Log-Linear Models for Wide-Coverage CCG Parsing

Stephen Clark and James R. Curran
School of Informatics
University of Edinburgh
2 Buccleuch Place, Edinburgh. EH8 9LW
{stephenc,jamesc}@cogsci.ed.ac.uk

Abstract

This paper describes log-linear parsing models for Combinatory Categorial Grammar (CCG). Log-linear models can easily encode the long-range dependencies inherent in coordination and extraction phenomena, which CCG was designed to handle. Log-linear models have previously been applied to statistical parsing, under the assumption that all possible parses for a sentence can be enumerated. Enumerating all parses is infeasible for large grammars; however, dynamic programming over a packed chart can be used to efficiently estimate the model parameters. We describe a parellelised implementation which runs on a Beowulf cluster and allows the complete WSJ Penn Treebank to be used for estimation.

1 Introduction

Statistical parsing models have recently been developed for Combinatory Categorial Grammar (CCG, Steedman (2000)) and used in wide-coverage parsers applied to the WSJ Penn Treebank (Clark et al., 2002; Hockenmaier and Steedman, 2002). An attraction of CCG is its elegant treatment of coordination and extraction, allowing recovery of the long-range dependencies inherent in these constructions. We would like the parsing model to include long-range dependencies, but this introduces problems for generative parsing models similar to those described by Abney (1997) for attribute-value grammars; hence Hockenmaier and Steedman do not include such dependencies in their model, and Clark et al. include the dependencies but use an inconsistent model. Following Abney, we propose a log-linear framework which incorporates long-range dependencies as features without loss of consistency.

Log-linear models have previously been applied to statistical parsing (Johnson et al., 1999; Toutanova et al., 2002; Riezler et al., 2002; Osborne, 2000). Typically, these approaches have enumerated all possible parses for model estimation and finding the most probable parse. For grammars extracted from the Penn Treebank (in our case CCGbank (Hockenmaier, 2003)), enumerating all parses is infeasible. One approach to this problem is to sample the parse space for estimation, e.g. Osborne (2000). In this paper we use a dynamic programming technique applied to a packed chart, similar to those proposed by Geman and Johnson (2002) and Miyao and Tsujii (2002), which efficiently estimates the model parameters over the complete space without enumerating parses. The estimation method is similar to the inside-outside algorithm used for estimating a PCFG (Lari and Young, 1990).

Miyao and Tsujii (2002) apply their estimation technique to an automatically extracted Tree Adjoining Grammar using Improved Iterative Scaling (IIS, Della Pietra et al. (1997)). However, their model has significant memory requirements which limits them to using 868 sentences as training data. We use a parallelised version of Generalised Iterative Scaling (GIS, Darroch and Ratcliff (1972)) on a Beowulf cluster which allows the complete WSJ Penn Treebank to be used as training data.

This paper assumes a basic knowledge of CCG; see Steedman (2000) and Clark et al. (2002) for an introduction.

2 The Grammar

Following Clark et al. (2002), we augment CCG lexical categories with head and dependency information. For example, the extended category for *persuade* is as follows:

$$\text{persuade} := ((S[dcl]_{persuade}\backslash NP_1)/(S[to]_2\backslash NP_X))/NP_{X,3} \quad (1)$$

The feature $[dcl]$ indicates a declarative sentence; the resulting $S[dcl]$ is headed by *persuade*; and the numbers indicate dependency relations. The variable X denotes a head, identifying the head of the infinitival complement's subject with the head of the object, thus capturing the object control relation. For example, in *Microsoft persuades IBM to buy Lotus*, *IBM* fills the subject slot of *buy*.

Formally, a dependency is defined as a 5-tuple: $\langle h_f, f, s, h_a, l\rangle$, where h_f is the head word of the functor, f is the functor category (extended with head and dependency information), s is the argument slot, and h_a is the head word of the argument. The l is an additional field used to encode whether the dependency is long-range. For example, the dependency encoding *Lotus* as the object of *bought* (as in *IBM bought Lotus*) is represented as follows:

$$\langle bought, (S[dcl]_{bought}\backslash NP_1)/NP_2, 2, Lotus, \text{ null}\rangle \quad (2)$$

If the object has been extracted using a relative pronoun with the category $(NP\backslash NP)/(S[dcl]/NP)$ (as in *the company that IBM bought*), the dependency is as follows:

$$\langle bought, (S[dcl]_{bought}\backslash NP_1)/NP_2, 2, company, *\rangle \quad (3)$$

where $*$ is the category $(NP\backslash NP)/(S[dcl]/NP)$ assigned to the relative pronoun. A dependency structure is simply a set of these dependencies.

