and Results

We train all our systems on a training set of 477 sentence-frame pairs. The systems are then tested on an unseen test set of 50 sentences. For the test sentences, the system generated frames are compared against the manually built gold standard frames, and Precision, Recall and F-scores are calculated for each frame.

Table 1 shows the average Precision, Recall and F-scores of the different systems for the 50 test sentences: Voting based (Voting), Maximum Entropy based (ME), Support Vector Machine based (SVM), Language Model based with unigrams (LM1) and Language Model based with trigrams (LM2). The F-scores show that the LM2 system performs the best though the system scores in general for all the systems are very close. To test the statistical significance of these scores, we conduct a two-tailed paired Student's t test (Manning and Schtze, 1999) on the F-scores of these systems for the 50 test cases. The test shows that there is no statistically significant difference in their performances.

4 Conclusions

This work illustrates that one can achieve fair success in building a statistical NLU engine for a restricted domain using relatively little training data and surprisingly using a rather simple voting model. The consistently good results obtained from all the systems on the task clearly indicate the feasibility of using using only word/ngram level features for parsing.

5 Future Work

Having successfully met the initial challenge of building a statistical NLU with limited training data, we have identified multiple avenues for further exploration. Firstly, we wish to build an hybrid system that will combine the strengths of all the systems to produce a much more accurate system. Secondly, we wish to see the effect that ASR output has on each of the systems. We want to test the robustness of systems against an increase in the ASR word error rate. Thirdly, we want to build a multi-clause utterance chunker to integrate with our systems. We have identified that complex multi-clause utterances have consistently hurt the system performances. To handle this, we are making efforts along with our colleagues in the speech community to build a real-time speech utterance-chunker. We are eager to discover any performance benefits. Finally, since we already have a corpus containing sentence and their corresponding semantic-frames, we want to explore the possibility of building a Statistical Generator using the same corpus that would take a frame as input and produce a sentence as output. This would take us a step closer to the idea of building a Reversible System that can act as a parser when used in one direction and as a generator when used in the other.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *Proceedings of COLING/ACL*, page 8690, Montreal, Canada.

Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Rahul Bhagat, Anton Leuski, and Eduard Hovy. 2005. Statistical shallow semantic parsing despite little training data. Technical report available at `http://www.isi.edu/~rahul`.

Christopher J. C. Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

Donghui Feng and Eduard Hovy. 2003. Semantics-oriented language understanding with automatic adaptability. In *Proceedings of Natural Language Processing and Knowledge Engineering*.

Paul Kingsbury, Martha Palmer, and Mitch Marcus. 2002. Adding semantic annotation to the penn treebank. In *Proceedings of HLT Conference*.

Christopher D. Manning and Hinrich Schtze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

Jay M. Ponte and W. Bruce Croft. 1997. Text segmentation by topic. In *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, pages 120–129.

W. Swartout, R. Hill, J. Gratch, W. Johnson, C. Kyriakakis, C. LaBore, R. Lindheim, S. Marsella, D. Miraglia, B. Moore, J. Morie, J. Rickel, M. Thiebaux, L. Tuch, R. Whitney, and J. Douglas. 2001. Toward the holodeck: Integrating graphics, sound, character and story. In *Proceedings of Autonomous Agents*.

The Quick Check Pre-unification Filter for Typed Grammars: Extensions

Liviu Ciortuz
CS Department, University of Iaşi, Romania
ciortuz@infoiasi.ro

The so called quick check (henceforth QC) pre-unification filter for feature structure (FS) unification was introduced by (Kiefer et al., 1999). QC is considered the most important speed-up technique in the framework of non-compiled FS unification. We present two potential ways in which the design of the quick check can be further extended: consistency sort check on coreferenced paths, and pre-unification type-checking. We analyse the effect of these extensions on LinGO, the large-scale HPSG grammar for English (Flickinger et al., 2000) using the compiler system LIGHT (Ciortuz, 2002).

1 Coreferenced Based Quick Check

Suppose that the FS φ is going to be unified with ψ, and that φ contains the coreference $\varphi.\pi \doteq \varphi.\pi'$. In this setup, if for a certain path η it happens that $\mathit{sort}(\varphi.(\pi\eta)) \wedge \mathit{sort}(\psi.(\pi\eta)) \wedge \mathit{sort}(\psi.(\pi'\eta)) = \bot$, then certainly φ and ψ are not unifiable. There is no a priori reason why, on certain typed grammars, coreference-based sort inconsistency would not be more effective in ruling out FS unification than sort inconsistency on mutual paths. Moreover, the integration of the two forms of QC is not complicated. However, up to our knowledge no system parsing LinGO-like grammars included the above newly presented form of (coreference-based) pre-unification QC test.

On the GR-reduced form LinGO (Ciortuz, 2004) we identified 12 pairs of non-cross argument coreferences inside rule arguments (at LinGO's source level). Interestingly enough, all these coreferences occur inside key arguments, belonging to only 8 (out of the total of 61) rules in LinGO.

To perform coreference-based QC, we computed the closure of this set Λ of coreference paths. The closure of Λ will be denoted $\bar{\Lambda}$. If the pair π_1 and π_2 is in Λ, then together with it will be included in $\bar{\Lambda}$ all pairs of QC-paths such that $\pi_1\eta$ and $\pi_2\eta$, where η is a feature path (a common suffix to the two newly selected paths). For the GR-reduced form of LinGO, the closure of Λ defined as above amounted to 38 pairs. It is on these pairs of paths that we performed the coreference-based QC test.

Using all these coreference paths pairs, 70,581 unification failures (out of a total of 2,912,623 attempted unifications) were detected on the CSLI test suite. Only 364 of these failures were not detectable through classical QC. When measuring the "sensitivity" of coreferenced-based QC to individual rule arguments, we found that out of a total of 91 rule arguments in LinGO only for 4 rule arguments the coreference-based QC detects inconsistencies, and the number of these inconsistencies is far lower than those detected by the classical QC on the same arguments. None of the pairs of coreference paths exhibited a higher failure detection rate than the first ranked 32 QC-paths. If one would work with 42 QC-paths, then only 4 of the pairs of coreference paths would score failure detection frequencies that would qualify them to be taken into consideration for the (extended form of) QC-test.

As a conclusion, it is clear that for LinGO, running the coreference-based QC test is virtually of no use. For other grammars (or other applications involving FS unification), one may come to a different conclusion, if the use of non-cross argument coreferences balances (or outnumbers) that of cross-

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 188–189,
Vancouver, October 2005.

argument coreferences.

2 Type Checking Based Quick Check

Failure of run-time type checking — the third potential source of inconsistency when unifying two typed FSs — is in general not so easily/efficiently detectable at pre-unification time, because this check requires calling a type consistency check routine which is much more expensive than the simple sort consistency operation.

While exploring the possibility to filter unification failures due to type-checking, the measurements we did using LinGO (the GR-reduced form) on the CSLI test suite resulted in the following facts:

1. Only 137 types out of all 5235 (non-rule and non-lexical) types in LinGO were involved in (either successful or failed) type-checks.[1] Of these types, only 29 types were leading to type checking failure.[2]

2. Without using QC, 449,779 unification failures were due to type-checking on abstract instructions, namely on intersects_sort; type-checking on test_feature acts in fact as type unfolding. When the first 32 QC-paths (from the GR-set of paths) were used (as standard), that number of failures went down to 92,447. And when using all 132 QC-paths (which have been detected on the non GR-reduced form of LinGO), it remained close to the preceding figure: 86,841.

3. For QC on 32 paths, we counted that failed type-checking at intersect_sort occurs only on 14 GR-paths. Of these paths, only 9 produced more than 1000 failures, only 4 produced more than 10,000 failures and finally, for only one GR-path the number of failed type-checks exceeded 20,000.

The numbers given at the above second point suggest that when trying to extend the 'classical' form of QC towards finding all/most of failures, a considerably high number of type inconsistencies will remain in the FSs produced during parsing, even when we use all (GR-paths as) QC-paths. Most of these inconsistencies are due to failed type-checking. And as shown, neither the classical QC nor its extension to (non-cross argument) coreference-based QC is able to detect these inconsistencies.

