CCG parsing with one syntactic structure per n-gram

Tim Dawborn and **James R. Curran**
School of Information Technologies
University of Sydney
NSW, 2006, Australia
{tdaw3088,james}@it.usyd.edu.au

Abstract

There is an inherent redundancy in natural languages whereby certain common phrases (or n-grams) appear frequently in general sentences, each time with the same syntactic analysis. We explore the idea of exploiting this redundancy by pre-constructing the parse structures for these frequent n-grams. When parsing sentences in the future, the parser does not have to re-derive the parse structure for these n-grams when they occur. Instead, their pre-constructed analysis can be reused. By generating these pre-constructed databases over WSJ sections 02 to 21 and evaluating on section 00, a preliminary result of no significant change in F-score nor parse time was observed.

1 Introduction

Natural language parsing is the task of assigning syntactic structure to text. Initial parsing research mostly relied on manually constructed grammars. Statistical parsers have been able to achieve high accuracy since the creation of the Penn Treebank (Marcus et al., 1993); a corpus of Wall Street Journal text used for training. Statistical parsers are typically inefficient, parsing only a few sentences per second on standard hardware (Kaplan et al., 2004). There has been substantial progress on addressing this issue over the last few years. Clark and Curran (2004) presented a statistical CCG parser, C&C, which was an order of magnitude faster than those analysed in Kaplan et al. (2004). However the C&C parser is still limited to around 25 sentences per second.

This paper investigates whether the speed of statistical parsers can be improved using a novel form of caching. Currently, parsers treat each sentence independently, despite the fact that some phrases are constantly reused. We propose to store analyses for common phrases, instead of re-computing their syntactic structure each time the parser encounters them.

Our first idea was to store a single, spanning analysis for frequent n-grams. However, the most frequent n-grams often did not form constituents. Given that n-gram distributions are very long-tailed, this meant that the constituent n-grams covered only a small percentage of n-grams in the corpus.

We then turned our attention to the n-grams that were not forming constituents. First, we found that some actually *should* form constituents. However, the structure of noun-phrases in the Penn Treebank is underspecified, leading to incorrect derivations in the C&C parser's training corpus (Hockenmaier, 2003). Secondly, we investigated whether the spurious ambiguity of CCG derivations could be exploited to force frequent n-grams to compose into constituents, while still producing a semantically equivalent derivation. Here we encountered problems using the composition rule to create new constituents. Some of these problems were due to further issues with the analyses in the corpus, while others were due to the ambiguity of the n-grams.

The sparsity of n-grams in a corpus of this size meant that we had very few caching candidates to work with. Our approach may be more successful when the caching process is performed using a data set.

2 Background

We are interested in storing the parse structure for common n-grams, so that the analysis can be reused across multiple sentences. In a way, this is an extension of an important innovation in parsing: the CKY chart parsing algorithm (Younger, 1967). Our proposal is an attempt to memoise sections of the chart across multiple sentences.

Most constituency parsers use some form of chart for constructing a derivation, so our investigation could have begun with a number of different parsers. We decided to use the C&C parser (Clark and Curran, 2007) for the following reasons. First, the aim of the caching we are proposing is to improve the speed of a parser. It makes sense to look at a parser that has already been optimised, to ensure that we do not demonstrate an improvement that could have been achieved using a much simpler solution. Secondly, there are aspects of the parser's grammar formalism, Combinatory Categorial Grammar, that are relevant to the issues we want to consider.

2.1 Chart Parsing

The chart is a triangular hierarchical structure used for storing the nodes in a parse tree, as seen in Figure 1. A chart for a sentence consisting of n tokens contains $\frac{n(n+1)}{2}$ *cells*, represented as squares in the figure. Each cell in the chart contains the parse of a contiguous *span* or sequence of tokens of the sentence. As such, a cell stores the root nodes of all possible parse trees for the tokens which the cell covers. This coverage is called the *yield* of that node. This is illustrated as the linked-list style data structure highlighted as being the contents of cell $(1, 3)$ in Figure 1. The cell (p, s) in the chart contains all possible parses for all of the tokens in the range $[p, p+s)$ for a given sentence. The chart is built from the bottom up, starting with constituents spanning a single token, and then increasing the span to cover more tokens, until the whole sentence is covered.