Every argument in every lexical category is encoded as a dependency. Unlike Clark et al., we do not require dependencies to be always marked on atomic categories. For example, the marked up category for *about* (as in *about 5,000 pounds*) is:

$$(N_X/N_X)_Y/(N/N)_{Y,1} \quad (4)$$

If *5,000* has the category $(N_X/N_X)_{5,000}$, the dependency relation marked on the $(N/N)_{Y,1}$ argument in (4) allows the dependency between *about* and *5,000* to be captured.

Clark et al. (2002) give examples showing how heads can fill dependency slots during a derivation, and how long-range dependencies can be recovered through unification of co-indexed head variables.

3 Log-Linear Models for CCG

Previous parsing models for CCG include a generative model over *normal-form* derivations (Hockenmaier and Steedman, 2002) and a conditional model over dependency structures (Clark et al., 2002). We follow Clark et al. in modelling dependency structures, but, unlike Clark et al., do so in terms of derivations. An advantage of our approach is that the model can potentially include derivation-specific features in addition to dependency information. Also, modelling derivations provides a close link between the model and the parsing algorithm, which makes it easier to define dynamic programming techniques for efficient model estimation and decoding[1], and also apply beam search to reduce the search space.

The probability of a dependency structure, $\pi \in \Pi$, given a sentence, S, is defined as follows:

$$P(\pi|S) = \sum_{d\in\Delta(\pi,S)} P(d,\pi|S) \quad (5)$$

where $\Delta(\pi, S)$ is the set of derivations for S which lead to π and Π is the set of dependency structures. Note that $\Delta(\pi, S)$ *includes* the non-standard derivations allowed by CCG. This model allows the possibility of including features from the non-standard derivations, such as features encoding the use of type-raising or function composition.

A log-linear model of a parse, $\omega \in \Omega$, given a sentence S, is defined as follows:

$$P(\omega|S) = \frac{1}{Z_S}\prod_i \mu_i^{f_i(\omega)} \quad (6)$$

This model can be applied to any kind of parse, but for this paper a parse, ω, is a $\langle d,\pi\rangle$ pair (as given in (5)). The function f_i is a *feature* of the parse

[1] We use the term *decoding* to refer to the process of finding the most probable dependency structure from a packed chart.

which can be any real-valued function over the space of parses Ω. In this paper $f_i(\omega)$ is a count of the number of times some dependency occurs in ω. Each feature f_i has an associated *weight* μ_i which is a parameter of the model to be estimated. Z_S is a normalising constant which ensures that $P(\omega|S)$ is a probability distribution:

$$Z_S = \sum_{\omega' \in \rho(S)} \prod_i \mu_i^{f_i(\omega')} \quad (7)$$

where $\rho(S) \subseteq \Omega$ is the set of possible parses for S.

The advantage of a log-linear model is that the features can be arbitrary functions over parses. This means that any dependencies – including overlapping and long-range dependencies – can be included in the model, irrespective of whether those dependencies are independent.

The theory underlying log-linear models is described in Della Pietra et al. (1997) and Berger et al. (1996). Briefly, the log-linear form in (6) is derived by choosing the model with maximum entropy from a set of models that satisfy a certain set of constraints (Rosenfeld, 1996). The constraints are that, for each feature f_i:

$$\sum_{\omega,S} \tilde{P}(S)P(\omega|S)f_i(\omega) = \sum_{\omega,S} \tilde{P}(\omega, S)f_i(\omega) \quad (8)$$

where the sums are over all possible parse-sentence pairs and $\tilde{P}(S)$ is the relative frequency of sentence S in the data. The value on the left of (8) is the expected value of f_i according to the model, $E_p f_i$, and the value on the right is the empirical expected value of f_i, $E_{\tilde{p}} f_i$.