[1] For 32 QC-paths: 122 types, for 132 QC-paths: also 122.

[2] For 32 QC-paths and 132 QC-paths: 24 and 22 respectively.

s = GRφ[i] $\wedge$ GRψ[i];
if $s \neq$ GRφ[i] and $s \neq$ GRψ[i] and
$\varphi.\pi_j$ or $\psi.\pi_j$ is defined for
a certain non-empty path $\pi_j = \pi_i\pi$,
an extension of π_i,
such that π_j is a QC-path,
then
if $\Psi(s).\pi \wedge$ GR$_\psi$[i] = $\perp$, where
a $\Psi(s)$ is the type corresponding to s,
or type-checking $\psi.\pi_i$ with $\Psi(s)$ fails
then φ and ψ do not unify.

Figure 1: The core of a type-checking specialised compiled QC sub-procedure.

The first and third points from above say that in parsing the CSLI test suite with LinGO, the failures due to type checking tend to agglomerate on certain paths. But due to the fact that type-checking is usually time-costly, our *conclusion*, like in the case of non-cross argument coreference-based QC, is that extending the classical QC by doing a certain amount of type-checking at pre-unification time is not likely to improve significantly the unification (and parsing) performances on LinGO.

For another type-unification grammar one can extend (or replace) the classical QC test with a *type-check QC filter procedure*. Basically, after identifying the set of paths (and types) which most probably cause failure during type-checking, that procedure works as shown in Figure 1.

References

L. Ciortuz. 2002. LIGHT —a constraint language and compiler system for typed-unification grammars. *KI-2002: Advances in Artificial Intelligence*. M. Jarke, J. Koehler and G. Lakemeyer (eds.), pp. 3–17. Springer-Verlag, vol. 2479.

L. Ciortuz. 2004. On two classes of feature paths in large-scale unification grammars. *Recent Advances in Parsing Technologies*. H. Bunt, J. carroll and G. Satta (eds.). Kluwer Academic Publishers.

D. Flickinger, A. Copestake and I. Sag. 2000. HPSG analysis of English. *Verbmobil: Foundations of speech-to-speech translation*. Wolfgang Wahlster (ed.), pp. 254–263. Springer-Verlag.

B. Kiefer, H-U. Krieger, J. Carroll and R. Malouf. 1999. A bag of useful techniques for efficient and robust parsing. *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, pp. 473–480.

From Metagrammars to Factorized TAG/TIG Parsers

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

Abstract

This document shows how the factorized syntactic descriptions provided by Meta-Grammars coupled with factorization operators may be used to derive compact large coverage tree adjoining grammars.

1 Introduction

Large coverage Tree Adjoining Grammars (TAGs) tend to be very large, with several thousands of tree schemata, i.e., trees with at least one anchor node. Such large grammars are difficult to develop and maintain. Of course, their sizes have also a strong impact on parsing efficiency. The size of such TAGs mostly arises from redundancies, due to the extended domain of locality provided by trees.

Recently, Meta-Grammars (Candito, 1999) have been introduced to factorize linguistic information through a multiple inheritance hierarchy of small classes, each of them grouping elementary constraints on nodes. A MG compiler exploits these bits of information to generate a set of trees. While MGs help reducing redundancies at a descriptive level, the resulting grammars remain very large.

We propose to exploit the fact that MGs are already factorized to get compact grammars through the use of factorized trees, as provided by system DYALOG (Thomasset and Villemonte de la Clergerie, 2005).

This proposal has been validated by quickly developing and testing a large coverage French MG.

2 Generic factorization operators

The first factorization operators provided by DYALOG are the *disjunction*, *Kleene star*, and *optionality* operators. A finer control of optionality is provided through the notion of *guards*, used to state conditions on the presence or absence of a node (or of a node sequence). An expression $(G_+, x; G_-)$ means that the guard G_+ (resp. G_-) should be satisfied for x to be present (resp. absent). A guard G is a boolean expression on equations between FS paths and is equivalent to a finite set of substitutions Σ_G. Used to handle local free-word orderings, the *interleaving* (or shuffling) of two sequences $(a_i)_{i=1\cdots n}\#\#(b_j)_{j=1\cdots m}$ returns all sequences containing all a_i and b_j in any order that preserves the original orderings (i.e., $a_i < a_{i+1}$ and $b_j < b_{j+1}$).

These operators do not increase the expressive power or the worst-case complexity of TAGs. They are implemented without expansion, ensuring good performances and more natural parsing output (with no added non-terminals).

3 Meta-Grammars

MGs allow modular descriptions of syntactic phenomena, using elementary constraints grouped into classes. A class may inherit constraints from several parent classes and can also provide a resource or require a resource. Constraints on nodes include equality, precedence, immediate and indirect dominances. The constraints may also be on node and class decorations, expressed with Feature Structures.

The objective of our MG compiler, also developed with DYALOG, is to cross the terminal classes (i.e. any class without descendants) in order to obtain *neutral classes* where each provided resource

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 190–191,
Vancouver, October 2005.

has been consumed and conversely. Constraints are accumulated during crossing and are only kept the neutral classes whose accumulated constraints are satisfiable, taking into account their logical consequence. Minimal trees satisfying the constraints of the neutral classes are then produced.

Getting factorized trees results from several mechanisms. A node may group alternatives, and may be made optional or repeatable (for Kleene stars). When generating trees, underspecified precedences between sibling nodes are handled by the interleaving operator.

Positive and negative guards may be attached to nodes and are accumulated in a conjunctive way during the crossing phase, i.e. $N \Rightarrow G_1$ and $N \Rightarrow G_2$ is equivalent to $N \Rightarrow (G_1, G_2)$. The compiler checks the satisfiability of the guards, removing the alternatives leading to failures and equations in guards which become trivially true. The remaining guards are emitted as DyALog guards in the trees.

4 Grammar anatomy

In just a few months, we have developed, for French, a MG with 191 classes, used to generate a very compact TAG of only 126 trees. Only 27 trees are anchored by verbs and they are sufficient to cover canonical, passive and extracted verbal constructions with at most 2 arguments (including objects, attributes, completives, infinitives, prepositional arguments, wh-completives). These trees would correspond to several thousand trees, if the factorization operators were expanded. This strong compaction rate stems from the presence of 820 guards, 92 disjunctions (to handle choices in realizations), 26 interleavings (to handle verb argument positions) and 13 Kleene stars (to handle coordinations). The grammar is mostly formed of simple trees (with less than 17 nodes), and a few complex trees (26 trees between 30 and 46 nodes), essentially anchored by verbs.

For instance, tree #111[1], used for canonical verb constructions, results from the crossing of 25 terminal classes, and has 43 nodes, plus 3 disjunction nodes (for the different realizations of the subject and other verb arguments) and 1 interleaving node (between the verb arguments and a possible post-verbal subject). The tree is controlled by 35 guards, governing, for instance, the presence and position of a subject and of clitics.

Such a tree covers much more verb subcategorization frames than the number of frames usually attached to a given verb. The anchoring of a tree α by a word w is done by unifying two feature structures $\mathcal{H}_\alpha$ and $\mathcal{H}_w$, called *hypertags* (Kinyon, 2000), that list the syntactic properties covered by α and allowed by w. The link between $\mathcal{H}_\tau$ and the allowed syntactic constructions is done through the variables occurring in $\mathcal{H}_\tau$ and in the guards and node decorations.

5 Evaluation

The resulting French grammar has been compiled, with DyALog, into an hybrid TAG/TIG parser, by identifying the left and right auxiliary insertion trees. Following a left-to-right top-down tabular parsing strategy, the parser may be used to get either full or partial parses.[2] Coverage rate for full parsing is around 95% for two test suites (EUROTRA and TSNLP) and around 42% on various corpora (including more than 300K sentences of a raw journalistic corpus).

Our MG is still very young and needs to be improved to ensure a better coverage. However, we can already conclude that coupling MGs with factorized trees is a generic and powerful approach to control the size of grammars and to get efficient parsers.