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a lexicalised grammar formalism. This means that each word in a sentence is assigned a composite object that reflects its function in the derivation. In CCG, these objects are called lexical categories.

Categories can be built recursively from atomic

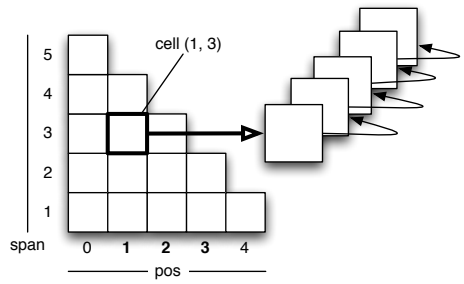

Figure 1: An illustration of the chart data structure used in parsing algorithms such as CKY

units, such as S (sentence), N (noun), and NP (noun phrase). Recursive construction of categories means that very few atomic units need to be used. For instance, there is no atomic category for a determiner in CCG. Instead, a determiner is a function which maps from a noun to a noun phrase.

Similarly, verbs are functions from some set of arguments to a complete sentence. For example, the transitive verb `like` would be assigned the category $(S\backslash NP)/NP$. Here, the slashes indicate the directionality of arguments, stating that an NP object is expected to the right, and an NP subject is expected to the left. An example CCG derivation containing the transitive verb `like` is:

$$\dfrac{\dfrac{\dfrac{\texttt{I}}{NP} \quad \dfrac{\texttt{like}}{(S[dcl]\backslash NP)/NP} \quad \dfrac{\dfrac{\texttt{the}}{NP[nb]/N} \quad \dfrac{\texttt{cat}}{N}}{NP[nb]}>}{S[dcl]\backslash NP}<}{S[dcl]}<$$

This derivation uses the rules of forward and backward application to build the representation of the sentence. Most of the information is contained in the lexical categories.

2.2 The C&C Parser

The C&C parser makes use of this property of CCG by dividing the parsing problem into two phases, following (Bangalore and Joshi, 1999). First, a *supertagger* proposes a set of likely categories for each token in the sentence. The parser then attempts to build a spanning analysis from the proposed categories, using the modified CKY algorithm described in Steedman (2000). The supertagging phase dramatically reduces the search space the parser must

explore, making the C&C parser very efficient.

3 Motivation

It is important to note that the concepts motivating this paper could be applied to any grammar formalism. However, our experiments were conducted using CCG and the C&C parser for a number of reasons, which are outlined throughout the paper.

In order for our "one structure per n-gram" idea to work in practice, the parsed data must possess two properties. Firstly, there must be a small number of n-grams which account for a large percentage of the total n-grams in the corpus. If this property was not present, this this would imply that most of the n-grams within the text appear very infrequently. As a result, the size of the database containing the memoised analyses would grow in size, as there are no n-grams which clearly are more useful to memoise than others. The result of this would be that the time taken to load the analyses from the database would exceed the time taken to let the parser construct a derivation from scratch.

The second property is that the most frequent n-grams in the corpus must have on average very few distinct analyses. If the most frequent n-grams in the corpus all occurred with a large number of different analyses, then every time we see these frequent n-grams in the future, these multiple analyses will all have to be loaded up from the database. This again would result in more time taken in the database loading than letting the parser construct the derivation from scratch. If the most frequent n-grams in the corpus thus only occur with a very small number of analyses, then the time taken to load the pre-constructed structures should be less than the time the parser will take to construct the derivation from scratch.

4 Analysis

By analysing all of the n-grams within sections 02 to 21 of CCGbank for varying n, we were able to show that, under a very basic analysis, CCGbank satisfies both the properties discussed in Section 3. The results of this analysis can be seen in Table 1.

