

A Distributed Architecture for Interactive Parse Annotation

Baden Hughes

Department of Computer Science
and Software Engineering
The University of Melbourne
Victoria 3010, Australia

badenh@csse.unimelb.edu.au

James Haggerty, Saritha Manickam

Joel Nothman, James R. Curran
School of Information Technologies
University of Sydney
NSW 2006, Australia

{j.h,jnothman,saritha}@student.usyd.edu.au
james@it.usyd.edu.au

Abstract

In this paper we describe a modular system architecture for distributed parse annotation using *interactive correction*. This involves interactively adding constraints to an existing parse until the returned parse is correct. Using a *mixed initiative approach*, human annotators interact live with distributed CCG parser servers through an annotation GUI. The examples presented to each annotator are selected by an *active learning* framework to maximise the value of the annotated corpus for machine learners. We report on an initial implementation based on a distributed workflow architecture.

1 Introduction

Annotating sentences with parse trees is perhaps the most complex and intensive linguistic annotation. The time and expense of developing parsed corpora is almost prohibitive. As a result there are only a small number of such corpora, including the Penn Treebank (Marcus et al., 1994), the German TiGer Corpus (Skut et al., 1997) and more recently the LinGO Redwoods Treebank (Oepen et al., 2002). These corpora are also limited in size, typically around one million words.

Unfortunately, the statistical approaches to parsing which have been most successful rely heavily on both the quality and quantity of annotated resources. Also, these approaches are very sensitive to the statistical properties of the corpus, and so a parser trained on one genre may perform badly on another (Gildea, 2001).

Another major problem with parsed corpora is that they must, at least to some extent, follow a particular syntactic theory or formalism. This is a major difficulty for two reasons: firstly, it means we need separate annotated corpora for each formalism; and secondly, it means that comparing parser evaluations across formalisms is difficult.

Fully automated conversion of parse trees between formalisms is difficult because each analyses certain constructs in idiosyncratic ways. An example is CCGbank (Hockenmaier and Steedman, 2002), a treebank of Combinatory Categorical Grammar (Steedman, 2000) derivations which were converted semi-automatically from the Penn Treebank trees. The result still required laborious editing to produce idiomatic CCG derivations (Hockenmaier, 2003).

We intend to create a new corpus of CCG derivations on a wide range of text. We face three key problems: 1) selecting sentence to annotate which creates the most *useful* corpus for statistical parsers. 2) maximising the *annotator efficiency* and *minimising error*; 3) allowing distributed annotators to *share expertise*.

The selection problem is addressed using *active learning* (AL). Active learning involves computing which training instances provide the most new information to one (or more) machine learners (Cohn et al., 1995; Dagan and Engelson, 1995). The annotators become oracles answering specific queries posed by the learners.

The annotation problem is addressed by *interactive correction* of the output of our statistical CCG parser. This is similar to the *discriminant* strategy employed for Redwoods annotation (Oepen et al., 2002) but generalises to grammars where parse enumeration is infeasible. Annotators interactively add constraints to the parser which returns the most probable parse satisfying the constraints.

The distributed expertise problem is addressed using a *workflow manager*. Annotators will be able to add comments and queries to derivations and have them sent to (potentially remote) experienced annotators for verification. The workflow manager will also handle scheduling for the active learning infrastructure.

This paper describes the architecture and initial implementation of a system which addresses these problems for distributed parse annotation.

The WSJ is a publication that I enjoy reading

$\frac{NP}{N}$ $\frac{N}{N}$ $\frac{(S[decl]\backslash NP)/NP}{NP/N}$ $\frac{NP/N}{N}$ $\frac{(NP\backslash NP)/(S[decl]/NP)}{NP}$ $\frac{NP}{NP}$ $\frac{(S[decl]\backslash NP)/(S[ng]\backslash NP)}{(S[ng]\backslash NP)/NP}$

Figure 1: Example sentence with CCG lexical categories

2 Related Work

Since annotating parse trees is a significant bottleneck in NLP there have been several attempts to make the process more efficient. In this work we exploit two approaches: using choice points to select the correct parse and using active learning to select sentences to parse.