The various tools and linguistic resources mentioned in this abstract are freely available at `http://atoll.inria.fr/`.

References

M.-H. Candito. 1999. *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, Université Paris 7.

A. Kinyon. 2000. Hypertags. In *Proc. of COLING*, pages 446–452.

F. Thomasset and É. Villemonte de la Clergerie. 2005. Comment obtenir plus des méta-grammaires. In *Proceedings of TALN'05*, volume 1, pages 3–12, Dourdan, France, June. ATALA.

[1] browsable online at `http://atoll.inria.fr/perl/frmg/tree.pl`.

[2] The parser may be tried online at `http://atoll.inria.fr/parserdemo`.

Parsing Generalized ID/LP Grammars

Michael W. Daniels
Department of Linguistics
Indiana University
402 Memorial Hall
1021 E Third Street
Bloomington, IN 47405
daniels@ling.osu.edu

1 Introduction

The Generalized ID/LP (GIDLP) grammar formalism (Daniels and Meurers 2004a,b; Daniels 2005) was developed to serve as a processing backbone for linearization-HPSG grammars, separating the declaration of the recursive constituent structure from the declaration of word order domains. This paper shows that the key aspects of this formalism – the ability for grammar writers to explicitly declare word order domains and to arrange the right-hand side of each grammar rule to minimize the parser's search space – lead directly to improvements in parsing efficiency.

2 Defining GIDLP Grammars

A brief overview of GIDLP syntax is given in 1, and an example GIDLP grammar is given in 2 that recognizes a very small fragment of German, focusing on the free word order of arguments and adjuncts in the *Mittelfeld*.[1] The basic idea of this grammar is that no word order constraints apply below the level of the clause. This allows the verb's arguments and adjuncts to freely intermingle, before being compacted at the clause level, at which point the constraints on the location of the finite verb apply. It is important to note that this grammar cannot be straightforwardly expressed in the ID/LP formalism, where LP constraints only apply within local trees.

3 The GIDLP Parsing Algorithm

The GIDLP parser Daniels and Meurers (2004a); Daniels (2005) is based on Earley's algorithm for context-free parsing, suitably modified to handle discontinuous constituents.

A central insight of the GIDLP parsing algorithm is that the same data structure used to describe the coverage of an edge can also encode restrictions on the parser's search space. This is done by adding two bitvectors to each edge: a *negative mask* (n-mask), which marks positions that must not be part of the edge, and a *positive mask* (p-mask), which marks positions that must be part of the edge. These masks are generated during the prediction phase and then tested during the completion phase using efficient bitvector operations. Compiling LP constraints into

Terminal: t
Non-terminal: C
Lexical entry: $C \rightarrow t$
Grammar rule: $C \rightarrow C^+; LP^*; DD^*$
Start declaration: $start(S) : LP^*$
LP [Constraint]: $C_1\{<, \ll\}C_2$
D[omain] D[eclaration]: $\langle\{C^+\}, C, LP^*\rangle$

Figure 1: GIDLP syntax

a) start(s): []
b) s → s(cmp)
c) s → s(que)
d) s(cmp) → cmp, clause;
⟨{[0]}, s(cmp), cmp < _, _ < v(_)⟩
e) s(que) → clause;⟨{[0]}, s(que), v(_) < _⟩
f) clause → np(n), vp
g) vp → v(ditr), np(a), np(d)
h) vp → adv, vp
i) vp → v(cmp), s(cmp)
j) [np(Case)] → det(Case), n(Case);1 ≪ 2

Figure 2: Example GIDLP Grammar

[1] For compactness, categories are described in this example with prolog-style terms; the actual GIDLP syntax assumes feature structure categories.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 192–193,
Vancouver, October 2005.

bitmasks in this way allows the LP constraints to be integrated directly into the parser at a fundamental level. Instead of weeding out inappropriate parses in a cleanup phase, LP constraints in this parser can immediately block an edge from being added to the chart.

4 Evaluation

To evaluate the effectiveness of the GIDLP formalism, a moderate-scale grammar of German was obtained from Professor Martin Volk (Stockholm University). This grammar combines ID/LP rules with PS rules, as argued for in (Volk 1996), and uses a flat structure to encode the flexibility of the German *Mittelfeld*. As an example, the rule for ditransitive verbs is given in (1).

(1) `S -> N2 V N2 N2 ADV* (ERG) (PRAEF)`

This grammar can be mechanically translated into the GIDLP format, as each of Volk's PS rules corresponds to a GIDLP rule. This translation establishes an 'initial' GIDLP grammar.[2] The grammar was then optimized in two successive steps to take advantage of the GIDLP formalism. First, a 'medial' grammar was created in which word order domains were introduced only when necessary. (In the ID/LP formalism, every local tree is an order domain.) Second, a 'final' grammar was created by reordering the RHS order of each rule so as to put the most discriminatory RHS element first – generally the finite verb.

To compare these three grammars, a testsuite of 150 sentences was constructed. The sentences were generally chosen to equally cover the sentence types recognized by the grammar. The results from parsing this testsuite with each grammar are summarized in Figure 3, which shows the average number of chart insertion attempts at each sentence length. (Chart insertion attempts have traditionally been used as an overall metric for parsing efficiency, as parse time tends to be dominated by the time taken searching the chart for blocking edges.) Overall, the final grammar shows a clear improvement over the medial and initial grammars.

[2] As Volk's parser is not available, the relative performance of the GIDLP parser on the initial grammar and of Volk's parser on his grammar cannot be determined. Thus Volk's grammar is only used as a basis for the three GIDLP grammars described here.

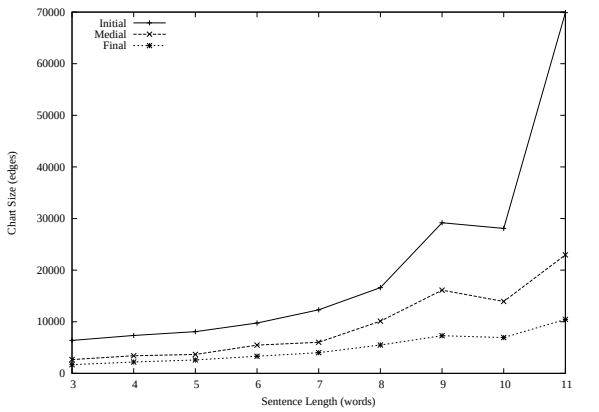


Figure 3: Average Chart Size per Sentence Length

Averaging over all 150 sentences, the final grammar sees a decrease of 69.2% in the number of chart insertion attempts compared to the initial grammar. Thus the expressive capabilities provided by the GIDLP formalism lead directly to improvements in parsing efficiency.

5 Summary

This paper has shown that two key aspects of the GIDLP grammar formalism – the ability for grammar writers to explicitly declare word order domains in the spirit of the linearization-HPSG tradition and the ability to completely order the RHS of a grammar rule to minimize the parser's overall search space – lead directly to improvements in parse efficiency.

References

Daniels, Michael W. 2005. *Generalized ID/LP Grammar: A Formalism for Parsing Linearization-Based HPSG Grammars*. Ph. D.thesis, The Ohio State University.

Daniels, Michael W. and Meurers, W. Detmar. 2004a. A Grammar Formalism and Parser for Linearization-based HPSG. In *Proceedings of the Twentieth International Conference on Computational Linguistics*, pages 169–175.

Daniels, Mike and Meurers, Detmar. 2004b. GIDLP: A Grammar Format for Linearization-Based HPSG. In Stefan Müller (ed.), *Proceedings of the Eleventh International Conference on Head-Driven Phrase Structure Grammar*, pages 93–111, Stanford: CSLI Publications.

Volk, Martin. 1996. Parsing with ID/LP and PS rules. In *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference (Bielefeld)*, pages 342–353, Berlin: Mouton de Gruyter.