One interesting result here is the average number of derivations varying n-grams occur with. On its first attempt at parsing a sentence, the C&C parser assigns on average 1.27 CCG categories per word (Clark and Curran, 2007). Since the average number of derivations for varying sized n-grams is less than the ambiguity introduced during the first attempt at the parsing process, this process of inserting pre-built chart structures can potentially decrease the overall parsing time, as the pre-built structures would introduce less ambiguity to the overall parse compared to what the parser would provide normally.

n-gram size	2	3	4
Avg number derivations	1.19	1.09	1.04
Always form constituents	23%	10%	5%
Never form a constituent	73%	89%	93%

Table 1: Statistics about varying sized n-gram in CCGbank sections 02 to 21

bigram	# No	# Yes	# Uniq
the company	8	1157	1
a share	3	1082	7
New York	4	868	7
a year	34	572	9
do n't	0	474	9
the market	37	410	1
did n't	0	378	11
is n't	1	367	21
The company	0	359	1
does n't	0	328	10

Table 2: Constituent statistics about the 10 most frequent bigrams in CCGbank 02 to 21 which form constituents the majority of the time

4.1 Constituents

The first idea explored is how well can we do by storing only n-grams which primarily form constituents. Table 2 shows the 10 most frequent bigrams in CCGbank sections 02 to 21 which primarily form constituents. The columns show the number of times the n-gram was seen not forming and not forming a constituent, as well as the number of unique constituent-forming analyses formed.

A number of interesting observations can be made here. Firstly, the number of times these bigrams occur drops off very quickly, with the 4th most frequent bigram appearing just under half the number

of times the most frequent bigram occurs. This drop off contradicts our first desirable property for the corpus, that there should be a large number of frequent n-grams.

Observing the numbers in the last column of Table 2, it is easily seen that only 3 of the top 10 bigrams occur with less than 5 unique derivations, which goes against our second desirable property, that the most frequent n-grams occur with very few unique derivations.

These two factors indicate that an approach which persists only constituent-forming n-grams in these databases will not perform well, as neither of the two properties discussed in Section 3 are fulfilled.

4.2 Non-constituents

Section 4.1 showed that an approach to this problem which utilises only constituent-forming n-grams most likely will not produce the desired speed boost due to the properties mentioned in Section 3 not being upheld.

The next natural direction to take is to the storing of analyses for the non-constituent-forming n-grams. The use of CCG *type raising* and *composition* allow us to store non-constituent-forming analyses in these databases for n-grams, yet still be able to use these derivations later on to form correct semantically correct spanning analyses. For example, one of the frequent non-constituent-forming occurrences of the phrase `of the` in CCGbank is

$$\dfrac{\dfrac{\texttt{of} \quad \dfrac{\texttt{the} \quad \texttt{company}}{N}}{NP}>}{NP\backslash NP}>$$

`The` is forward applied to `company` before `of` can be joined with `the`. Instead, we could construct the following derivation and insert it into the pre-constructed database

$$\dfrac{\texttt{of} \quad \texttt{the}}{(NP\backslash NP)/N}\,{>}\mathbf{B}$$

Here we use CCG forwards composition to combine `of` and `the` into a constituent-forming analysis. This chart structure could be reused with CCG forwards application to construct a span of the original phrase in the following manner

$$\dfrac{\dfrac{\texttt{of} \quad \texttt{the} \quad \texttt{company}}{(NP\backslash NP)/N}\,{>}\mathbf{B}}{NP\backslash NP}>$$

Using the forward composed version of the bigram `of the`, an analysis for the whole phrase was still able to be constructed, even though in the original derivation `of` and `the` did not form a constituent. This technique of utilising CCG forward composition and type raising allows us to add these n-grams which primarily do not form constituents, into the database.