Estimating the parameters of a log-linear model requires the values in (8) to be calculated for each feature. Calculating the empirical expected values requires a treebank of CCG derivations plus dependency structures. For this we use CCGbank (Hockenmaier, 2003), a corpus of normal-form CCG derivations derived semi-automatically from the Penn Treebank. Following Clark et al., gold standard dependency structures are obtained for each derivation by running a dependency-producing parser over the derivations. The empirical expected value of a feature f_i is calculated as follows:

$$E_{\tilde{p}} f_i = \frac{1}{N} \sum_{j=1}^{N} f_i(\omega_j) \quad (9)$$

where $\omega_1 \dots \omega_N$ are the parses in the training data (consisting of a normal-form derivation plus dependency structure) and $f_i(\omega_j)$ is the number of times f_i appears in parse ω_j.[2]

Parameter estimation also requires calculation of expected values of the features according to the model, $E_p f_i$. This requires summing over all parses (derivation plus dependency structure) for the sentences in the data, a difficult task since the total number of parses can grow exponentially with sentence length. For some sentences in CCGbank, the parser described in Section 6 produces trillions of parses. The next section shows how a packed chart can efficiently represent the parse space, and how GIS applied to the packed chart can be used to estimate the parameters.

4 Packed Charts

Geman and Johnson (2002) have proposed a dynamic programming estimation method for packed representations of unification-based parses. Miyao and Tsujii (2002) have proposed a similar method for *feature forests* which they apply to the derivations of an automatically extracted Tree-Adjoining Grammar. We apply Miyao and Tsujii's method to the derivations and dependency structures produced by our CCG parser.

The dynamic programming method relies on a *packed chart*, in which chart entries of the same *type* in the same cell are grouped together, and back pointers to the daughters keep track of how an individual entry was created. The intuition behind the dynamic programming is that, for the purposes of building a dependency structure, chart entries of the same type are equivalent. Consider the following composition of *will* with *buy* using the forward composition rule:

$$\begin{array}{ccc} & ((S[dcl]_{will}\backslash NP)/NP) & \\ & / \quad \backslash & \\ ((S[dcl]_{will}\backslash NP)/(S[b]\backslash NP)) & & ((S[b]_{buy}\backslash NP)/NP) \end{array}$$

The type of the resulting chart entry is determined by the CCG category plus heads, in this case $((S[dcl]_{will}\backslash NP)/NP)$, plus the dependencies yet to be filled. The dependencies are not shown, but there

[2] An alternative is to use feature counts from *all* derivations leading to the gold standard dependency structure, including the non-standard derivations, to calculate $E_{\tilde{p}} f_i$.

are two subject dependencies on the first *NP*, one encoding the subject of *will* and one encoding the subject of *buy*[3], and there is an object dependency on the second *NP* encoding the object of *buy*. Entries of the same type are identical for the purposes of creating new dependencies for the remainder of the parsing.

Any rule instantiation[4] used by the parser creates both a set of dependencies and a set of features. For the previous example, one dependency is created:

$$\langle \text{will}, (S[dcl]_{will} \backslash NP_{X,1})/(S[b]_2 \backslash NP_X), 2, \text{buy} \rangle$$

This dependency will be a feature created by the rule instantiation. We also use less specific features, such as the dependency with the words replaced by POS tags. Section 7 describes the features used.

The feature forests of Miyao and Tsujii are defined in terms of *conjunctive* and *disjunctive* nodes. For our purposes, a conjunctive node is an individual entry in a cell, including the features created when the entry was derived, plus pointers to the entry's daughters. A disjunctive node represents an equivalence class of nodes in a cell, using the type equivalence relation described above. A conjunctive node results from either the combination of two disjunctive nodes using a binary rule, e.g. forward composition; or results from a single disjunctive node using a unary rule, e.g. type-raising; or is a leaf node (a word plus lexical category).

Features in the model can only result from a single rule instantiation. It is possible to define features covering a larger part of the dependency structure; for example we might encode all three elements of the triple in a PP-attachment as a single feature. The disadvantage of using such features is that this reduces the efficiency of the dynamic programming. Note, however, that the equivalence relation defining disjunctive nodes takes into account unfilled dependencies, which may be long-range dependencies being "passed up" the derivation tree. This means that long-range dependencies can be features in our model, even though the lexical items involved may be far apart in the sentence.

[3]In this example, the co-indexing of heads in the markedup category for *will* ($(S[dcl]_{will} \backslash NP_{X,1})/(S[b]_2 \backslash NP_X)$) ensures the subject dependency for *buy* is "passed up" to the subject *NP* of the resulting category.

[4]By *rule instantiation* we mean the local tree arising from the application of a CCG combinatory rule.