TFLEX: Speeding up Deep Parsing with Strategic Pruning

Myroslava O. Dzikovska
Human Communication Research Centre
University of Edinburgh
Edinburgh, EH8 9LW, UK
mdzikovs@inf.ed.ac.uk

Carolyn P. Rose
Carnegie Mellon University
Language Technologies Institute
Pittsburgh PA 15213, USA
cprose@cs.cmu.edu

1 Introduction

This paper presents a method for speeding up a deep parser through backbone extraction and pruning based on CFG ambiguity packing.[1] The TRIPS grammar is a wide-coverage grammar for deep natural language understanding in dialogue, utilized in 6 different application domains, and with high coverage and sentence-level accuracy on human-human task-oriented dialogue corpora (Dzikovska, 2004). The TRIPS parser uses a best-first beam search algorithm and a chart size limit, both of which are a form of pruning focused on finding an n-best list of interpretations. However, for longer sentences limiting the chart size results in failed parses, while increasing the chart size limits significantly impacts the parsing speed.

It is possible to speed up parsing by implementing faster unification algorithms, but this requires considerable implementation effort. Instead, we developed a new parser, TFLEX, which uses a simpler technique to address efficiency issues. TFLEX combines the TRIPS grammar with the fast parsing technologies implemented in the LCFLEX parser (Rosé and Lavie, 2001). LCFLEX is an all-paths parser which uses left-corner prediction and ambiguity packing, and which was shown to be efficient on other unification augmented context-free grammars. We describe a way to transfer the TRIPS grammar to LCFLEX, and a pruning method which achieves significant improvements in both speed and coverage compared to the original TRIPS parser.

[1]This material is based on work supported by grants from the Office of Naval Research under numbers N000140510048 and N000140510043.

2 TFLEX

To use the TRIPS grammar in LCFLEX we first extracted a CFG backbone from the TRIPS grammar, with CFG non-terminals corresponding directly to TRIPS constituent categories. To each CFG rule we attach a corresponding TRIPS rule. Whenever a CFG rule completes, a TRIPS unification function is called to do all the unification operations associated with the TRIPS rule. If the unification fails, the constituent built by the CFG is cancelled.

The TFLEX pruning algorithm uses ambiguity packing to provide good pruning points. For example, in the sentence "we have a heart attack victim at marketplace mall" the phrase "a heart attack victim" has two interpretations depending on whether "heart" modifies "attack" or "attack victim". These interpretations will be ambiguity packed in the CFG structure, which offers an opportunity to make pruning more strategic by focusing specifically on competing interpretations for the same utterance span. For any constituent where ambiguity-packed non-head daughters differ only in local features, we prune the interpretations coming from them to a specified prune beam width based on their TRIPS scores. In the example above, pruning will happen at the point of making a VP "have a heart attack victim". The NP will be ambiguity packed, and we will prune alternative VP interpretations resulting from combining the same sense of the verb "have" and different interpretations of the NP.

This approach works better than the original TRIPS best-first algorithm, because for long sentence the TRIPS chart contains a large number

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 194–195,
Vancouver, October 2005.

of similar constituents, and the parser frequently reaches the chart size limit before finding the correct constituent to use. Ambiguity packing in TFLEX helps chose the best constituents to prune by pruning competing interpretations which cover the same span and have the same non-local features, thus making it less likely that a constituent essential for building a parse will be pruned.

3 Evaluation

Our evaluation data is an excerpt from the Monroe corpus that has been used in previous TRIPS research on parsing speed and accuracy (Swift et al., 2004). The test contained 1042 utterances, from 1 to 45 words in length (mean 5.38 words/utt, st. dev. 5.7 words/utt). Using a hold-out set, we determined that a beam width of 3 was an optimal setting for TFLEX. We then compared TFLEX at beam width 3 to the TRIPS parser with chart size limits of 1500, 5000, and 10000. As our evaluation metrics we report are average parse time per sentence and probability of finding at least one parse, the latter being a measure approximating parsing accuracy.

The results are presented in Figure 1. We grouped sentences into equivalence classes based on length with a 5-word increment. On sentences greater than 10 words long, TFLEX is significantly more likely to produce a parse than any of the TRIPS parsers (evaluated using a binary logistic regression, $p < .001$). Moreover, for sentences greater than 20 words long, no form of TRIPS parser returned a complete parse. TFLEX is significantly faster than TRIPS-10000, statistically indistinguishable in terms of parse time from TRIPS-5000, and significantly slower than TRIPS-1500 ($p < .001$).

Thus, TFLEX presents a superior balance of coverage and efficiency especially for long sentences (10 words or more) since for these sentences it is significantly more likely to find a parse than any version of TRIPS, even a version where the chart size is expanded to an extent that it becomes significantly slower (i.e., TRIPS-10000).

4 Conclusions

In this paper, we described a combination of efficient parsing techniques to improve parsing speed and coverage with the TRIPS deep parsing grammar.

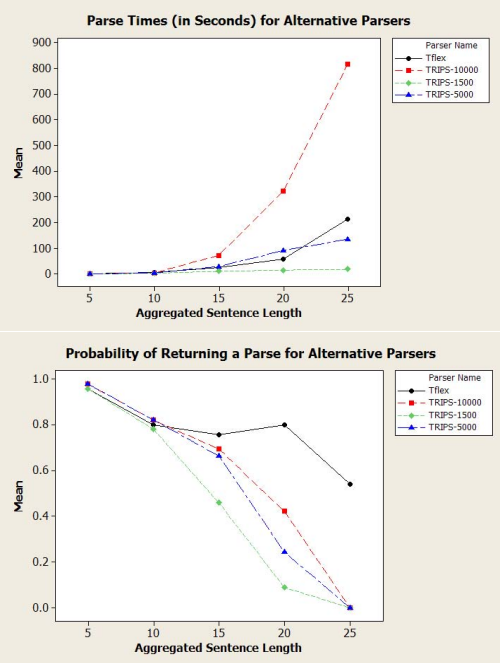


Figure 1: Parse times and probability of getting a parse depending on (aggregated) sentence lengths. 5 denotes sentences with 5 or fewer words, 25 sentences with more than 20 words.

The TFLEX system uses an all-paths left-corner parsing from the LCFLEX parser, made tractable by a pruning algorithm based on ambiguity packing and local features, generalizable to other unification grammars. Our pruning algorithm provides a better efficiency-coverage balance than best-first parsing with chart limits as utilised by the TRIPS parser.

References

M. O. Dzikovska. 2004. *A Practical Semantic Representation For Natural Language Parsing*. Ph.D. thesis, University of Rochester.

C. P. Rosé and A. Lavie. 2001. Balancing robustness and efficiency in unification-augmented context-free parsers for large practical applications. In J.C. Junqua and G Van Noord, editors, *Robustness in Language and Speech Technology*. Kluwer Academic Press.

M. Swift, J. Allen, and D. Gildea. 2004. Skeletons in the parser: Using a shallow parser to improve deep parsing. In *Proceedings of COLING-04*.

J. Tetreault, M. Swift, P. Prithviraj, M. Dzikovska, and J. Allen. 2004. Discourse annotation in the monroe corpus. In *ACL-04 workshop on Discourse Annotation*.

Generic parsing for multi-domain semantic interpretation

Myroslava Dzikovska*, Mary Swift†, James Allen†, William de Beaumont†
* Human Communication Research Centre
University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, United Kingdom
m.dzikovska@ed.ac.uk
† Department of Computer Science University of Rochester, Rochester, NY 14627-0226
{swift, james, wdebeaum}@cs.rochester.edu

1 Introduction

Producing detailed syntactic and semantic representations of natural language is essential for practical dialog systems such as plan-based assistants and tutorial systems. Development of such systems is time-consuming and costly as they are typically hand-crafted for each application, and dialog corpus data is more difficult to obtain than text. The TRIPS parser and grammar addresses these issues by providing broad coverage of common constructions in practical dialog and producing semantic representations suitable for dialog processing across domains. Our system bootstraps dialog system development in new domains and helps build parsed corpora.[1]

Evaluating deep parsers is a challenge (e.g., (Kaplan et al., 2004)). Although common bracketing accuracy metrics may provide a baseline, they are insufficient for applications such as ours that require complete and correct semantic representations produced by the parser. We evaluate our parser on bracketing accuracy against a statistical parser as a baseline, then on a word sense disambiguation task, and finally on full sentence syntactic and semantic accuracy in multiple domains as a realistic measure of system performance and portability.