4.2.1 Prepositional Phrase Attachment

This technique does not work all of the time, however it does work for many cases. One situation where this technique does not work is is with prepositional phrase attachment. The correct CCG derivation for the phrase `on the king of England` is

$$\dfrac{\dfrac{\dfrac{\mathrm{X} \quad \dfrac{\texttt{on} \quad \dfrac{\texttt{the king} \quad \texttt{of England}}{NP}<}{NP\backslash NP}>}{NP}<}{}}{}$$

If we were to use in this example the same forwards composed derivation of the bigram `of the` as described earlier for the bigram `on the`, the wrong analysis would be constructed.

$$\dfrac{\dfrac{\mathrm{X} \quad \dfrac{\texttt{on} \quad \dfrac{\texttt{the} \quad \texttt{king}}{N}>}{NP\backslash NP} \quad \texttt{of England}}{NP}<}{NP}<$$

While an NP was still the resultant overall category assigned to the phrase, the internal noun phrases are incorrect; the named entity `the king of England` is not represented within this incorrect derivation.

From an implementation point of view, being able to construct and use this forward composed parse structure for `of the` involves violating one of the normal-form constraints proposed in Eisner (1996) to eliminate CCG's "spurious ambiguity". The constraint which was violated states that the left child

of forward application cannot be the result of forward composition, as is the case in our previous example. The C&C parser implements these Eisner constraints, and as such a special rule was added to the parser to allow any chart structures which were loaded from a pre-constructed database to violate the Eisner constraints.

4.2.2 Coordination

In CCG parsing, commas can be parsed in one of two ways depending on their semantic role in the sentence. They are either used for coordination or they are absorbed. Consider the CCG derivation for the sentence shown in Figure 2. The second comma between `England` and `owned` is absorbed, as shown in the second last line of the derivation. The first comma, however, between `George` and `the king of England`, is used to express apposition. Apposition in CCG is represented using the same coordination structure which `and` uses; the conjoining combinator. This combinator is denoted as $conj$ or $\mathbf{\Phi}$ in CCG. The type signature of this combinator is

$$X \ conj \ X' \Rightarrow_{\mathbf{\Phi}} X''$$

stating that the CCG category has to be the same on both sides of a $conj$, and when the functor is invoked, the resultant category is the same. Our n-gram pre-construction attempts to memoise analyses based purely on the tokens of n-grams. Because comma appears as a $conj$, we are unable to use any n-grams which contain commas in our database, as at the token level it is not possible to determine if the comma will be absorbed or will be used in apposition.

4.3 Statistics

Table 3 shows the 15 most frequent bigrams in CCGbank sections 02 to 21. The first thing to note about this table is that only two of the top 15 most frequent bigrams primarily form a constituent, again leading to a conclusion that using only constituent-forming bigrams is not the correct approach to the problem. The second point to observe is that seven out of these 15 bigrams contain a comma, which as described in Section 4.2.2, implies these cannot be used in our database.

The Σ column shows an accumulative sum of the number of tokens covered in sections 00 to 21 just by using the bigrams in the table. The coverage figures shown in the neighbouring column show this sum as a percentage of the total number of tokens in sections 00 to 21. This shows that by considering just the 15 most frequent bigrams, a coverage of 6.5% of the total number of tokens has been achieved. If a trend like this continues linearly down this list of frequency sorted bigrams and a pre-constructed analysis for the first 1000 bigrams could be memoised, for example, there is a great potential for the parse time to be improved.

5 Evaluation

The effect of these n-gram databases on the parsing process is evaluated in terms of the overall parsing time, as well as the accuracy of the resultant derivations. The accuracy is measured in terms of F-score values for both labelled and unlabelled dependencies when evaluated against the predicate-argument dependencies in CCGbank (Clark and Hockenmaier, 2002). The parsing times reported do not include the time to load the grammar, statistical models, or our database.

6 Implementation

6.1 Data

The models used by the C&C parser for our experiments were trained using two different corpora. The WSJ models were trained using the CCG version of the Penn Treebank, CCGbank (Hockenmaier, 2003; Hockenmaier and Steedman, 2007), which is available from the Linguistic Data Consortium[1]. The second corpus is a version of CCGbank where the noun phrase bracketing has been corrected (Vadas and Curran, 2008; Vadas, 2009).

6.2 Tokyo Cabinet

Tokyo Cabinet[2] is an open source, lightweight database API which provides a number of different database implementations, including a hash database, B+ tree, and a fixed-length key database. Our experiments used Tokyo Cabinet to store the pre-constructed n-grams because of its ease of use, speed, and maximum database size (8EB). A large

[1] `http://ldc.upenn.edu/`

[2] `http://tokyocabinet.sourceforge.net/`

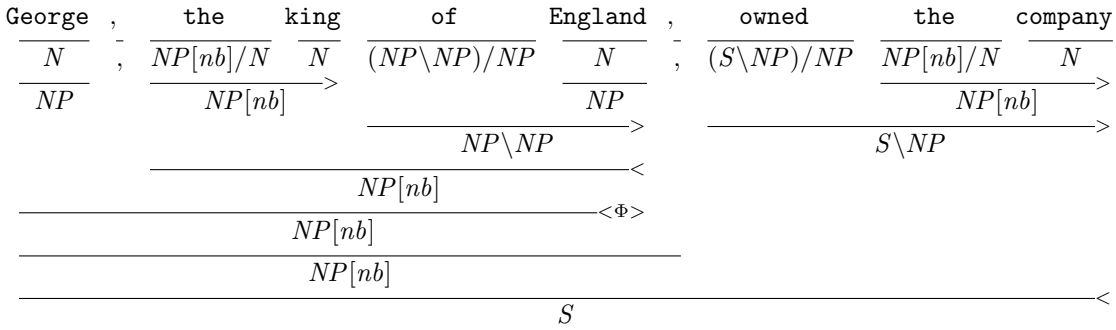

Figure 2: A CCG derivation containing commas used for apposition and absorption

bigram	Constituent			Coverage		Ambiguity
	# No	# Yes	# Uniq	Σ	%	
of the	4936	0	0	9872	1.06	1.031
in the	3911	5	1	17704	1.90	1.747
, the	3489	0	0	24682	2.66	1.005
, and	2219	9	3	29138	3.13	1.000
, a	2167	0	0	33472	3.60	1.000
, which	1705	0	0	36882	3.97	1.118
for the	1638	0	0	40158	4.32	1.958
to the	1588	1	1	43336	4.66	1.925
on the	1533	0	0	46402	4.99	1.962
, said	1258	0	0	48918	5.26	1.290
, but	1193	1	1	51306	5.52	1.045
the company	8	1157	1	53636	5.77	1.000
, he	1165	0	0	55966	6.02	1.000
that the	1150	0	0	58266	6.27	1.283
a share	3	1082	7	60436	6.50	1.107

Table 3: Constituent statistics about the 15 most frequent bigrams in CCGbank 02 to 21. The columns show the number of times the bigram was seen forming a non-constituent, forming a constituent, and then the number of unique constituent-forming chart structures. The next two columns show accumulatively what percentage of sections 02 to 21 these bigrams alone cover. The last column shows the ambiguity the C&C supertagger associates to each n-gram

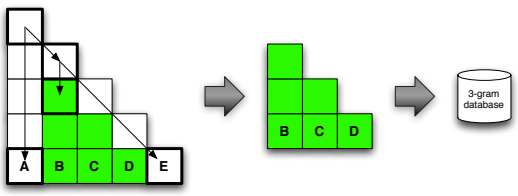

Figure 3: When creating the trigram database, if a trigram forms a constituent in the chart, it is added to the database

maximum database size is important because more data is better for the database construction phase.

6.3 Constructing the n-gram Databases

The construction of the final database is a multi-stage process, with intermediate databases being generated and then refined. The first stage in this process is to parse all of the training data, which in our case is WSJ sections 02 to 21. The parse tree for every sentence is then analysed for constituent-forming n-grams. If a constituent-forming n-gram is found and its size (number of tokens) is one for which we would like to construct a database for, then the n-gram and its corresponding chart structure are written out to a database. These first stage databases are implemented using a simple key-value Tokyo Cabinet hash database. The structure of the keys and values in this database are

Key = `(n-gram, hash of chart)`
Value = `(chart, occurrence counter)`

The `chart` attribute in the value is a serialised version of the chart which can be unserialised at some later point for reuse. The `occurrence` counter is incremented each time an occurrence of a key is seen in the parsed training data. A record is also kept in the database for the number of times a particular n-gram was seen forming a non-constituent, for use in the filtering stage discussed in later Section 6.4.

This process of n-gram chart serialisation is illustrated in Figure 3. When parsing the sentence `A B C D E`, the trigram `B C D` formed a constituent in the spanning analysis for the sentence. Because it formed a constituent, the trigram is added to the first stage trigram database.

6.4 Frequency Reduction

When constructing the initial set of databases over a body of text, a large number of the n-grams which were memoised should not be kept in the final databases because they occur too infrequently, or because the number of times they are seen forming a non-constituent outweighs the number of times they are seen forming a constituent. As such, a frequency based filtering stage is performed on the initial set of databases to produce the final database.

$$\frac{C_0}{\sum_{k \neq 0} C_k} < X \tag{1}$$

$$m = \arg\max_k C_k \tag{2}$$

$$\frac{(\sum_k C_k) - m}{m} < X \wedge m > Y \tag{3}$$

An n-gram was chosen to be filtered out differently depending on whether or not it was seen forming a non-constituent during the database development phase. Equations 1 and 3 describe the predicates which need to be fulfilled in order for a particular n-gram not to be filtered out. In these inequalities, C is a mapping from chart structure to frequency count for the current n-gram, the 0th index into C is the non-constituent-forming frequency count, and X and Y are parameters to the filtering process.

If an n-gram was seen forming a non-constituent during the initial database development phase, then Equation 1 is used. If an n-gram was never seen forming a non-constituent during the development phase, then Equation 3 is used.

The values given to the X and Y parameters in the filtering process were determined through a trial and error process, training on sections 02 to 21 and testing on section 00 of the noun phrase corrected CCGbank. For all of our results, X was set to 0.05 and Y was set to 15.

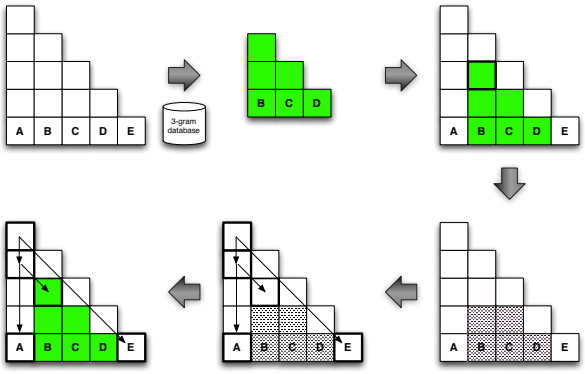

Figure 4: Illustration of using the n-gram databases. The trigram `B C D` is loaded from the pre-constructed database, and blocks out the corresponding cells

6.5 Using the n-gram Databases

Once the n-gram database has been constructed, it is used when parsing sentences in the future. For every sentence that is parsed, the parser checks to see if any n-gram contained within the current sentence exists within the database, and if so, uses the memoised analysis for the n-gram. This process is illustrated in Figure 4.

This n-gram check is performed by iterating top to bottom, left to right through the chart for the current sentence. A consequence of this is that if two n-grams overlap and both exist in the database, then only the first n-gram encountered will have its analyses loaded in from the database. Once the analyses are loaded into the current chart for the n-gram, the corresponding cells in the current chart are blocked off from further use in the parse tree creation process (CKY), as illustrated in Figure 4. It is due to this cell blocking that the pre-constructed charts for the 2nd overlapping n-gram are not also loaded.

Model		Baseline	2-gram	3-gram
WSJ derivs	Time	82.8	82.8	82.6
	LF	85.26	85.26	85.26
	UF	92.01	92.01	92.01
	Cov	98.64	98.64	98.64
WSJ hybrid	Time	84.1	83.8	83.9
	LF	87.40	87.40	87.40
	UF	93.10	93.10	93.10
	Cov	98.64	98.64	98.64
NP derivs	Time	84.1	84.0	84.3
	LF	83.60	83.60	83.60
	UF	90.48	90.48	90.48
	Cov	98.54	98.54	98.54
NP hybrid	Time	84.8	84.9	85.3
	LF	85.88	85.88	85.88
	UF	91.63	91.63	91.63
	Cov	98.54	98.54	98.54

Table 4: The speed versus performance trade-off for varying sized n-grams evaluated on CCGbank 00 using different parsing models. The evaluation attributes are parse time (s), labelled and unlabelled F-score (%), and percentage of sentences covered

7 Results

A set of experiments were conducted using CCGbank sections 02 to 21 as the corpus for developing our database. This corpus was parsed using a variety of statistical parsing models. Section 00 was then used for evaluation. Table 4 shows our preliminary results. The first two parsing models used were trained on the original CCGbank (WSJ derivs and hybrid), and the second two models were trained on the noun phrase corrected CCGbank corpus described in Vadas and Curran (2008) (NP derivs and hybrid). The databases used to obtain these results contained only constituent-forming n-grams.

These results show a non-significant change in speed nor F-score. One positive aspect of this non-significant change is that performance did not decrease even though additional computation is needed to perform our database lookups and chart insertion. The C&C parser is already very fast, and the amount of time taken to perform the chart loading and insertion from the databases happens to be very similar to the time taken to construct the derivations from scratch.

Another experiment was then performed in order to assess the potential of using non-constituent-forming n-grams for memoisation. The bigrams `of the` and `in the` are the two most frequently occurring non-constituent-forming bigrams in CCGbank sections 02 to 21. In order to assess the viability of using non-constituents in our database, our experiments here used only the most frequently occurring analyses for these two bigrams. If no improvement in performance is observed using the most frequently occurring bigrams, then the idea is not worth pursuing further.

	Baseline	`of the`	`in the`	Combined
Time	67.2	67.0	65.8	66.0
LF	87.58	87.30	87.30	87.24
UF	93.14	92.86	92.89	82.83
Cov	94.30	94.30	84.41	84.30

Table 5: Memoised structures were constructed for the most frequent derivations for varying non-constituent-forming bigrams, which were then used and evaluated against section 00 of the noun-phrase corrected CCGbank

The results of these experiments can be seen in Table 5. As was the case in our constituent-forming experiment, no significant change in performance was achieved; positive or negative.

8 Conclusion

Through the analysis of this one structure per n-gram idea using CCG, combined with a preliminary set of empirical results, we have shown that memoising parse structures based on frequently occurring n-grams does not result in any form of performance improvement.

9 Acknowledgments

We would like to thank the anonymous reviewers for their useful feedback. This work was partially supported by the Capital Markets Cooperative Research Centre Limited. Aspects of this work were carried out as part of the Summer Research Workshop on Machine Learning for Language Engineering at the Center for Language and Speech Processing, Johns Hopkins University.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Compu-*

tational Linguistics, 25(2):237–265.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics, Main Volume*, pages 103–110, Barcelona, Spain, July.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Stephen Clark and Julia Hockenmaier. 2002. Evaluating a wide-coverage CCG parser. In *Proceedings of the LREC Workshop on Beyond Parseval*, pages 60–66, Las Palmas, Spain.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorical grammar. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 79–86, Morristown, NJ, USA. Association for Computational Linguistics.

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

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

Ronald M. Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell III, and Er Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT-NAACL04*, pages 97–104, Boston, Massachusetts, USA, May. Association for Computational Linguistics.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

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

David Vadas and James R. Curran. 2008. Parsing noun phrase structure with CCG. In *Proceedings of the 46th Meeting of the Association for Computational Linguists: HLT*, pages 335–343, Columbus, Ohio, June. Association for Computational Linguistics.

David Vadas. 2009. *Statistical Parsing of Noun Phrase Structure*. Ph.D. thesis, University of Sydney.

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