A *discriminant* is a property that distinguishes between a set of interpretations. They can be designed for linguistic non-experts (Carter, 1997). In the Redwoods project, the annotator is presented with discriminants on the trees themselves which eventually lead to the correct HPSG parse (Oepen et al., 2002). These discriminants are calculated from the enumerated set of all parses. Unfortunately, our automatically extracted CCG grammar produces far too many derivations (billions) for enumeration to be feasible.

Baldrige and Osborne (2004) demonstrate how active learning (AL) can be used to significantly reduce the annotation cost for annotating text with HPSG parses. They compare random selection with approaches based on uncertainty sampling (Cohn et al., 1995) and committee based sampling (Dagan and Engelson, 1995) and demonstrate a reduction in annotation effort of 72%. A key point that Baldrige and Osborne identify is that each sentence cannot be treated as equally difficult to annotate. Tang et al. (2002) also evaluate AL on statistical parsing and find the total cost of annotation can be reduced to one third. Finally, Becker et al. (2005) compares bootstrapping techniques including AL for developing new named entity corpora.

Formally introduced in Day et al. (1997), *mixed initiative annotation* (where the division of labour between computational facilities and human effort is coordinated for increased efficiency) has become an increasingly common methodology for the preparation of large corpora. Typically however, mixed initiative approaches have largely decoupled human and machine effort, even for larger scale tasks.

Extending the mixed initiative model specifically to a distributed environment, Hughes and Bird (2003) offer a model for the type of solution we implement here. Additionally, the ar-

chitecture advocated by Curran (2003) allows us flexibility in designing individual components of this system independently, and then marshalling them into a single application instance.

Experiments with distributed NLP tasks of building n-gram language models (Hughes et al., 2004a) and generalised textual indexing and linguistically motivated retrieval (Hughes et al., 2004b) are broadly indicative of other work in this area. To date, however we are not aware of any work in this vein specifically involving mixed initiative annotation.

3 Combinatory Categorical Grammar

Combinatory Categorical Grammar (Steedman, 2000) is a type-driven lexicalized theory of grammar based on categorial grammar. Almost all of the grammatical information in CCG is represented in the categories assigned to each word, which are either simple atomic categories (e.g. NP) or complex functor categories (e.g. $(S[decl]\backslash NP)/NP$ a transitive declarative verb). An example sentence with lexical categories is shown in Figure 1. These categories are combined together according to a small number of *combinatory rules*.

The set of these lexical categories is obtained from CCGbank (Hockenmaier and Steedman, 2002; Hockenmaier, 2003), a corpus of CCG normal-form derivations derived semi-automatically from the Penn Treebank. The category set consists of those category types which occur at least 10 times in sections 2-21 of CCGbank, which results in a set of 409 categories. Clark and Curran (2004a) demonstrates that this relatively small set has high coverage on unseen data and can be used to create a robust and accurate parser. In order to obtain semantic representations for a particular formalism, only 409 categories have to be annotated.

3.1 CCG Parsing

In our system we are using the C&C CCG parser (Clark and Curran, 2004b), which uses a log-linear model over normal-form derivations to select an analysis. The parser takes a POS tagged sentence as input with a set of one or more categories assigned to each word. A CCG supertagger (Clark and Curran, 2004a) assigns the lexi-

cal categories, using a log-linear model to identify the most probable categories. Clark and Curran (2004a) show how dynamic use of the supertagger — starting off with a small number of categories assigned to each word and gradually increasing the number until an analysis is found — can lead to a highly efficient and robust parser.

The parser uses the CKY chart-parsing algorithm from Steedman (2000). The combinatory rules used by the parser are functional application (forward and backward), generalised forward composition, backward composition, generalised backward-crossed composition, and type raising. There is also a coordination rule which conjoins categories of the same type.

3.2 CCG Supertagging

Lexicalised grammar formalisms, such as LTAG and CCG, assign one or more syntactic structures to each word which are then manipulated by the parser. Supertagging was introduced for LTAG to increase parsing efficiency by reducing the number of structures assigned to each word (Bangalore and Joshi, 1999).

The parser model parameters are estimated using a discriminative method, that is, one which requires statistics across all incorrect parses for a sentence as well as the correct parse. Since an automatically extracted CCG grammar can produce an extremely large number of parses, the use of a supertagger is crucial in limiting the total number of parses for the training data to a computationally manageable number.

The supertagger is also crucial for increasing the speed of the parser. We have shown that spectacular increases in speed can be obtained, without affecting accuracy or coverage, by tightly integrating the supertagger with the CCG grammar and parser (Clark and Curran, 2004a). To achieve maximum speed, the supertagger initially assigns only a small number of CCG categories to each word, and the parser only requests more categories from the supertagger if it cannot provide an analysis. Clark et al. (2004) has demonstrated that annotating new data at just the lexical category level can be enough to significantly improve the performance of a parser on a new domain.

3.3 Interactive Correction

For a given sentence, the automatically extracted grammar can produce a very large number of derivations. Clark and Curran (2004b) describes how a packed chart can be used to

efficiently represent the derivation space, and also efficient algorithms for finding the most probable derivation. Unfortunately, this massive derivation space means it is not possible to enumerate all parses, so the discriminant strategy for interactive annotation outlined previous is infeasible.

We therefore introduce the idea of *interactive correction* where the parser is given a number of constraints by the annotator. Rather than enumerate the parse, the process only involves finding the most probable parse that satisfies the expressed constraints. This can be performed efficiently as part of the dynamic programming algorithm which finds the highest probability derivation.

Given that the CCG categories contain so much information we expect that it will only require annotators to constrain the lexical categories on a few words to reach a correct parse.

4 Example Use Case

The annotation process begins by the annotator requesting a sentence to annotate. The active learning component determines which sentence from a large corpus of raw sentences may provide the most *new* information if it were to be annotated. AL can be very computationally intensive process and so will only occur after a given number of sentences have been annotated. The AL component will return a queue of sentences that then scheduled to be annotated.

The annotator will receive the top sentence on the queue along with the most probable derivation for that sentence. They can add the following constraints to a given sentence:

force a specific lexical category

ban a current lexical category

ban a current non-leaf category

force a specific chart span

ban a current chart span

Adding one (or more) new constraints will cause the derivation to be returned which satisfies the existing and new constraints (if such a derivation exists). This process continues until the correct derivation is reached and the parse is checked in as correct. Once enough new annotated sentences have been completed the AL component regenerates a new queue of sentences based on the retrained statistical parser model.

An alternative case is that the annotator is not sure about the correct derivation for the sentence. They can then annotate the derivation

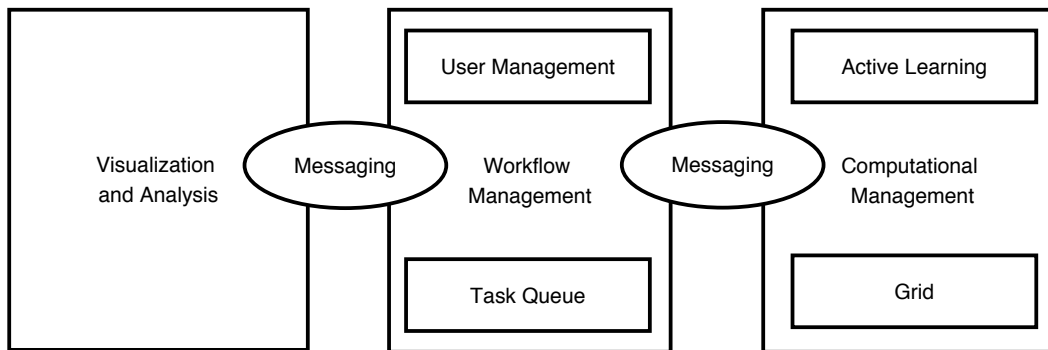


Figure 2: System Architecture

with a comment/question and it will be scheduled on the queue for other more experienced annotators. The experienced annotator will see the constraints and the comments added by the original annotator. They can make a decision or propagate it to some other annotator. Once a decision has been reached the information is returned to all annotators. This process is handled by the workflow manager.

5 System Architecture

The system architecture for distributed annotation and parsing with active learning can be seen in Figure 1. The Visualization and Analysis module provides the end user interface by which a human annotator can review and revise the parser output. The actual content rendered by this module is provided by the Workflow Management module.

The Workflow Management module has three main roles: first to interact with the Visualization and Analysis, providing parses to be visualised and refined; second to manage the user and tasks in the process of analysis; and third to interact with the Computational Management module by instantiating the active learning framework for incremental parsing of the corpus data, and subsequent grid execution.

The Computational Management module has two sub-modules. The Active Learning sub-module allows for incremental application of refined parses as training data for subsequent iterations of the parser. The Grid sub-module handles low level execution including the queuing, dispatch and execution of analysis tasks, and fetching the results from the distributed computation environment.

Having described the high level architecture of the system, we now turn to an in depth discussion of each of the components in turn.

6 Visualization and Analysis Module

The visualisation GUI is implemented in wxPython (Dunn, 2005), an extension of the cross-platform GUI toolkit wxWidgets (Smart et al., 2005) for Python. wxWidgets is particularly notable for its use of native graphical components for a given operating system platform, allowing the interface a native look and feel when run on Windows, Mac or Linux environments.

Both the GUI and the export to file functionality are built on a flexible cross-platform code base. As can be seen in Figure 3 an initial implementation already succeeds in displaying the CCG parser output in a user-friendly form similar to that used by (Steedman, 2000) and widely adopted as a standard format.

To facilitate the re-use of the rendered parse tree, our GUI uses the ReportLab Toolkit (Rep, 2005) to facilitate export to common vector graphics formats such as PDF, PostScript, PNG and SVG. These are useful for inclusion in publications and presentations.

By using basically the same rendering code, consistent output is provided to the GUI and the export formats, including plain-text. Each provides a Canvas object whose role is simply to provide metric information for displaying text on that output device, and then to place the text at given locations. All calculations for positioning text nodes are done external to the canvas, so new output formats can easily be supported. Similarly, if a user selects a change in font-styling, this is reflected in all graphical export formats and on-screen.

The graphical interface provides some interactivity in order to assist a user in viewing, manipulating and annotating a parsed sentence. Most importantly, an annotator can change or constrain the available categories for sentence