2 The TRIPS Parser and Logical Form

The TRIPS grammar is a linguistically motivated unification formalism using attribute-value structures. An unscoped neo-Davidsonian semantic representation is built in parallel with the syntactic representation. A sample logical form (LF) representation for *Load the oranges into the truck* is shown above. The TRIPS LF provides the necessary information for reference resolution, surface speech act analysis, and interpretations for a wide variety of fragmentary utterances and conventional phrases typical in dialog. The LF content comes from a domain-independent ontology adapted from FrameNet (Johnson and Fillmore, 2000; Dzikovska et al., 2004) and linked to a domain-independent lexicon (Dzikovska, 2004).

```
(SPEECHACT sa1 SA_REQUEST :content e123)
(F e123 (:* LF::Fill-Container Load)
        :Agent pro1 :Theme v1 :Goal v2)
(IMPRO pro1 LF::Person :context-rel *YOU*)
(THE v1 (SET-OF (:* LF::Fruit Orange)))
(THE v2 (:* LF::Vehicle Truck))
```

Figure 1: LF for *Load the oranges into the truck.*

The parser uses a bottom-up chart algorithm with beam search. Alternative parses are scored with factors assigned to grammar rules and lexical entries by hand, because due to the limited amount of corpus data we have not yet been able to train a statistical model that outperforms our hand-tuned factors.

3 Evaluation

As a rough baseline, we compared the bracketing accuracy of our parser to that of a statistical parser (Bikel, 2002), Bikel-M, trained on 4294 TRIPS

[1]We thank 4 anonymous reviewers for comments. This material is based on work supported by grants from ONR #N000149910165, NSF #IIS-0328811, DARPA #NBCHD030010 via subcontract to SRI #03-000223 and NSF #E1A-0080124.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 196–197,
Vancouver, October 2005.

parse trees from the Monroe corpus (Stent, 2001), task-oriented human dialogs in an emergency rescue domain. 100 randomly selected utterances were held out for testing. The gold standard for evaluation is created with the help of the parser (Swift et al., 2004). Corpus utterances are parsed, and the parsed output is checked by trained annotators for full-sentence syntactic and semantic accuracy, reliable with a kappa score 0.79. For test utterances for which TRIPS failed to produce a correct parse, gold standard trees were manually constructed independently by two linguists and reconciled. Table 1 shows results for the 100 test utterances and for the subset for which TRIPS finds a spanning parse (74). Bikel-M performs somewhat better on the bracketing task for the entire test set, which includes utterances for which TRIPS failed to find a parse, but it is lower on complete matches, which are crucial for semantic interpretation.

	All test utts (100)			Spanning parse utts (74)		
	R	P	CM	R	P	CM
BIKEL-M	79	79	42	89	88	54
TRIPS	77	79	65	95	95	86

Table 1: Bracketing results for Monroe test sets (R: recall, P: precision, CM: complete match).

Word senses are an important part of the LF representation, so we also evaluated TRIPS on word sense tagging against a baseline of the most common word senses in Monroe. There were 546 instances of ambiguous words in the 100 test utterances. TRIPS tagged 90.3% (493) of these correctly, compared to the baseline model of 75.3% (411) correct.

To evaluate portability to new domains, we compared TRIPS full sentence accuracy on a subset of Monroe that underwent a fair amount of development (Tetreault et al., 2004) to corpora of keyboard tutorial session transcripts from new domains in basic electronics (BEETLE) and differentiation (LAM) (Table 2). The only development for these domains was addition of missing lexical items and two grammar rules. TRIPS full accuracy requires correct speech act, word sense and thematic role assignment as well as complete constituent match.

Error analysis shows that certain senses and subcategorization frames for existing words are still needed in the new domains, which can be rectified fairly quickly. Finding and addressing such gaps is part of bootstrapping a system in a new domain.

Domain	Utts	Acc.	Cov.	Prec.
Monroe	1576	70%	1301	84.1%
BEETLE	192	50%	129	75%
LAM	934	42%	579	68%

Table 2: TRIPS full sentence syntactic and semantic accuracy in 3 domains (Acc: full accuracy; Cov.: # spanning parses; Prec: full acc. on spanning parses).

4 Conclusion

Our wide-coverage grammar, together with a domain-independent ontology and lexicon, produces semantic representations applicable across domains that are detailed enough for practical dialog applications. Our generic components reduce development effort when porting to new dialog domains where corpus data is difficult to obtain.

References

D. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *HLT-2002*.

M. O. Dzikovska, M. D. Swift, and J. F. Allen. 2004. Building a computational lexicon and ontology with framenet. In *LREC workshop on Building Lexical Resources from Semantically Annotated Corpora*.

M. O. Dzikovska. 2004. *A Practical Semantic Representation For Natural Language Parsing*. Ph.D. thesis, University of Rochester.

C. Johnson and C. J. Fillmore. 2000. The FrameNet tagset for frame-semantic and syntactic coding of predicate-argument structure. In *ANLP-NAACL 2000*.

R. M. Kaplan, S. Riezler, T. H. King, J. T. Maxwell III, A. Vasserman, and R. S. Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *HLT-NAACL 2004*.

A. J. Stent. 2001. *Dialogue Systems as Conversational Partners*. Ph.D. thesis, University of Rochester.

M. D. Swift, M. O. Dzikovska, J. R. Tetreault, and J. F. Allen. 2004. Semi-automatic syntactic and semantic corpus annotation with a deep parser. In *LREC-2004*.

J. Tetreault, M. Swift, P. Prithviraj, M. Dzikovska, and J. Allen. 2004. Discourse annotation in the Monroe corpus. In *ACL workshop on Discourse Annotation*.

Online Statistics for a Unification-Based Dialogue Parser

Micha Elsner, Mary Swift, James Allen, and Daniel Gildea
Department of Computer Science
University of Rochester
Rochester, NY 14627
{melsner,swift,allen,gildea}@cs.rochester.edu

Abstract

We describe a method for augmenting unification-based deep parsing with statistical methods. We extend and adapt the Bikel parser, which uses head-driven lexical statistics, to dialogue. We show that our augmented parser produces significantly fewer constituents than the baseline system and achieves comparable bracketing accuracy, even yielding slight improvements for longer sentences.

1 Introduction

Unification parsers have problems with efficiency and selecting the best parse. Lexically-conditioned statistics as used by Collins (1999) may provide a solution. They have been used in three ways: as a postprocess for parse selection (Toutanova et al., 2005; Riezler et al., 2000; Riezler et al., 2002), a preprocess to find more probable bracketing structures (Swift et al., 2004), and online to rank each constituent produced, as in Tsuruoka et al. (2004) and this experiment.

The TRIPS parser (Allen et al., 1996) is a unification parser using an HPSG-inspired grammar and hand-tuned weights for each rule. In our augmented system (Aug-TRIPS), we replaced these weights with a lexically-conditioned model based on the adaptation of Collins used by Bikel (2002), allowing more efficiency and (in some cases) better selection. Aug-TRIPS retains the same grammar and lexicon as TRIPS, but uses its statistical model to determine the order in which unifications are attempted.

2 Experiments

We tested bracketing accuracy on the Monroe corpus (Stent, 2001), which contains collaborative emergency-management dialogues. Aug-TRIPS is comparable to TRIPS in accuracy, but produces fewer constituents (Table 1). The Bikel parser has slightly higher precision/recall than either TRIPS or Aug-TRIPS, since it can choose any bracketing structure regardless of semantic coherence, while the TRIPS systems must find a legal pattern of feature unifications. Aug-TRIPS also has better precision/recall when parsing the longer sentences (Table 2).

(training=9282)	Bikel	Aug-TRIPS	TRIPS
Recall	79.40	76.09	76.77
Precision	79.40	77.08	78.20
Complete Match	42.00	46.00	65.00
% Constit. Reduction	-	36.96	0.00

Table 1: Bracketing accuracy for 100 random sentences ≥ 2 words.

	> 7 Aug-TRIPS	> 7 TRIPS
Recall	73.25	71.00
Precision	74.78	73.44
Complete Match	22.50	37.50

Table 2: Bracketing accuracy for the 40 sentences $>$ 7 words.

Since our motivation for unification parsing is to reveal semantics as well as syntax, we next evaluated Aug-TRIPS's production of correct interpretations at the sentence level, which require complete correctness not only of the bracketing structure but of the sense chosen for each word and the thematic

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 198–199,
Vancouver, October 2005.

roles of each argument (Tetreault et al., 2004).

For this task, we modified the probability model to condition on the senses in our lexicon rather than words. For instance, the words "two thousand dollars" are replaced with the senses "number number-unit money-unit". This allows us to model lexical disambiguation explicitly. The model generates one or more senses from each word with probability $P(sense|word, tag)$, and then uses sense statistics rather than word statistics in all other calculations. Similar but more complex models were used in the PCFG-sem model of Toutanova et al. (2005) and using WordNet senses in Bikel (2000).

We used the Projector dialogues (835 sentences), which concern purchasing video projectors. In this domain, Aug-TRIPS makes about 10% more interpretation errors than TRIPS (Table 3), but when parsing sentences on which TRIPS itself makes errors, it can correct about 10% (Table 4).

(training=310)	TRIPS	Aug-TRIPS
Correct	26	21
Incorrect	49	54
% Reduction in Constituents	0%	45%

Table 3: Sentence-level accuracy on 75 random sentences.

(training=396)	TRIPS	Aug-TRIPS
Correct	0	8
Incorrect	54	46
% Reduction in Constituents	0%	46%

Table 4: Sentence-level accuracy on 54 TRIPS error sentences

Our parser makes substantially fewer constituents than baseline TRIPS at only slightly lower accuracy. Tsuruoka et al. (2004) achieved a much higher speedup (30 times) than we did; this is partly due to their use of the Penn Treebank, which contains much more data than our corpora. In addition, however, their baseline system is a classic HPSG parser with no efficiency features, while our baseline, TRIPS, is designed as a real-time dialogue parser which uses hand-tuned weights to guide its search and imposes a maximum chart size.

Acknowledgements Our thanks to Will DeBeaumont and four anonymous reviewers.

References

James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. 1996. A robust system for natural spoken dialogue. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96)*.

Daniel Bikel. 2000. A statistical model for parsing and word-sense disambiguation. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong.

Daniel Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Human Language Technology Conference (HLT)*, San Diego.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the ACL*, Hong Kong.

Stefan Riezler, Tracy H. King, Richard Crouch, and John T. Maxwell. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation. In *Proceedings of the 40th Annual Meeting of the ACL*, Philadelphia.

Amanda J. Stent. 2001. *Dialogue Systems as Conversational Partners*. Ph.D. thesis, University of Rochester.

Mary Swift, James Allen, and Daniel Gildea. 2004. Skeletons in the parser: Using a shallow parser to improve deep parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, Geneva, Switzerland, August.

Joel Tetreault, Mary Swift, Preethum Prithviraj, Myroslava Dzikovska, and James Allen. 2004. Discourse annotation in the Monroe corpus. In *ACL workshop on Discourse Annotation*, Barcelona, Spain, July.

Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG parse disambiguation using the Redwoods corpus. *Journal of Logic and Computation*.

Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2004. Towards efficient probabilistic HPSG parsing: Integrating semantic and syntactic preference to guide the parsing. In *Proceedings of IJCNLP-04 Workshop: Beyond Shallow Analyses- Formalisms and Statistical Modeling for Deep Analyses*, Sanya City, China.

SUPPLE: A Practical Parser for Natural Language Engineering Applications

Robert Gaizauskas, Mark Hepple, Horacio Saggion,
Mark A. Greenwood and Kevin Humphreys*
Department of Computer Science
University of Sheffield, Sheffield, UK
{robertg|hepple|saggion|m.greenwood|-}@dcs.shef.ac.uk

Abstract

We describe SUPPLE, a freely-available, open source natural language parsing system, implemented in Prolog, and designed for practical use in language engineering (LE) applications. SUPPLE can be run as a stand-alone application, or as a component within the GATE General Architecture for Text Engineering. SUPPLE is distributed with an example grammar that has been developed over a number of years across several LE projects. This paper describes the key characteristics of the parser and the distributed grammar.

1 Introduction

In this paper we describe SUPPLE[1] — the Sheffield University Prolog Parser for Language Engineering — a general purpose parser that produces both syntactic and semantic representations for input sentences, which is well-suited for a range of LE applications. SUPPLE is freely available, and is distributed with an example grammar for English that was developed across a number of LE projects. We will describe key characteristics of the parser and the grammar in turn.

2 The SUPPLE Parser

SUPPLE is a general purpose bottom-up chart parser for feature-based context free phrase structure grammars (CF-PSGs), written in Prolog, that has a number of characteristics making it well-suited for use in LE applications. It is available both as a language processing resource within the GATE General Architecture for Text Engineering (Cunningham et al., 2002) and as a standalone program requiring various preprocessing steps to be applied to the input. We will here list some of its key characteristics.

Firstly, the parser allows multiword units identified by earlier processing components, e.g. named entity recognisers (NERs), gazetteers, etc, to be treated as non-decomposable units for syntactic processing. This is important as the identification of such items is an essential part of analyzing real text in many domains.

The parser allows a layered parsing process, with a number of separate grammars being applied in series, one on top of the other, with a "best parse" selection process between stages so that only a subset of the constituents constructed at each stage is passed forward to the next. While this may make the parsing process incomplete with respect to the total set of analyses licensed by the grammar rules, it makes the parsing process much more efficient and allows a modular development of sub-grammars.

Facilities are provided to simplify handling feature-based grammars. The grammar representation uses flat, i.e. non-embedded, feature representations which are combined used Prolog term unification for efficiency. Features are predefined and source grammars compiled into a full form representation, allowing grammar writers to include only relevant features in any rule, and to ignore feature ordering. The formalism also permits disjunctive and optional right-hand-side constituents.

The chart parsing algorithm is simple but very

*At Microsoft Corporation since 2000 (Speech and Natural Language Group). Email: kevinhum@microsoft.com.

[1] In previous published materials and in the current GATE release the parser is referred to as buChart. This is name is now deprecated.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 200–201,
Vancouver, October 2005.

efficient, exploiting the characteristics of Prolog to avoid the need for active edges or an agenda. In informal testing, this approach was roughly ten times faster than a related Prolog implementation of standard bottom-up active chart parsing.

The parser does not fail if full sentential parses cannot be found, but instead outputs partial analyses as syntactic and semantic fragments for user-selectable syntactic categories. This makes the parser robust in applications which deal with large volumes of real text.

3 The Sample Grammar

The sample grammar distributed with SUPPLE has been developed over several years, across a number LE projects. We here list some key characteristics.

The morpho-syntactic and semantic information required for individual lexical items is minimal — inflectional root and word class only, where the word class inventory is basically the PTB tagset.

A conservative philosophy is adopted regarding identification of verbal arguments and attachment of nominal and verbal post-modifiers, such as prepositional phrases and relative clauses. Rather than producing all possible analyses or using probabilities to generate the most likely analysis, the preference is to offer a single analysis that spans the input sentence only if it can be relied on to be correct, so that in many cases only partial analyses are produced. The philosophy is that it is more useful to produce partial analyses that are correct than full analyses which may well be wrong or highly disjunctive. Output from the parser can be passed to further processing components which may bring additional information to bear in resolving attachments.

An analysis of verb phrases is adopted in which a core verb cluster consisting of verbal head plus auxiliaries and adverbials is identified before any attempt to attach any post-verbal arguments. This contrasts with analyses where complements are attached to the verbal head at a lower level than auxiliaries and adverbials, e.g. as in the Penn TreeBank. This decision is again motivated by practical concerns: it is relatively easy to recognise verbal clusters, much harder to correctly attach complements.