The packed structure we have described is an example of a *feature forest* (Miyao and Tsujii, 2002), defined as follows:

A *feature forest* Φ is a tuple $\langle C, D, R, \gamma, \delta \rangle$ where

- C is a set of conjunctive nodes;
- D is a set of disjunctive nodes;
- $R \subseteq D$ is a set of root disjunctive nodes;[5]
- $\gamma : D \rightarrow 2^C$ is a conjunctive daughter function;
- $\delta : C \rightarrow 2^D$ is a disjunctive daughter function.

For each feature function $f_i : \Omega \rightarrow \mathcal{N}$, there is a corresponding feature function $f_i : C \rightarrow \mathcal{N}$ which counts the number of times f_i appears on a particular conjunctive node.[6] The value of f_i for a parse is then the sum of the values of f_i for each conjunctive node in the parse.

5 Estimation using GIS

GIS is a very simple algorithm for estimating the parameters of a log-linear model. The parameters are initialised to some arbitrary constant and the following update rule is applied until convergence:

$$\mu_i^{(t+1)} = \mu_i^{(t)} \left(\frac{E_{\tilde{p}} f_i}{E_{p^{(t)}} f_i} \right)^{\frac{1}{C}} \tag{10}$$

where (t) is the iteration index and the constant C is defined as $\max_{\omega,S} \sum_i f_i(\omega)$. In practice C is maximised over the sentences in the training data. Implementations of GIS typically use a "correction feature", but following Curran and Clark (2003) we do not use such a feature, which simplifies the algorithm.

Calculating $E_{p^{(t)}} f_i$ requires summing over all derivations which include f_i for each packed chart in the training data. The key to performing this sum efficiently is to write the sum in terms of *inside* and *outside* scores for each conjunctive node. The inside and outside scores can be defined recursively, as in the inside-outside algorithm for PCFGs. If the inside score for a conjunctive node c is denoted ϕ_c, and the

[5]Miyao and Tsujii have a single root conjunctive node; the disjunctive root nodes we define correspond to the roots of CCG derivations.

[6]The value of $f_i(c)$ for $c \in C$ will typically be 0 or 1, but it is possible for the count to be greater than 1.

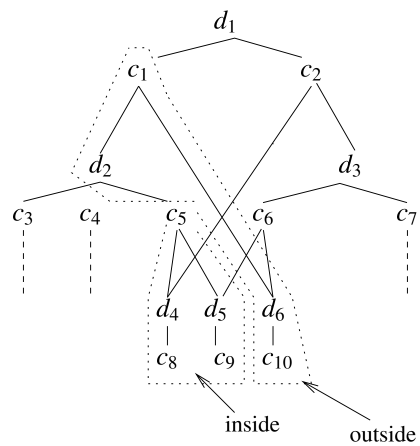

Figure 1: Example feature forest

outside score denoted ψ_c, then the expected value of f_i can be written as follows:[7]

$$E_p f_i = \sum_S \tilde{P}(S) \frac{1}{Z_S} \sum_{c \in C_s} f_i(c)\ \phi_c\ \psi_c \qquad (11)$$

where C_s is the set of conjunctive nodes for S.

Consider the example feature forest in Figure 1. The figure shows the nodes used to calculate the inside and outside scores for conjunctive node c_5. The inside score for a disjunctive node, ϕ_d, is the sum of the inside scores for its conjunctive node daughters:

$$\phi_d = \sum_{c \in \gamma(d)} \phi_c \qquad (12)$$

The inside score for a conjunctive node, ϕ_c, can then be defined recursively:

$$\phi_c = \prod_{d \in \delta(c)} \phi_d \prod_i \mu_i^{f_i(c)} \qquad (13)$$

The intuition for calculating outside scores is similar, but a little more involved. The outside score for a conjunctive node, ψ_c, is the outside score for its disjunctive node mother:

$$\psi_c = \psi_d \ \text{ where } c \in \gamma(d) \qquad (14)$$

The outside score for a disjunctive node is a sum over the mother nodes, of the product of the outside score of the mother, the inside score of the sister, and the feature weights on the mother.[8] For example, the outside score of d_4 in Figure 1 is the sum of the following two values: the product of the outside score of c_5, the inside score of d_5 and the feature weights at c_5; and the product of the outside score of c_2, the inside score of d_3 and the feature weights at c_2. The recursive definition is as follows. The outside score for a root disjunctive node is 1, otherwise:

$$\psi_d = \sum_{\{c | d \in \delta(c)\}} \left(\psi_c \prod_{\{d' | d' \in \delta(c), d' \neq d\}} \phi_{d'} \prod_i \mu_i^{f_i(c)} \right) \qquad (15)$$

The normalisation constant Z_S is the sum of the inside scores for the root disjunctive nodes:

$$Z_S = \sum_{d_r \in R} \phi_{d_r} \qquad (16)$$

In order to calculate inside scores, the scores for daughter nodes need to be calculated before the scores for mother nodes (and vice versa for the outside scores). This can easily be achieved by ordering the nodes in the bottom-up CKY parsing order.

