RH: A Retro Hybrid Parser

Paula S. Newman
`newmanp@acm.org`

Abstract

Contemporary parser research is, to a large extent, focused on statistical parsers and deep-unification-based parsers. This paper describes an alternative, hybrid architecture in which an ATN-like parser, augmented by many preference tests, builds on the results of a fast chunker. The combination is as efficient as most stochastic parsers, and accuracy is close and continues to improve. These results raise questions about the practicality of deep unification for symbolic parsing.

1 Introduction

The original goals of the RH parser were to obtain accurate parses where (a) application speed was needed, and (b) large amounts of annotated material for a subject idiom were not available. Additional goals that evolved were (c) that parses for particular documents could be brought to an almost arbitrary level of correctness for research purposes, by grammar correction, and (d) that information collected during parsing could be modified for an application with a modest amount of effort. Goal (a) ruled out the use of unification-based symbolic parsers, because deep unification is a relatively slow operation, no matter what amount of computational sophistication is employed. Until very recently, goal (b) ruled out stochastic parsers, but new results (McClosky et al. 2006) suggest this may no longer be the case. However, the "additional" goals still favor symbolic parsing.

To meet these goals, the RH parser combines a very efficient shallow parser with an overlay parser that is "retro", in that the grammar is related to Augmented Transition Networks (Woods, 1970), operating on the shallow-parser output. A major "augmentation" is a preference-scoring component.

Section 2 below reviews the shallow parser used, and Section 3 describes the overlay parser. Some current results are presented in section 4. Section 5 examines some closely-related work, and Section 6 discusses some implications.

2 The XIP Parser for English

XIP is a robust parser developed by Xerox Research Center Europe. It is actually a full parser that produces a tree of chunks, plus identification of (sometimes alternative) typed dependencies among the chunk heads (Ait-Mokhtar et al. 2002, Gala 2004). But because the XIP dependency analyzer for English was incomplete when RH work began, and because classic parse trees are more convenient for discourse-related applications, we focused on the chunk output.

XIP is astonishingly fast, contributing very little to RH parse time. It consists of the XIP engine, plus language-specific grammars, each consisting of: (a) a finite state lexicon producing alternative tags and morphological analyses for each token, together with subcategorization, control and (some) semantic class features, (b) a part of speech tagger, and (c) conveniently expressed, layered rule sets that perform the following functions:

- **Lexicon extension**, which adds words and adds or overrides feature information,
- **Lexical disambiguation** (including use of the tagger to provide default assignments)
- **Multi-word identification** for named entities, dates, short constructions, etc.
- **Chunking,** obtaining basic chunks such as basic adjective, adverbial, noun and prepositional phrases.
- **Dependency Analysis** (not used in RH)

All rule sets have been extended within RH development except for the dependency rule sets..

3 Overlay Parser

The overlay parser builds on chunker output to produce a single tree (figure 1) providing syntactic categories and functions, heads, and head features. The output tree requires further processing to obtain long distance dependency information, and make some unambiguous coordination adjustments

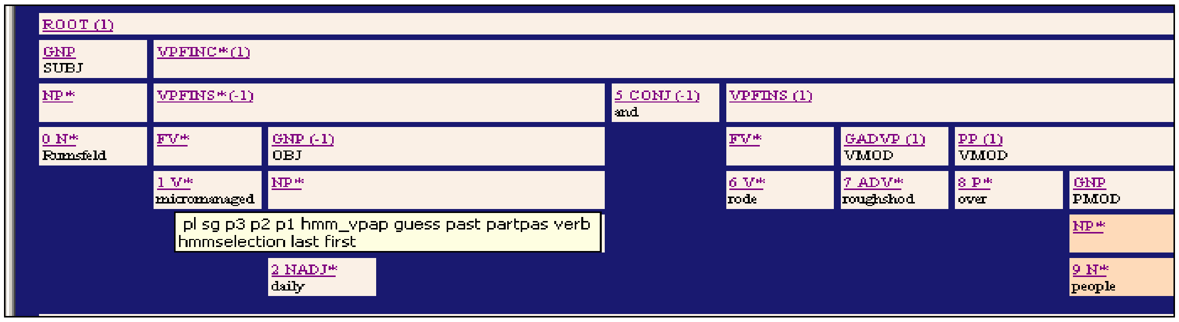

Figure 1. Output Parse Tree. * indicates head. Mouseover shows head features

Some of this has already been done in a post-parse phase. The feasibility of such post-parse deepening (for a statistical parser) is demonstrated by Cahill et al (2004).

The major parser components are a control, the ATN-like grammar networks, and collections of tests. The control is invoked recursively to build non-chunk constituents by following grammar network paths and creating output networks.

Figure 2 shows the arcs of an excerpt from a grammar network used to build a noun phrase. The Test labels on the arcs resemble specialized categories. The MetaOps (limited in the illustration to Prolog-like cuts) expedite processing by permitting or barring exploration of further ordered arcs originating at the same state.

An output network, illustrated in figure 3, mirrors the full paths traversed in a grammar network by one invocation of the control. The arcs refer either to chunks or to final states of other output networks. Output networks do not contain cycles or converging arcs, so states represent unique paths. They carry head and other path information, and a preference score. The final parser output is a single tree, derived from a highest scoring path of a topmost output network. Ties are broken by low attach considerations.

From	To	Test	Syn fun	Final?	Meta Op
S1	S1	PREADV	PRE	No	cut
S1	S2	PRON	HEAD	Yes	cut
S1	S3	PROPER	HEAD	Yes	cut
S1	S4 S7	BASENP	HEAD	Yes	cut
//After pronoun					
S2	-	REFL	REFL	Yes	cut
S2	-	PEOPLE	APPS	Yes	cut

Figure 2. Some arcs of grammar network for GNP

From	To	Cat	Synfun	Ref
OSa	OSb	NP	HEAD	NPChunk (*The park*)
OSb	OSc	PP	NMOD	Final state of PP net for (*in Paris*)

States	Score	Final?
Osa	0	No
Osb	0	Yes
OSc	1	Yes

Figure 3. Output network for "*The park in Paris*"

Each invocation of the control is given a grammar network entry state and a desired constituent category. After initializing a new output network, the arcs from the given entry state are followed. Processing an arc may begin with an optional pretest. If that succeeds, or there is no pretest, a constructive test follows. The tests are indexed by grammar network test labels, and are expressed as blocks of procedural code, for initial flexibility in determining the necessary checks.

Pretests include fast feasibility checks, and contexted checks of consistency of the potential new constituent with the current output network path. Constructive tests can make additional feasibility checks. If these checks succeed, either a chunk is returned, or the control is reentered to try to build a subordinate output network. Results are cached, to avoid repeated testing.

After a chunk or subordinate network *ON'* is returned from a constructive test, one new arc *Ai* is added to the current output network *ON* to represent each full path through *ON'*. All added arcs have the same origin state in *ON*, but unique successor states and associated preference scores. The preference score is the sum of the score at the common origin state, plus the score of the represented path in *ON'*, plus a contexted score for the alternative within *ON*. The latter is one of <-1, 0, +1>, and expresses the consistency of *Ai* with the current path with respect to dependency, coordination and apposition. Structural and punctuation aspects are also considered. Preference tests are indexed by syntactic category or syntactic function, and are organized for speed. Most tests are independent of *Ai* length, and can be applied once and the results assumed for all *Ai*.

Before a completed output network is returned, paths ending at those lower scoring final states which cannot ultimately be optimal are pruned. Such pruning is critical to efficiency.

4 Indicative Current Results

To provide a snapshot of current RH parser performance, we compare its current speed and accuracy directly to those of a widely used statistical parser, Collins model 3 (Collins, 1999), and indirectly to two other parsers. Wall Street Journal section 23 of the Penn Treebank (Marcus et al. 1994) was used in all experiments.

"Training" of the RH parser on the Wall Street Journal area (beyond general RH development) occupied about 8 weeks, and involved testing and (non-exhaustively) correcting the parser using two WSJ texts: (a) section 00, and (b) 700 sentences of section 23 used as a dependency bank by King et al. (2003). The latter were used early in RH development, and so were included in the training set.

4.1 Comparative Speed

Table 1 compares RH parser speed with Collins model 3, using the same CPU, showing the elapsed times for the entire 2416-line section 23.

The results are then extrapolated to two other parsers, based on published comparisons with Collins. The extrapolation to XLE, a mature unification-based parser that uses a disambiguating statistical post-processor, is drawn from Kaplan et al. (2004). Results are given for both the full grammar and a reduced version that omits less likely rules. The second comparison is with the fast stochastic parser by Sagae and Lavie (2005).

Summarizing these results, RH is much faster than Collins model 3 and the reduced version of XLE, but a bit slower than Sagae-Lavie.

The table also compares coverage, as percentages of non-parsed sentences. For RH this was 10% for the test set discussed below, which did not contain any training sentences, and was 10.4% for the full section 23. This is reasonable for a symbolic parser with limited training on an idiom, and better than the 21% reported for XLE English.

	Time	No full parse
Sagae/ Lavie	***~ 4 min***	*1.1%*
RH parser	5 min	10%
Collins m3	16 min	**.6%**
XLE full	*~80 minutes*	~21%
XLE reduced	*~24 minutes*	unknown

Table 1: Speeds and *Extrapolated speeds*

	Fully accurate	F-score	Avg cross brackets
Sagae/Lavie	unknwn	86%	unknwn
Collins Lbl	33.6%	88.2%	1.05
CollinsNoLbl	35.4%	**89.4 %**	1.05
RH NoLbl	**46%**	86 %	**.59**

Table 2. Accuracy Comparison

4.2 Comparative Acccuracy

Table 2 primarily compares the accuracy of the Collins model 3 and RH parsers. The entries show the proportion of fully accurate parses, the f-score average of bracket precision and recall, and average crossing brackets, as obtained by EVALB (Sekine and Collins, 1997). The RH f-score is currently somewhat lower, but the proportion of fully correct parses is significantly higher.

This data may be biased toward RH, because, of necessity, the test set used is smaller, and a different bracketing method is used. For Collins model 3, the entries show both labeled and unlabeled results for all of WSJ section 23. The Collins results were generated from the bracketed output and Penn Treebank gold standard files provided in a recent Collins download.

But because RH does not generate treebank style tags, the RH entries reflect a test only on a random sample of 100 sentences from the 1716 sentences of section 23 not used as "training" data, using a different, available, gold standard creation and bracketing method. In that method (Newman, 2005), parser results are produced in a "TextTree" form, initially developed for fast visual review of parser output, and then edited to obtain gold standard trees. Both sets of trees are then bracketed by a script to obtain, e.g.,

```
    {An automatic transformation
         {of parse trees}
         {to text trees}}
    {can expedite
         {parser output reviews}}
```

For non-parsed sentences in the parser outputs, brackets are applied to the chunks. EVALB is then used to compare the two sets of bracketed results.

Accuracy for XLE is not given, because the results reported by Kaplan et al. (2004) compare labeled functional dependencies drawn from LFG f-structures with equivalents derived automatically from Collins outputs. (All f-scores are <= 80%).

5 Related Work

Several efforts combine a chunker with a dependency analyzer operating on the chunks, including XIP itself. The XIP dependency analyzer is very fast, but we do not have current coverage or accuracy data for XIP English.

Other related hybrids do not build on chunks, but, rather, adjust full parsers to require or prefer results consistent with chunk boundaries. Daum et al. (2003) use chunks to constrain a WCDG grammar for German, reducing parse times by about 2/3 (but the same results are obtained using a tagger alone). They estimate that an ideal chunker would reduce times by about 75%. No absolute numbers are given. Also, Frank et al. (2003) use a German topological field identifier to constrain an HPSG parser. They show speedups of about 2.2 relative to a tagged baseline, on a corpus whose average sentence length is about 9 words.

6 Discussion

We have shown that the RH hybrid can compete with stochastic parsers in efficiency and, with only limited "training" on an idiom, can approach them in accuracy. Also, the test organization prevents speed from degrading as the parser is improved.

The method is significant in itself, but also leads to questions about the advantages of deep-unification-based parsers for practical NLP. These parsers are relatively slow, and their large numbers of results require disambiguation, e.g., by corpus-trained back-ends. They do provide more information than RH, but there is much evidence that the additional information can be obtained by rapid analysis of a single best parse. Also, it has never been shown that their elegant notations actually facilitate grammar development and maintenance. Finally, while unification grammars are reversible for use in generation, good generation methods remain an open research problem.

References

Salah Aït-Mokhtar, Jean-Pierre Chanod, and Claude Roux. 2002. Robustness beyond shallowness: incremental deep parsing, *Natural Language Engineering* 8:121-144, Cambridge University Press.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Josef van Genabith, and Andy Way. 2004. Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations, In *Proc ACL'04.* Barcelona

Michael Collins. 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, University of Pennsylvania.

Michael A. Daum, Kilian A. Foth, and Wolfgang Menzel. 2003. Constraint-based integration of deep and shallow parsing techniques. In *Proc EACL'03*, Budapest

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer and Ulrich Schaefer. 2003. Integrated Shallow and Deep Parsing: TopP Meets HPSG. In *Proc ACL'2003,* Sapporo

Nuria Gala. 2004. Using a robust parser grammar to automatically generate UNL graphs. *In Proc Workshop on Robust Methods for Natural Language Data at COLING'04,* Geneva

Ronald M. Kaplan, Stephan Riezler, Tracy H. King, John T. Maxwell, Alex Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proc HLT/NAACL'04,* Boston, MA.

Tracy H. King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proc Workshop on Linguistically Interpreted Corpora, (LINC'03),* Budapest

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and Self-Training for Parser Adaptation. In *Proc ACL'06.* Sydney

Paula Newman. 2005. TextTree Construction for Parser and Grammar Development. In *Proc. Workshop on Software at ACL'05* Ann Arbor, MI. Available at http://www.cs.columbia.edu/nlp/acl05soft/

Satoshi Sekine and Michael Collins. 1997. *EvalB.* Available at http://nlp.cs.nyu.edu/evalb

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proc. 9th Int'l Workshop on Parsing Technologies*. Vancouver

William Woods. 1970. Transition network grammars for natural language analysis. *Communications of the ACM* 13(10), 591-606