A semantic analysis, or simplified quasi-logical form (SQLF), is produced for each phrasal constituent, in which tensed verbs are interpreted as referring to unique events, and noun phrases as referring to unique objects. Where relations between syntactic constituents are identified in parsing, semantic relations between associated objects and events are asserted in the SQLF.

While linguistically richer grammatical theories could be implemented in the grammar formalism of SUPPLE, the emphasis in our work has been on building robust wide-coverage tools — hence the requirement for only minimal lexical morphosyntactic and semantic information. As a consequence the combination of parser and grammars developed to date results in a tool that, although capable of returning full sentence analyses, more commonly returns results that include chunks of analysis with some, but not all, attachment relations determined.

4 Downloading SUPPLE Resources

SUPPLE resources, including source code and the sample grammar, and also a longer paper providing a more detailed account of both the parser and grammar, are available from the supple homepage at:

`http://nlp.shef.ac.uk/research/supple`

5 Conclusion

The SUPPLE parser has served as a component in numerous LE research projects, and is currently in use in a Question Answering system which participated in recent TREC/QA evaluations. We hope its availability as a GATE component will facilitate its broader use by NLP researchers, and by others building applications exploiting NL technology.

Acknowledgements

The authors would like to acknowledge the support of the UK EPSRC under grants R91465 and K25267, and also the contributions of Chris Huyck and Sam Scott to the parser code and grammars.

References

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

k-NN for Local Probability Estimation in Generative Parsing Models

Deirdre Hogan
Department of Computer Science
Trinity College Dublin
Dublin 2, Ireland
dhogan@cs.tcd.ie

Abstract

We describe a history-based generative parsing model which uses a *k*-nearest neighbour (*k*-NN) technique to estimate the model's parameters. Taking the output of a base *n*-best parser we use our model to re-estimate the log probability of each parse tree in the *n*-best list for sentences from the Penn Wall Street Journal treebank. By further decomposing the local probability distributions of the base model, enriching the set of conditioning features used to estimate the model's parameters, and using *k*-NN as opposed to the Witten-Bell estimation of the base model, we achieve an *f*-score of 89.2%, representing a 4% relative decrease in *f*-score error over the 1-best output of the base parser.

1 Introduction

This paper describes a generative probabilistic model for parsing, based on Collins (1999), which re-estimates the probability of each parse generated by an initial base parser (Bikel, 2004) using memory-based techniques to estimate local probabilities.

We used Bikel's re-implementation of the Collins parser (Bikel, 2004) to produce the *n*-best parses of sentences from the Penn treebank. We then recalculated the probability of each parse tree using a probabilistic model very similar to Collins (1999) Model 1. In addition to the local estimation technique used, our model differs from Collins (1999) Model 1 in that we extend the feature sets used to predict parse structure to include more features from the parse history, and we further decompose some of the model's parameter classes.

2 Constraint Features for Training Set Restriction

We use the same k-NN estimation technique as Toutonava et al (2003) however we also found that restricting the number of examples in the training set used in a particular parameter estimation helped both in terms of accuracy and speed. We restricted the training sets by making use of constraint features whereby the training set is restricted to only those examples which have the same value for the constraint feature as the query instance.

We carried out experiments using different sets of constraint features, some more restrictive than others. The mechanism we used is as follows: if the number of examples in the training set, retrieved using a particular set of constraint features, exceeds a certain threshold value then use a higher level of restriction i.e. one which uses more constraint features. If, using the higher level of restriction, the number of samples in the training set falls below a minimum threshold value then "back-off" to the less restricted set of training samples.

3 Experiments

Our model is trained on sections 2 to 21 inclusive of the Penn WSJ treebank and tested on section 23. We used sections 0, 1, 22 and 24 for validation.

We re-estimated the probability of each parse using our own baseline model, which is a replication of Collins Model 1. We tested *k*-NN estimation on the head generation parameter class

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 202–203,
Vancouver, October 2005.

and the parameter classes for generating modifying nonterminals. We further decomposed the two modifying nonterminal parameter classes. Table 1 outlines the parameter classes estimated using *k*-NN in the final model settings and shows the feature sets used for each parameter class as well as the constraint feature settings.

Parameter Class	History	Contraint Features
$P(C_H \mid \ldots)$	C_p, C_H, w_p, t_p, t_{gp}	$\{C_p\}$
$P(t_i \mid \ldots)$	dir, C_p, C_H, w_p, t_p, dist, t_{i-1}, t_{i-2}, C_{gp}	{dir, C_p}, {dir, C_p, C_H }
$P(C_i \mid \ldots)$	dir, t_i, C_p C_H, w_p, t_p, dist, t_{i-1},t_{i-2}, C_{gp}	{dir,ti},{dir, t_i, C_p}
P(coord,punc|…)	dir, C_i, t_i, C_p, C_H, w_p, ,t_p	{dir, C_i, t_i}
$P(C_i\ t_i \mid C_p$ =NPB…)	dir, C_H, w_p, C_{i-2}, w_{i-2}, C_{i-3}, w_{i-3}, C_{gp}, C_{ggp}, C_{gggp}	{dir, C_H }
P(punc| C_p =NPB…)	dir, t_i, C_i, C_H, w_p,t_p, t_{i-2}, t_{i-3}	{dir, t_i}

Table 1: The parameter classes estimated using *k*-NN in the final model. C_H is the head child label, C_p the parent constituent label, w_p the head word, t_p the head part-of-speech (POS) tag. C_i, w_i and t_i are the modifier's label, head word and head POS tag. t_{gp} is the grand-parent POS tag, C_{gp}, C_{ggp}, C_{gggp} are the labels of the grand-parent, great-grandparent and great-great-grandparent nodes. *dir* is a flag which indicates whether the modifier being generated is to the left or the right of the head child. *dist* is the distance metric used in the Collins parser. *coord*, *punc* are the coordination and punctuation flags. NPB stands for base noun phrase.

We extend the original feature sets by increasing the order of both horizontal and vertical markovization. From each constituent node in the vertical or horizontal history we chose features from among the constituent's nonterminal label, its head word and the head word's part-of-speech tag. We found for all parameter classes $k = 10{,}000$ or $k = 20{,}000$ worked best. Distance weighting function that worked best were the inverse distance weighting functions either $(1/(d+1))^6$ or $(1/(d+1))^7$.

Model	LR	LP
WB Baseline	88.2%	88.5%
CO99 M1	87.9%	88.2%
CO99 M2	88.5%	88.7%
Bikel 1-best	88.7%	88.7%
***k*-NN**	89.1%	89.4%

Table 2: Results for sentences of less than or equal to 40 words, from section 23 of the Penn treebank. LP/LR =Labelled Precision/Recall. CO99 M1 and M2 are (Collins 1999) Models 1 and 2 respectively. Bikel 1-best is (Bikel, 2004). *k*-NN is our final *k*-NN model.

With our *k*-NN model we achieve LR/LR of 89.1%/89.4% on sentences ≤ 40 words. These results show an 8% relative reduction in *f*-score error over our Model 1 baseline and a 4% relative reduction in *f*-score error over the Bikel parser. We compared the results of our *k*-NN model against the Bikel 1-best parser results using the paired *T* test where the data points being compared were the scores of each parse in the two different sets of parses. The 95% confidence interval for the mean difference between the scores of the paired sets of parses is [0.029, 0.159] with $P < .005$. Following (Collins 2000) the score of a parse takes into account the number of constituents in the gold standard parse for this sentence. These results show that using the methods presented in this paper can produce significant improvements in parser accuracy over the baseline parser.

References

Daniel M. Bikel. 2004. On the Parameter Space of Generative Lexicalized Statistical Parsing Models. *PhD thesis, University of Pennsylvania.*

Michael Collins. 1999. Head-driven statistical models for natural language processing. *PhD thesis, University of Pennsylvania.*

Michael Collins. 2000. Discriminative reranking for natural language parsing. *In Proceedings of the 7th ICML.*

Kristina Toutanova, Mark Mitchell and Christopher Manning. 2003. Optimizing Local Probability Models for Statistical Parsing. In *Proceedings of 14th ECML.*

Robust Extraction of Subcategorization Data from Spoken Language

Jianguo Li & Chris Brew
Department of Linguistics
The Ohio State University, USA
{jianguo|cbrew}@ling.ohio-state.edu

Eric Fosler-Lussier
Department of Computer Science & Engineering
The Ohio State University, USA
fosler@cse.ohio-state.edu

1 Introduction

Subcategorization data has been crucial for various NLP tasks. Current method for automatic SCF acquisition usually proceeds in two steps: first, generate all SCF cues from a corpus using a parser, and then filter out spurious SCF cues with statistical tests. Previous studies on SCF acquisition have worked mainly with written texts; spoken corpora have received little attention. Transcripts of spoken language pose two challenges absent in written texts: uncertainty about utterance segmentation and disfluency.

Roland & Jurafsky (1998) suggest that there are substantial subcategorization differences between spoken and written corpora. For example, spoken corpora tend to have fewer passive sentences but many more zero-anaphora structures than written corpora. In light of such subcategorization differences, we believe that an SCF set built from spoken language may, if of acceptable quality, be of particular value to NLP tasks involving syntactic analysis of spoken language.

2 SCF Acquisition System

Following the design proposed by Briscoe and Carroll (1997), we built an SCF acquisition system consisting of the following four components: Charniak's parser (Charniak, 2000); an SCF extractor; a lemmatizer; and an SCF evaluator. The first three components are responsible for generating SCF cues from the training corpora and the last component, consisting of the Binomial Hypothesis Test (Brent, 1993) and a back-off algorithm (Sarkar & Zeman, 2000), is used to filter SCF cues on the basis of their reliability and likelihood.

We evaluated our system on a million word written corpus and a comparable spoken corpus from BNC. For type precision and recall, we used 14 verbs selected by Briscoe & Carroll (1997) and evaluated our results against SCF entries in COMLEX (Grishman *et al*., 1994). We also calculated token recall and the results are summarized in the following table.

Corpus	Written	Spoken
type precision	93.1%	91.2%
type recall	48.2%	46.4%
token recall	82.3%	80%

Table 1: Type precision, recall and token recall

3 Detecting Incorrect SCF Cues

We examined the way segmentation errors and disfluency affects our acquisition system – the statistical parser and the extractor in particular – in proposing SCF cues and explored ways to detect incorrect SCF cues. We extracted 500 SCF cues from the ViC corpus (Pitt, *et al*, 2005) and identified four major reasons that seem to have caused the extractor to propose incorrect SCF cues: multiple utterances; missing punctuation; disfluency; parsing errors.

Error analysis reveals that segmentation errors and disfluencies cause the parser and the extractor to tend to make systematic errors in proposing SCF cues – incorrect SCF cues are likely to have an extra complement. We therefore proposed the following two sets of linguistic heuristics for automatically detecting incorrect SCF cues:

Linguistic Heuristic Set 1: The following SCF cues are extremely unlikely whatever the verb. Reject an SCF cue as incorrect if it contains the following patterns:

- [(NP) PP **NP**]: We reach out [to your friends] [**your neighbor**].
- [NP PP-to **S**]: Would I want them to say [that][to me] [**would I want them to do that to me**].
- [NP NP **S**]: They just beat [Indiana in basketball] [the- Saturday] [**I think it was um-hum**].

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 204–205,
Vancouver, October 2005.

- [**PP-p** PP-p]: He starts living [**with the**] [with the guys].

Linguistic Heuristic Set 2: The following SCF cues are all possibly valid SCFs: for SCF cues of the following type, check if the given verb takes it in COMLEX. If not, reject it:

- [(NP) **S**]: When he was dying [**what did he say**].
- [PP-to **S**]: The same thing happened [to him] [**uh he had a scholarship**].
- [(NP) **NP**]: OU had a heck of time beating [them] [**uh-hum**].
- [(NP) **INF**]: You take [the plate] from the table [**rinse them off**] and put them by the sink.

Given the utilization of a gold standard in the heuristics, it would be improper to build an end-to-end system and evaluate against COMLEX. Instead, we evaluate by seeing how often our heuristics succeed producing results agreeable to a human judge.

To evaluate the robustness of our linguistic heuristics, we conducted a cross-corpora and cross-parser comparison. We used 1,169 verb tokens from the ViC corpus and another 1,169 from the Switchboard corpus.

Cross-corpus Comparison: The purpose of the cross-corpus comparison is to show that our linguistic heuristics based on the data from one spoken corpus can be applied to other spoken corpora. Therefore, we applied our heuristics to the ViC and the Switchboard corpus parsed by Charniak's parser. We calculated the percentage of incorrect SCF cues before and after applying our linguistic heuristics. The results are shown in Table 2.

Charniak's parser	ViC	Switchboard
before heuristics	18.8%	9.5%
after heuristics	6.4%	4.6%

Table 2: Incorrect SCF cue rate before and after heuristics

Table 2 shows that the incorrect SCF cue rate has been reduced to roughly the same level for the two spoken corpora after applying our linguistic heuristics.

Cross-parser Comparison: The purpose of the cross-parser comparison is to show that our linguistic heuristics based on the data parsed by one parser can be applied to other parsers as well. To this end, we applied our heuristics to the Switchboard corpus parsed by both Charniak's parser and Bikel's parsing engine (Bikel, 2004). Again, we calculated the percentage of incorrect SCF cues before and after applying our heuristics. The results are displayed in Table 3.

Although our linguistic heuristics works slightly better for data parsed by Charniak' parser, the incorrect SCF cue rate after applying heuristics remains at about the same level for the two different parsers we used.

Switchboard	Charniak	Bikel
before heuristics	9.5%	9.2%
after heuristics	4.6%	5.4%

Table 3: Incorrect SCF cue rate before and after heuristics

4 Conclusion

We showed that it should not be assumed that standard statistical parsers will fail on language that is very different from what they are trained on. Specifically, the results of Experiment 1 showed that it is feasible to apply current SCF extraction technology to spoken language. Experiment 2 showed that incorrect SCF cues due to segmentation errors and disfluency can be recognized by our linguistic heuristics. We have shown that our SCF acquisition system as a whole will work for the different demands of spoken language.

5 Acknowledgements

This work was supported by NSF grant 0347799 to the second author, and by a summer fellowship from the Ohio State Center for Cognitive Science to the first author.

References

Biekl, D. 2004. Intricacies of Collins' Parsing Model. *Computational Linguistics*, 30(4): 470-511

Brent, M. 1993. From Grammar to Lexicon: Unsupervised Learning of Lexical Syntax. Computational Lingusitics: 19(3): 243-262

Briscoe, E. & Carroll, G. 1997. Automatic Extraction of Subcategorization from Corpora. In *Proceedings of the 5th ACL Conference on Applied Natural Language Processing*, Washington, DC. 356-363

Chaniak, E. 2000. A Maximum-Entropy-Inspired Parser. In Proceedings of the 2000 Conference of the North American Chapter of ACL. 132-139

Grishman, R., Macleod, C. & Meyers, A. 1994. COMLEX Syntax: Building a Computational Lexicon. In *Proceedings of the International Conference on Computational Lingusitics*, COLING-94, Kyoto, Japan. 268-272

Pitt, M., Johnson, K., Hume, E., Kiesling, S., Raymond, W. 2005. They Buckeye Corpus of Conversational Speech: Labeling Conventions and a Test of Transcriber Reliability. *Speech Communication*, 45: 89-95

Roland, D. & Jurafsky, D. 1998. How Verb Subcategorization Frequency Affected by the Corpus Choice. In *Proceedings of 17th International Conference on Computational Lingusitics*, 2: 1122-1128

Sarkar, A. & Zeman, D. 2000. Automatic Extraction of Subcategorization Frames for Czech. In Proceedings of the 19th International Conference on Computational Lingusitics. 691-697

Author Index