Note that the inside-outside approach can be combined with any maximum entropy estimation procedure, such as those evaluated by Malouf (2002).

Finally, in order to avoid overfitting, we use a Gaussian prior on the parameters of the model (Chen and Rosenfeld, 1999), which requires a slight modification to the update rule in (10). A Gaussian prior also handles the problem of "pseudo-maximal" features (Johnson et al., 1999).

6 The Parser

The parser is based on Clark et al. (2002) and takes as input a POS-tagged sentence with a set of possible lexical categories assigned to each word. The supertagger of Clark (2002) provides the lexical categories, with a parameter setting which assigns around 4 categories per word on average. The parsing algorithm is the CKY bottom-up chart-parsing algorithm described in 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. Restrictions are placed on some of the rules, such as that given by

[7] The notation is taken from Miyao and Tsujii (2002).

[8] Miyao and Tsujii (2002) ignore the feature weights on the mother, but this ignores some of the probability mass for the outside (at least for the feature forests we have defined).

Steedman (2000, p.62) for backward-crossed composition.

Type-raising is applied to the categories *NP*, *PP* and $S[adj]\backslash NP$ (adjectival phrase), and is implemented by adding the relevant set of type-raised categories to the chart whenever an *NP*, *PP* or $S[adj]\backslash NP$ is present. The sets of type-raised categories are based on the most commonly used type-raising rule instantiations in sections 2-21 of CCGbank, and contain 8 type-raised categories for *NP* and 1 each for *PP* and $S[adj]\backslash NP$.

The parser also uses a number of lexical rules and punctuation rules. These rules are based on those occurring roughly more than 200 times in sections 2-21 of CCGbank. An example of a lexical rule used by the parser is the following, which takes a passive form of a verb and creates a nominal modifier:

$$S[pss]\backslash NP \;\Rightarrow\; NP_X\backslash NP_{X,1} \tag{17}$$

This rule is used to create *NP*s such as *the role played by Kim Cattrall*. Note that there is a dependency relation on the resulting category; in the previous example *role* would fill a nominal modifier dependency headed by *played*.

Currently, the only punctuation marks handled by the parser are commas, and all other punctuation is removed after the supertagging phase. An example of a comma rule is the following:

$$S_X/S_X \;\;, \;\Rightarrow\; S_X/S_X \tag{18}$$

This rule takes a sentential modifier followed by a comma (for example *Currently ,* in the sentence above in the text) and returns a sentential modifier of the same type.

The next section describes the efficient implementation of the parser and model estimator.

7 Implementation

7.1 Parser Implementation

The non-standard derivations allowed by CCG, together with the wide coverage grammar, result in extremely large charts. This means that efficient implementation of the parsing process is imperative for performing large-scale experiments.

The packed chart prevents combinatorial explosion in the number of category combinations by grouping equivalent categories into a single entry. The speed of the parser is heavily dependent on the efficiency of equivalence testing, and category unification and construction. These are performed efficiently by always creating categories in a canonical form which can then be compared rapidly using hash functions over categories.

The parser produces a packed chart from which the most probable dependency structure can be recovered. Since the same dependency structure can be generated by more than one derivation, a dependency structure's score is the sum of the log-linear scores for each derivation. Finding the structure with the highest score is not trivial, since filled dependencies are only stored at the conjunctive nodes where they are created. This means that a dependency appearing in a structure can be created in different parts of the chart for different derivations. We solve this in practice using a hash function over dependencies, which can be used to quickly determine whether two derivations lead to the same structure. For each node in the chart, we can keep track of the derivation leading to the set of dependencies with the highest score for that node.

7.2 Data Generation

Data for model estimation is created in two steps. First, the parser is run over the normal-form derivations in Sections 2-21 of CCGbank outputting the corresponding dependencies and other features. The features used in our preliminary implementation are as follows:

- dependency features;
- lexical category features;
- root category features.

Dependency features are 5-tuples as defined in Section 2. Further dependency features are formed by substituting POS tags for the words, which leads to a total of 4 features for each dependency. Lexical category features are word category pairs on the leaf nodes and root features are head-word category pairs on root nodes. Extra features are formed by replacing words with their POS tags. The total number of features is 817,658, but we reduce this to 243,603 by only including features which appear at least twice in the data.