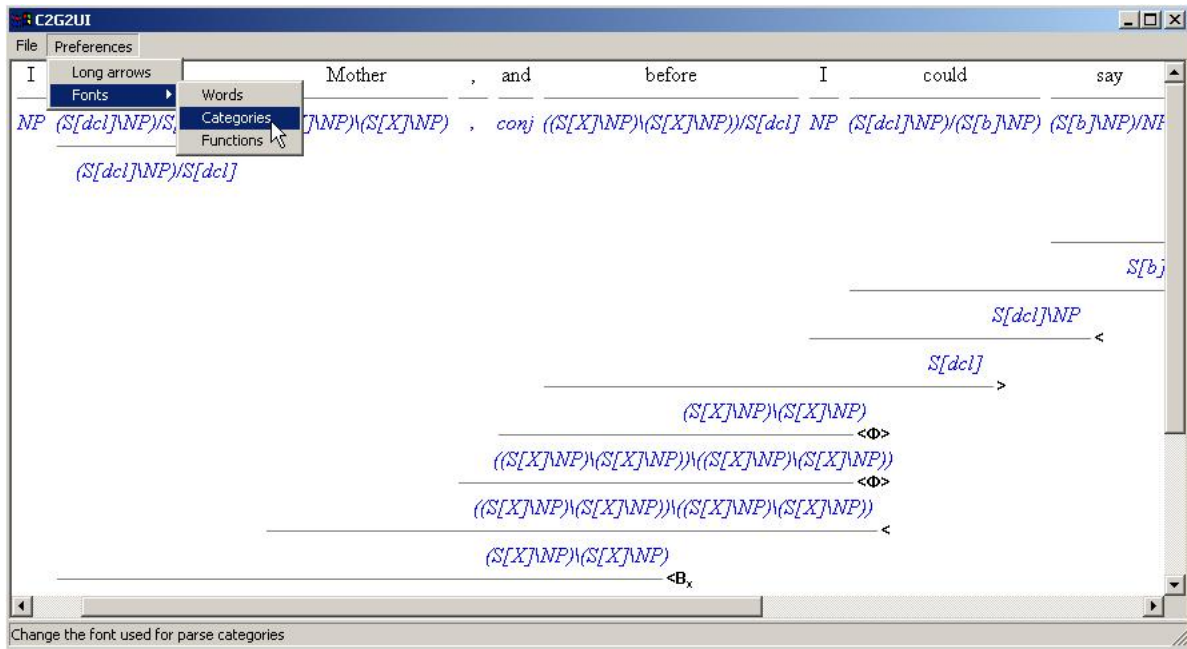


Figure 3: Screenshot from GUI

constituents. As soon as these constraints are added, they are passed back in real-time to the Workflow Management module, and the GUI is updated to reflect the results of those changes. Thus, in almost all cases, correct parses can be generated without the need for the user to laboriously construct an entire derivation.

To simplify the viewing of large parse trees, a user has the facility to ‘collapse’ chosen sections of the tree with a simple point-and-click operation. Hiding most collapsed words from view and only showing the derived category can significantly reduce the horizontal and vertical space occupied by the parse image. This feature is particularly useful when an annotator needs to focus on a particular section of a given parse: once a certain partial derivation is checked, it may be collapsed and will remain fixed there through other parse modifications.

Collectively these features should make the job of annotators far less painful. The graphical rendering of the CCG parser’s output makes the incorrect grouping of words obvious, and by using the parser in collaboration with the user as described above, annotating a sentence correctly will usually be a matter of seconds rather than minutes.

7 Workflow Management Module

The basic workflow of the system is as follows. Sentences are parsed from a corpus by the CCG

parser on the grid, results being added to review stack. A user logs in and starts a session, requesting the next review parse. The user reviews parse, and either confirms parse, or modifies and submits a revised parse or promotes the candidate parse to new reviewer. Accepted parses are sent back to the active learner for subsequent retraining of the CCG parser.

As mentioned earlier, the Workflow Management module has two interfaces: one to the Visualization and Analysis module, and the other to the Execution module; as well as an internal user and task management function. We will first discuss the latter, before returning to the interfaces themselves. It is however, important to note that the workflow here is analytical, as distinct from computational.

7.1 User Management Sub-module

The user registry allows for the tracking of user (i.e. annotator) names, together with corresponding passwords and user level attributes. In addition, a log is kept of the activity of each user, in particular annotation times, which are useful for monitoring the effectiveness of the interactive correction approach.

7.2 Task Queue Sub-module

The task queue sub-module is basically a parse review queue contains the list of sentences pending review together with an user allocation, and

a parse/sentence status (pending, reviewed).

7.3 Interface to Visualization and Analysis

To facilitate communication the Simple Object Access Protocol (SOAP) (Gudgin et al., 2003) has been used to implement a lightweight interface. The SOAP implementation supports 5 basic functions:

- `authenticate_user` implements a lightweight user authentication protocol based on a username and password;
- `submit_accepted_parse` is used when the current parse and constraints are acceptable, and is parameterised by the sentence ID, a parse ID, and the set of constraints with any associated commentary.
- `submit_uncertain_parse` is used when the current parse is not fully understood by the user, and a second opinion is required. It is parameterised by the sentence ID, a parse ID, and the set of constraints with any associated commentary.
- `get_next_parse` is the stack retrieval method, used to retrieve the next parsed sentence in the individual user queue. It returns a the next sentence ID, sentence, parse ID, and set of constraints and associated commentary.
- `get_modified_parse` allows the user to get a subsequent sentence and parse matching a revised set of constraints. It takes a parse ID and a set of constraints.

7.4 Interface to Computational Management

Again to facilitate communication the Simple Object Access Protocol (SOAP) (Gudgin et al., 2003) has been used to implement a lightweight interface between the Workflow Management module and the Computational Management module. The SOAP implementation supports 2 basic functions: `submit_sentence_for_parsing` and `get_next_sentence_for_review`.

- `submit_sentence_for_parsing` is used for transferring the sentences from the reviewed sentences queue to the active learning framework; and
- `get_next_sentence_for_review` is used for transferring the parsed sentences from the

active learning framework to the Workflow Management module

8 Computational Management Module

As mentioned earlier, the Computational Management Module consists of two further sub-modules, one for active learning and the other for computational grid interaction management.

8.1 Active Learning Sub-Module

Our implementation of active learning involves a variety of differently parameterised instances of the CCG parser, with the view that an evaluation of each model will identify the best parse and constraints for a given sentence.

This module is instantiated based on some threshold - either time based (e.g. once every 24 hours) or queue based (e.g. when there are 100 modified parses). This asynchronous server side component allows discontinuity between the user-centric review process and the computational impact of large scale re-parsing.

Our active learner uses committee-based sampling (Dagan and Engelson, 1995) using differing supertagging and parsing statistical models as committee members. Where there are many annotation options, the most popular alternatives from the committee will be passed to annotators to help select the correct annotation more efficiently. This will minimise the cognitive load of selecting between too many alternatives.

8.2 Grid Sub-Module

The purpose of the Grid component is to manage all aspects of interaction with the distributed computational environment in which the parser itself is running. The Grid sub-module handles low level execution including the queuing, dispatch and execution of analysis tasks, and fetching the results.

The experimental environment is setup with compute infrastructure in Sydney and in Melbourne. At the Sydney node, the system environment is a cluster of 9 dual-CPU Linux machines running MPI middleware. At the Melbourne node the system environment is a cluster of machines running Linux, managed by the NorduGrid Advanced Resource Connector (ARC). On each node, the CCG parser is installed.

The CCG toolkit is installed on the respective clusters and simply instantiated by the active learning framework as threshold boundaries are reached.

The Grid sub module selects the relevant compute node for execution of the current parse task. (In the simplest case, perhaps a round robin approach to selecting the compute facility for subsequent re-parsing runs would appear to be sufficient, although for more intensive human annotation sessions, batch mode parser execution with probe-based load measurement is probably desirable for a scalable and robust implementation). A job description is then created specific to the node requirements. The job is then passed to the head node of the cluster.

9 Discussion

The previous sections describe an architecture for distributed, computationally intensive, mixed initiative linguistic analysis. We believe this contribution is notable for a number of reasons including:

- a completely modular systems architecture, in contrast with tightly bound end-to-end systems which typically dominate this application space;
- coordinated yet distinctly decoupled computational and human effort, allowing both parties to contribute to the overall effort with maximum efficiency;
- re-usable, open sourced components which are sufficiently flexible to allow other interested parties to build from an established base, rather than the ground up
- an instantiation of service oriented NLP via open standards

We are motivated to modularise the overall system as much as possible to allow maximum flexibility for future extensions. In particular, our selection of the CCG parser is relatively arbitrary; any parser should be able to be swapped in for the CCG parser (e.g. an HPSG parser) with the only overhead being support for parallelisation and an API which can be functionally mapped to our SOAP based interface. Correspondingly, we envisage that the GUI component should be generalised sufficiently to allow for the rendering of a variety of different parse tree representations.

10 Status

At the time of writing the status of the components required is as follows:

- Visualization and Analysis module

A prototype GUI has been implemented which can render CCG derivations in several formats. Lexical categories can now be modified in the GUI and the parse regenerated directly with the new constraints. This does not currently use SOAP for getting next parse.

- Workflow Management module

The 7 SOAP methods (5 for Visualization and Analysis module interface, and 2 for Computational Management module interface) are implemented as a CGI application in Python. Basic user and task management implemented.

- Computational Management module

The parameterisation and brokering framework for grid execution is deployed in production.

Production grids are operational (building over existing infrastructure) at both Melbourne and Sydney sites. The CCG parser on these systems; in the Melbourne case, the active learning framework can be instantiated by a web services / SOAP based interface to NorduGrid's native job brokering system.

11 Conclusion

We have proposed an architecture for performing distributed annotation of CCG derivations. This architecture attempts to solve three key problems in the efficient preparation of large scale NLP resources: 1) selecting sentence to annotate which creates the most *useful* corpus for statistical parsers. 2) maximising the *annotator efficiency* and *minimising error*; 3) allowing distributed annotators to *share expertise*.

We have attempted to address these problems using a combination of machine learning techniques and grid computing infrastructure. In particular, Active Learning will identify the best sentences to annotate; interactive correction will make the most of our annotators time; and our workflow manager will allow (even remote) annotators to share their expertise more effectively.

While our implementation is relatively immature at this point, we believe the architecture proposed in this paper, along with the specific components, will be able to be reused in multiple contexts.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful feedback, and to David Vadas and Toby Hawker for testing the CCG GUI. This work has been supported by the Australian Research Council under Discovery Project DP0453131.

References

- Jason Baldridge and Miles Osborne. 2004. Active learning and the total cost of annotation. In *Proceedings of the EMNLP Conference*, pages 9–16, Barcelona, Spain.
- Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.
- Marcus Becker, Ben Hachey, Beatrice Alex, and Claire Grover. 2005. Optimising selective sampling for bootstrapping named entity recognition. In *Proc. of the ICML-2005 Workshop on Learning with Multiple Views*.
- David Carter. 1997. The treebanker: a tool for supervised training of parsed corpora. In *Proc. of the Workshop on Computational Environments for Grammar Development and Language Engineering*, Madrid, Spain.
- Stephen Clark and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proc. of the 20th COLING*, pages 282–288, Geneva, Switzerland.
- Stephen Clark and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the ACL*, pages 103–110.
- Stephen Clark, Mark Steedman, and James R. Curran. 2004. Object-extraction and question-parsing using CCG. In *Proc. of the EMNLP Conference*, pages 111–118, Barcelona, Spain.
- David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. 1995. Active learning with statistical models. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 705–712. MIT Press.
- James Curran. 2003. Blueprint for a high performance nlp infrastructure. In *Proc. of the Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS)*, Edmonton, Canada.
- Ido Dagan and Sean P. Engelson. 1995. Committee-based sampling for training probabilistic classifiers. In *Proc. of the ICML*, pages 150–157.
- David Day, John Aberdeen, Lynette Hirschman, Robyn Kozierok, Patricia Robinson, and Marc Vilain. 1997. Mixed-initiative development of language processing systems. In *Proc. of the 5th conference on Applied NLP*, pages 348–355.
- Robin Dunn. 2005. wxPython toolkit. <http://www.wxpython.org>.
- Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the EMNLP Conference*, pages 167–202, Pittsburgh, PA.
- Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. 2003. SOAP version 1.2 part 1: Messaging framework. <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>.
- Julia Hockenmaier and Mark Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the 3rd LREC Conference*, pages 1974–1981, Las Palmas, Spain.
- Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorical Grammar*. Ph.D. thesis, University of Edinburgh.
- Baden Hughes and Steven Bird. 2003. Grid-enabling natural language engineering by stealth. In *Proc. of the Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS)*, Edmonton, Canada.
- Baden Hughes, Steven Bird, Ewan Klein, and Haejoong Lee. 2004a. Experiments with data intensive nlp on a computational grid. In *Proc. of the 2004 Hong Kong International Workshop on Language Technology*.
- Baden Hughes, Srikumar Venugopal, and Rajkumar Buyya. 2004b. Grid-based indexing of a newswire corpus. In *Proc. of the 5th IEEE Workshop on Grid Computing*.
- 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.
- Stephan Open, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods Treebank: Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1253–1257, Taipei, Taiwan.
2005. ReportLab toolkit. http://www.reportlab.org/rl_toolkit.html.
- Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the 5th ACL Conference on Applied NLP*, pages 88–95, Washington, DC.
- Julian Smart, Kevin Hock, and Stefan Csomor. 2005. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall.
- Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.
- Min Tang, Xiaoqing Luo, and Salim Roukos. 2002. Active learning for statistical natural language parsing. In *Proc. of the 40th Annual Meeting of the ACL*, pages 120–127, Philadelphia, PA USA.