The second step of data generation involves using the parser to create a feature forest for each sentence, using the feature set extracted from CCGbank. The parser is interrupted if a sentence takes longer than 60 seconds to process or if more than 500,000 conjunctive nodes are created in the chart. If this occurs, the process is repeated but with a smaller number of categories assigned to each word by the supertagger. Approximately 93% of the sentences in sections 2-21 can be processed in this way, giving 36,400 training sentences. Creating the forests takes approximately one hour using 40 nodes of our Beowulf cluster, and produces 19.9 GB of data.

7.3 Estimation

The parse forests regularly represent trillions of possible parses for a sentence. The estimation process involves summing feature weights over all these parses, a total which cannot be represented using double precision arithmetic (limited to less than 10^{308}). Our implementation uses the sum, rather than product, form of (6), so that logarithms can be used to avoid numerical overflow. For converting the sum of products in Equation 15 to log space, we use a technique commonly used in speech recognition (p.c. Simon King).

We have implemented a parallel version of our GIS code using the MPICH library (Gropp et al., 1996), an open-source implementation of the Message Passing Interface (MPI) standard. MPI parallel programming involves explicit synchronisation and information transfer between the parallel processes using messages. It is ideal for development of parallel programs for cluster architectures.

GIS over parse forests is straightforward to parallelise. The parse forests are divided among the machines in the cluster (in our current implementation, each machine receives 979 forests). Each machine calculates the inside and outside scores for each node in the parse forest and updates the estimated feature expectations. The feature expectations are then summed across all of the machines using a global operation (called a *reduce* operation). Every machine receives this sum which is then used to calculate the normal GIS weight update. In our preliminary tests, each process used approximately 750 MB of RAM, giving a total usage of 30 GB across the cluster. One iteration of GIS takes approximately 1 minute. Given the large number of features, we estimate at least 1,000 iterations will be needed for convergence.

	GIS - CCG	IIS - TAG
number of features	243,603	5,715
number of sentences	36,400	868
avg. num. of nodes	52,000	17,412
memory usage	30 GB	1.5 GB
disk usage	19.9 GB	–

Table 1: Results compared with Miyao and Tsujii

8 Conclusions and Further Work

Table 1 gives the overall statistics for the model estimation process, and compares them with Miyao and Tsujii (2002). These numbers represent the largest-scale parsing model of which we are aware. Parsing and model estimation on this scale introduce a number of interesting theoretical and computational challenges. We have demonstrated how packed charts and feature forests can be combined to meet the theoretical challenges. We have also described an MPI implementation of GIS which solves the computational challenges. These techniques are necessary for discriminative estimation techniques applied to wide-coverage parsing.

We have just begun the process of evaluating parsing performance using the same test data as Clark et al. (2002). We are especially interested in the effectiveness of incorporating long-range dependencies as features, which CCG was designed to handle and for which we expect a log-linear model to be particularly effective.

Acknowledgements

We would like to thank Mark Steedman, Julia Hockenmaier, Jason Baldridge, David Chiang, Yusuke Miyao, Mark Johnson, Yuval Krymolowski, Tara Murphy and the anonymous reviewers for their helpful comments. This research is supported by EPSRC grant GR/M96889, and a Commonwealth scholarship and a Sydney University Travelling scholarship to the second author.

References

Steven 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.

Stanley Chen and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical report, Carnegie Mellon University, Pittsburgh, PA.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Meeting of the ACL*, pages 327–334, Philadelphia, PA.

Stephen Clark. 2002. A supertagger for combinatory categorial grammar. In *Proceedings of the TAG+ Workshop*, pages 19–24, Venice, Italy.

James R. Curran and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL (to appear)*, Budapest, Hungary.

J. N. Darroch and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 19(4):380–393.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Meeting of the ACL*, pages 279–286, Philadelphia, PA.

W. Gropp, E. Lusk, N. Doss, and A. Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Meeting of the ACL*, pages 535–541, University of Maryland, MD.

K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Workshop on Natural Language Learning*, pages 49–55, Taipei, Taiwan.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*, San Diego, CA.

Miles Osborne. 2000. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 586–592, Saarbrücken, Germany.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the ACL*, pages 271–278, Philadelphia, PA.

Ronald Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187–228.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

Kristina Toutanova, Christopher Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria.