Efficient Combinatory Categorial Grammar Parsing

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

Abstract

Efficient wide-coverage parsing is integral to large-scale NLP applications. Unfortunately, parsers for linguistically motivated formalisms, e.g. HPSG and TAG, are often too inefficient for these applications.

This paper describes two modifications to the standard CKY chart parsing algorithm used in the Clark and Curran (2006) Combinatory Categorial Grammar (CCG) parser. The first modification extends the tight integration of the supertagger and parser, so that *individual* supertags can be added to the chart, which is then repaired rather than rebuilt. The second modification adds constraints to the chart that restrict which constituents can combine.

Parsing speed is improved by 30–35% without a significant accuracy penalty and a small increase in coverage when both of these modifications are used.

1 Introduction

Parsing is the process of determining the syntactic structure of a sentence. It is an integral part of the deep semantic analysis that any sophisticated Natural Language Processing (NLP) system, such as Question Answering and Information Extraction systems, must perform.

The sentences Bob killed Alice, Alice was killed by Bob and Bob was the man who killed Alice convey the same information. If we treat the sentence as a *bag* or *sequence of words* by assuming limited structure, the sentences appear to be very different. These examples demonstrate that full parsing is necessary for accurate semantic interpretation. Further, sophisticated linguistic analysis capable of modelling a wider range of phenomena should give us the most information.

Unfortunately, parsing is very inefficient because of the large degree of ambiguity present in natural language. This is particularly true for wide-coverage grammars in linguistically expressive formalisms, especially those automatically extracted from a treebank.

Many NLP systems use shallow parsing because full parsing is too slow (Grishman, 1997). To improve the approximate structure identified by shallow parsers, many systems use domain-specific knowledge to extract dependencies (Grishman, 1997; Cole et al., 1997). Ciravegna et al. (1997) show that the accuracy can be improved by using a better parser. The ability of NLP systems to extract useful and correct information could therefore be improved substantially if the speed of full parsing was acceptable.

The C&C CCG parser (Clark and Curran, 2006) is the fastest linguistically motivated parser in the literature, but it is still limited to about 25 sentences per second on commodity hardware.

This paper describes two modifications to the C&C parser that significantly improve parsing efficiency without reducing accuracy or coverage. The first involves *chart repair*, where the CKY chart is repaired when new categories are added, instead of rebuilt from scratch. This allows an even tighter integration of the parser and supertagger (described below) which results in an 11% speed improvement over the original parser.

The second modification involves parsing with *constraints*, that is, requiring certain spans to be constituents. This reduces the search space considerably by eliminating a large number of constituents that cross the boundary of these spans. The best set of constraints results in a 10% improvement over the original parser. These constraints are also useful for other tasks. Finally, when both chart repair and constraints are used, a 30-35% speed improvement is achieved while coverage increases and the accuracy is unchanged.

Proceedings of the 2006 Australasian Language Technology Workshop (ALTW2006), pages 3–10.

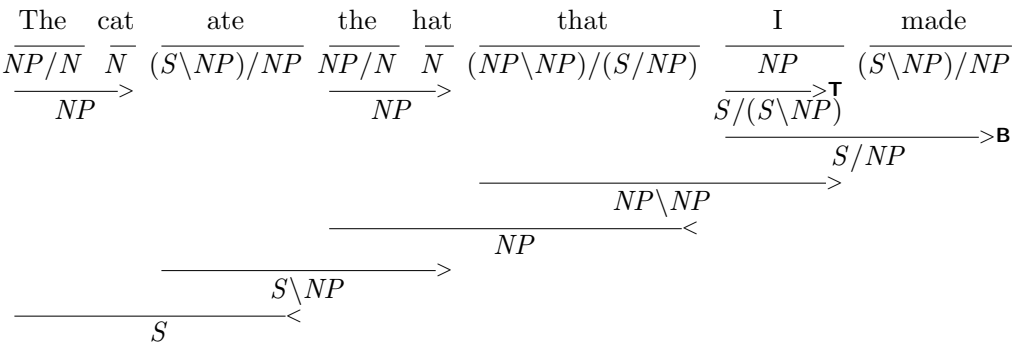

Figure 1: A Combinatory Categorial Grammar derivation

2 Combinatory Categorial Grammar

Context free grammars (CFGs) have traditionally been used for parsing natural language. However, some constructs in natural language require more expressive grammars. *Mildly context sensitive* grammars, e.g. HPSG and TAG, are powerful enough to describe natural language but like CFGs (Younger, 1967) are polynomial time parseable (Vijay-Shanker and Weir, 1990).

Combinatory Categorial Grammar (CCG) is another mildy context sensitive grammar (Steedman, 2000) that has significant advantages over CFGs, especially for analysing constructions involving coordination and long range dependencies. Consider the following sentence:

Give a teacher an apple and a policeman a flower.

There are local dependencies between give and teacher, and between give and apple. The additional dependencies between give, and policeman and flower are long range, but are extracted easily using CCG. a policeman a flower is a non-standard constituent that CCG deals with very elegantly, allowing a teacher an apple and a policeman a flower to be coordinated before attachment to the verb give.

In CCG each word is assigned a category which encodes sub-categorisation information. Categories are either *atomic*, such as N, NP and S for noun, noun phrase or sentence; or *complex*, such as NP/N for a word that combines with a noun on the right to form a noun phrase. $NP\backslash N$ is similarly a word that combines with a noun on the left to create an NP. The categories are then combined using combinatory rules such as *forward application* $>$ ($X/Y\ Y \Rightarrow X$) and *forward composition* $>\mathbf{B}$ ($X/Y\ Y/Z \Rightarrow_{\mathbf{B}} X/Z$).

An example derivation that uses a number of combination rules is shown in Figure 1. The example demonstrates how CCG handles long range dependencies such as the hat ... I made. Type raising ($>\mathbf{T}$) and forward composition ($>\mathbf{B}$) on I made are used to derive the same predicate-argument structure as if it was written as I made the hat.

A derivation creates predicate-argument dependencies, which are 5-tuples $\langle h_f, f, s, h_a, l\rangle$, where h_f is the word of the category representing the relationship, f is the category itself, s is the argument slot, h_a is the head word of the argument and l indicates if the dependency is local.

The argument slot is used to identify the different arguments of words like bet in $[\text{Alice}]_1$ bet $[\text{Bob}]_2$ $[\text{five dollars}]_3$ $[\text{that they win}]_4$. The dependency between bet and Bob would be represented as $\langle \text{bet}, (((S\backslash NP_1)/NP_2)/NP_3)/S[em]_4, 2, \text{Bob}, -\rangle$. The C&C parser is evaluated by comparing extracted dependencies against the gold standard. Derivations are not compared directly because different derivations can produce the same dependency structure.

The parser is trained on CCGbank, a version of Penn Treebank translated semi-automatically into CCG derivations and predicate-argument dependencies (Hockenmaier and Steedman, 2006). The resulting corpus contains 99.4% of the sentences in the Penn Treebank. Hockenmaier and Steedman also describe how a large CCG grammar can be extracted from CCGbank. A grammar automatically extracted from CCGbank is used in the C&C parser and supertagger.

3 C&C CCG Parser

The C&C parser takes one or more syntactic structures (categories) assigned to each word and attempts to build a spanning analysis of the sentence. Typically every category that a word was seen with in the training data is assigned.

Supertagging (Bangalore and Joshi, 1999) was introduced for Lexicalized Tree Adjoining Grammar (LTAG) as a way of assigning fewer categories to each word thus reducing the search space of the parser and improving parser efficiency.

Clark (2002) introduced supertagging for CCG parsing. The supertagger used in the C&C parser is a maximum entropy (Berger et al., 1996) sequence tagger that uses words and part of speech (POS) tags in a five word window as features. The label set consists of approximately 500 categories (or supertags) so the task is significantly harder than other NLP sequence tagging tasks.

The supertagger assigns one or more possible categories to each word together with the probability for each of the guesses. Clark and Curran (2004) discovered that initially assigning a very small number of categories per word and then attempting to parse was not only faster but more accurate than assigning many categories. If no spanning analysis could be found the parser requested more categories from the supertagger, and the parsing process was repeated until the number of attempts exceeded a limit (typically 5 levels of supertagger ambiguity). This tight integration of the supertagger and the parser resulted in state of the art accuracy and a massive improvement in efficiency, reaching up to 25 sentences a second.

Up until now, when a spanning analysis was not found the chart was destroyed, then extra categories are assigned to each word, and the chart is built again from scratch. However the chart rebuilding process is very wasteful because the new chart is always a superset of the previous one and could be created by just updating the old chart instead of rebuilding it.

This has limited how small the initial ambiguity levels can be set and thus how closely the parser and supertagger can interact. The first modification we describe below is to implement *chart repair* which allows additional categories to be assigned to an existing chart and the CKY algorithm to run efficiently over just the modified section.

4 Chart Parsing

Given a sentence of n words, we define position $pos \in \{0, \ldots, n-1\}$ to be the starting position of a span (contiguous sequence of words), and $span$, its size. So the hat in the cat ate the hat would have $pos = 3$ and $span = 2$. Each span can be parsed in a number of ways so a set of derivations will be created for each valid $(pos, span)$ pair. Let $(pos, span)$ represent this set of derivations. Then, the derivations for $(pos, span)$ will be combinations of derivations in (pos, k) and $(pos+k, span-k)$ for all $k \in \{1, \ldots, span-1\}$. The naïve way to parse a sentence using these definitions is to find the derivations that span the whole sentence $(0, n)$ by recursively finding derivations in $(0, k)$ and $(k, n-k)$ for all $k \in \{1, \ldots, n-1\}$. However, this evaluates derivations for each $(pos, span)$ pair multiple times, making the time complexity exponential in n.

To make the algorithm polynomial time, dynamic programming can be used by storing the derivations for each $(pos, span)$ when they are evaluated, and then reusing the stored values. The *chart* data structure is used to store the derivations. The chart is a two dimensional array indexed by pos and $span$. The valid pairs correspond to $pos + span \leq n$, that is, to spans that do not extend beyond the end of the sentence. The squares represent valid cells in Figure 2. The location of $cell(3, 4)$ is marked with a diamond. $cell(3, 4)$ stores the derivations whose yield is the four word sequence indicated.

The CKY (also called CYK or Cocke-Younger-Kasami) algorithm used in the C&C parser has an $O(n^3)$ worst case time complexity. Sikkel and Nijholt (1997) give a formal description of CKY (Younger, 1967) and similar parsing algorithms, such as the Earley parser (Earley, 1970).

The CKY algorithm is a bottom up algorithm and works by combining adjacent words to give a span of size two (second row from the bottom in Figure 2). It then combines adjacent spans in the first two rows to create all allowable spans of size three in the row above. This process is continued until a phrase that spans the whole sentence (top row) is reached.

The (lexical) categories in the bottom row (on the lexical items themselves) are assigned by the supertagger (Clark and Curran, 2004). The number of categories assigned to each word can be varied dynamically. Assigning a small number of categories (i.e. keeping the level of lexical category ambiguity low) increases the parsing speed significantly but does not always produce a spanning derivation. The original C&C parser uses a small number of categories first, and if no spanning tree is found the process is repeated with a larger number of categories.

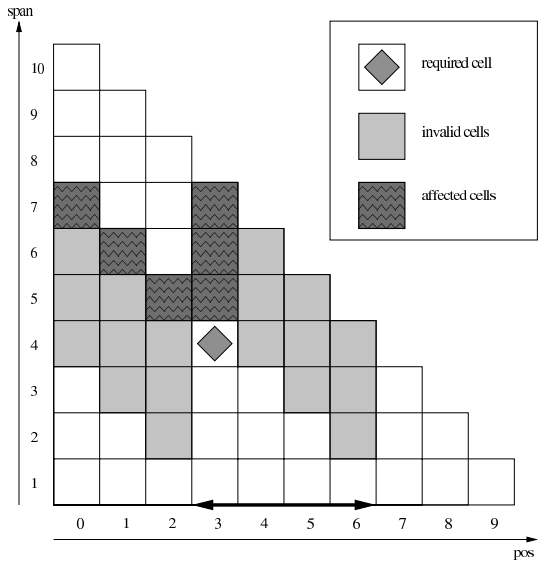

Figure 2: Cells affected by adding a constraint. The axes are the cell indices in the chart with **pos** the starting position, and **span** the length of the span, of constituents in the cell.

5 Constraints

The original C&C parser uses a supertagger to assign a number of categories to each word, together with the probability of each category (Clark and Curran, 2004). In the first iteration, only categories with $\beta \geq 0.075$ are used, where β is the ratio of the probability of the category and the probability of the most likely category for that word. For example, if the categories for dog are N, NP and N/N with probabilities 0.8, 0.12 and 0.08 then β value of N/N is $0.08/0.8 = 0.1$. If there is no spanning tree, a lower value is used for the cutoff β in the next iteration so that more tags are assigned to each word. The previous chart is destroyed and the new chart is built from scratch. Since the assigned categories are always a superset of the previously assigned ones, the derivations in the new chart will include all the derivations in the previous chart.

Instead of rebuilding the chart when new categories are added it can be simply repaired by modifying cells that are affected by the new tags. Considering the case where a single tag is added to the ith word in an n word sentence, the new tag can only affect the cells that satisfy $pos \leq i$ and $pos + span > i$. These cells are shown in Figure 3. The chart can therefore be repaired bottom up by updating a third of the cells on average.

The number of affected cells is $(n - pos) \times pos$ and the total number of cells is approximately $\frac{n^2}{2}$. The average number of affected cells is approximately $\frac{1}{n}\int_0^n (n-p)p \; dp = \frac{n^2}{6}$, so on average a third of the cells are affected.

The chart is repaired bottom up. A new category is added to one word by adding it to the list of categories in the appropriate cell in the bottom row. The list is marked so that we know which categories are new. For each cell C in the second row we look for each pair of cells A and B whose spans combine to create the span of C. In the original algorithm all categories from A are combined with all categories from B, but during the repair this is only done if at least one of them is new because otherwise the resulting category would already be in C. Again the list of categories in C is marked so that cells higher in the chart know which categories are new. This is repeated for all affected cells.

This speeds up the parser not only because previous computations are reused, but also because categories can be added one at a time until a spanning derivation is found. This increases coverage slightly because the number of categories can be varied one by one. In the original parser it was possible to have sentences that a spanning tree cannot be found for using for example 20 categories, but increasing the number of categories to 25 causes the total number of derivations in the chart to exceed a predefined limit, so the sentence does not get parsed even if 23 categories would produce a spanning tree without exceeding the limit.

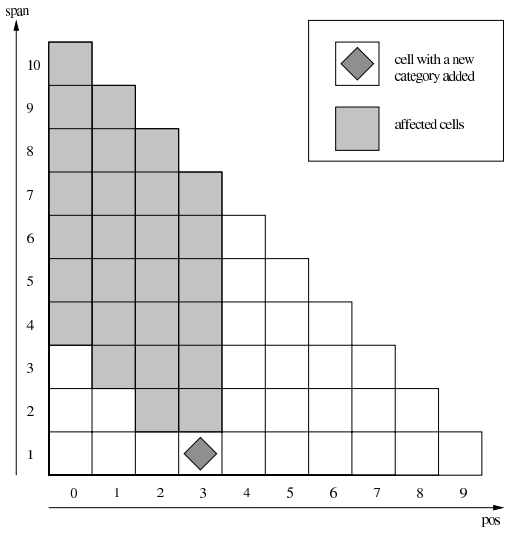

Figure 3: Cells affected by chart repair.

6 Chart Repair

Adding constraints to the parser was originally designed to enable more efficient manual annotation. The parser has been used for semi-automatic parsing of sentences to create gold standard derivations. It is used to produce a guess, which is then manually corrected. Adding a constraint could speed this process up by allowing annotators to quickly tell the parser which part it got wrong.

For example if the Royal Air Force contract was parsed as (the Royal (Air Force contract)), meaning that the Air Force contract was royal, instead of manually annotating the category for each word the annotator could simply create a constraint on Royal Air Force requiring it to be a constituent, and thus making the previous derivation impossible. The correct derivation, (the ((Royal Air Force) contract)) would then very likely be produced without further effort from the annotator.

However, parsing would be much faster in general if the search space could be constrained by requiring *known* spans P to be a single constituent. This reduces the search space because P must be the yield of a single cell $C_P(pos_P, span_P)$, so the cells with yields that cross the boundary of P do not need to be considered at all (grey squares in Figure 2). The problem of what spans are known in advance is described below.

In addition, if a cell contains P as a *prefix* or *suffix* (wavy pattern cells in Figure 2) then it also has constraints on how it can be created. In Figure 2, $P = cell(3, 4)$ is required, i.e. the span starting at word 3 of length 4 containing words 3, 4, 5 and 6 is a constituent. Consider $cell(3, 7)$. It includes words 3 to 9 and contains P as the prefix. Normally $cell(3, 7)$ can be created by combining $cell(3, 1)$ with $cell(4, 6)$, ..., $cell(pos, s)$ with $cell(pos + s, span - s)$, ..., and $cell(3, 6)$ with $cell(9, 1)$. However the first three of these combinations are not allowed because the second component would cross the boundary of P. This gives a lower limit for the span of the left component. Similarly if P is the suffix of the span of a cell then there is a lower limit on the span of the right component.

This eliminates a lot of work during parsing and can provide a significant speed increase. In the quick brown foxes like dogs, the following phrases would all be possible constituents: foxes like dogs, brown foxes like dogs, ..., and the quick brown foxes like dogs. Since the chart is built bottom up the parser has no knowledge of the surrounding words so each of those appears like a valid constituent and would need to be created. However if the quick brown foxes is known to be a constituent then only the last option becomes possible.

7 Creating Constraints

How can we know that specific spans must be yielded by constituents in advance? Surely the parsing is already solved if we have this information? In this paper, we have experimented with constraints determined from shallow parsing and hints in the sentence itself.

Chunk tags (gold standard and from the C&C chunker) were used to create constraints. Only NPs

were used because the accuracy for other chunks is low. The chunks required modification because the Penn Treebank has different analyses to CCGbank, e.g. Larry's dog is chunked as $[\text{Larry}]_{\text{NP}}$ $[\text{'s dog}]_{\text{NP}}$, which is different to the CCGbank analysis. Adjacent NPs of this form are concatenated.

A number of *punctuation constraints* were used and had a significant impact especially for longer sentences. The punctuation rules in CCGbank are very productive. For example the final punctuation in The dog ate a bone. will create all of these constituents: bone., a bone., ..., The dog ate a bone. However, in CCGbank the sentence final punctuation is always attached at the root. A constraint on the the first $n-1$ words was added to force the parser to only attach the sentence final punctuation once the rest of the sentence has been parsed.

Constraints are also placed on parenthesised expressions. In The dog (a hungry Pomeranian) ate a bone, the phrase a hungry Pomeranian clearly needs to be parsed first and then attached to the rest of the sentence. However, CCGbank would allow the right parenthesis to be absorbed by Pomeranian before hungry is attached. The same would apply to quoted expressions but CCGbank has removed the quotation marks. However, the parser must still deal with quoted expressions in practical systems.

Finally constraints are placed on phrases bounded by semicolons, colons and hyphens. This is especially useful with longer sentences of the form Alice paid $5; Bob paid $6; ..., where many clauses are separated by semicolons. This reduces the sentence to a number of smaller units which significantly improves parsing efficiency.

If the parser cannot parse the sentence with constraints then they are removed and the sentence is parsed again. This increases the coverage because the reduced search space means that longer sentences can be parsed without exceeding the memory or time limit. However if the constraint accuracy is low a large number of sentences will need to be parsed twice which would cancel out anything gained from using them.

8 Experiments

The parser is trained on CCGbank sections 02-21, with section 00 being used for development. The performance is measured in terms of coverage, accuracy and parsing time. The time reported includes loading the grammar and statistical model, which is $\sim$5 seconds, and parsing the 1913 sentences in section 00. Accuracy is measured in terms of the dependency F-score.

The failure rate (the opposite of coverage) is broken down into sentences with length up to 40 and with length over 40 because the longer sentences are the most problematic ones and the original parser already has high coverage on sentences with up to 40 words. There are 1784 1-40 word sentences and 129 41+ word sentences. The average length and standard deviation in 41+ are 50.8 and 31.5 respectively.

All experiments used gold standard POS tags. Some experiments use gold standard chunks to determine an upper bound on the utility of chunk constraints. *Original* and *Original+repair* do not use any constraints. *NP(gold)* indicates that gold standard noun phrase constraints are used. *NP* uses the C&C chunker, and *punctuation* adds punctuation constraints. The times reported for *NP* (using the C&C chunker) include the time to load the chunker model ($\sim$1.3 seconds).

Finally the best performing system was compared against the original on section 23, which has 2257 sentences of length 1-40 and 153 of length 41+. The maximum sentence length is only 65, which explains the high coverage for the 41+ section.

9 Results

The results in Table 1 show that using gold standard noun phrases does not improve efficiency, while using noun phrases identified by the chunker decreases speed by 10.8%. This is not surprising because the chunk data was not obtained from CCGbank and the chunker is not very accurate. Some frequent problems were fixed in a preprocessing step as explained in Section 5, but there could be less frequent constructions that cause problems. A more detailed analysis of these constructions is required.

Chart repair (without constraints) gave an 11.1% improvement in speed and 0.21% improvement in accuracy. The accuracy was improved because of the way the repair process adds new categories. Categories are added in decreasing order of probability and parsing stops once a spanning tree is found. This effectively allows the parser to use the probabilities which the supertagger assigns, which are not directly modelled in the parser. Once supertagger probabilities are added to the parser statistical model there should be no

	TIME		ACC	COVER	FAIL RATE %	
	secs	%	%	%	$n \leq 40$	$n > 40$
Original	88.3	—	86.54	98.85	0.392	11.63
punctuation	79.1	10.4	86.56	99.22	**0.168**	9.30
NP(gold)	88.4	-0.1	86.27	99.06	0.224	10.85
NP	97.8	-10.8	86.31	99.16	0.224	9.30
NP(gold) + punctuation	69.8	20.5	86.24	99.27	**0.168**	8.53
NP + punctuation	97.0	-9.9	86.31	99.16	**0.168**	10.08
Original + repair	78.5	11.1	**86.75**	99.01	0.336	10.08
NP(gold) + repair	65.0	26.4	86.04	99.37	0.224	6.20
NP + repair	77.5	12.2	86.35	99.37	0.224	6.20
punctuation + repair	57.2	35.2	86.61	99.48	**0.168**	5.43
NP(gold) + punctuation + repair	**48.2**	**45.4**	86.14	99.48	**0.168**	5.43
NP + punctuation + repair	63.2	28.4	86.43	**99.53**	0.163	**3.88**

Table 1: Parsing performance on section 00 with constraints and chart repair.

accuracy difference between the original method and chart repair.

The best results for parsing with constraints (without repair) were with gold standard noun phrase and punctuation constraints, with 20.5% improvement in speed and 0.42% in coverage. In that case, however the accuracy decreases by 0.3% which is again possibly because CCG constituents do not match up with the chunks every time. The best results obtained without a decrease in accuracy is using only punctuation constraints, with 10.4% increase in speed and 0.37% in coverage.

The best overall result was obtained when gold standard noun phrase and punctuation constraints were used with chart repair, with a 45.4% improvement in speed and 0.63% in coverage, and a 0.4% drop in accuracy. Again the best results without a drop in accuracy were with only punctuation constraints and chart repair, with improvements of 35.2% and 0.63%.

The results also show that coverage of both short and long sentences is improved using these methods. For example the best results show a 43% and 67% decrease in failure rate for sentence lengths in the ranges 1-40 and 41+.

Comparing the last three rows allows us to guess how accurate the chunker will need to be to achieve a faster speed than just using punctuation constraints. Noun phrases clearly have an impact on speed because using gold standard chunks gives a significant improvement, however the C&C chunker is currently not accurate enough. The chunker would need to have about half the error rate it currently has in order to be useful.

Table 2 shows the performance of the punctuation constraints and chart repair system on section 23. The results are consistent with previous results, showing a 30.9% improvement in speed and 0.29% in coverage, with accuracy staying at roughly the same level.

10 Future Work

A detailed analysis of where NPs chunks do not match the CCG constituents is required if NPs are to be used as constraints. The results show that NPs can provide a large improvement in efficiency if identified with sufficient precision.

The chart repair has allowed an even greater level of integration of the supertagger and parser. We intend to explore strategies for determining which category to add next if a parse fails.

Constraints and chart repair both manipulate the chart for more efficient parsing. Other methods of chart manipulation for pruning the search space will be investigated. Agenda based parsing, in particular A* parsing (Klein and Manning, 2003), will be implemented in the C&C parser, which will allow only the most probable parts of the chart to be built, improving efficiency while guaranteeing the optimal derivation is found.

11 Conclusion

We have introduced two modifications to CKY parsing for CCG that significantly increase parsing efficiency without an accuracy or coverage penalty.

Chart repair improves efficiency by reusing the partial CKY chart from the previous parse at-

	TIME		ACC	COVER	FAIL RATE %	
	secs	%	%	%	$n \leq 40$	$n > 40$
Original	91.9	—	**86.92**	99.29	0.621	1.961
punctuation + repair	**63.5**	**30.9**	86.89	**99.58**	**0.399**	**0.654**

Table 2: Parsing performance on Section 23 with constraints and chart repair.

tempts. This allows us to further exploit the tight integration of the supertagger and parser by adding one lexical category at a time until a parse of the sentence is found. Chart repair alone gives an 11% improvement in speed.

Constraints improve efficiency by avoiding the construction of sub-derivations that will not be used. They have a significant impact on parsing speed and coverage without reducing the accuracy, provided the constraints are identified with sufficient precision.

When both methods are used the speed increases by 30-35%, the failure rate decreases by 40-65%, both for sentences of length 1-40 and 41+, while the accuracy is not decreased. The result is an even faster state-of-the-art wide-coverage CCG parser.

12 Acknowledgments

We would like to thank the anonymous reviewers for their feedback. This research was funded under Australian Research Council Discovery grants DP0453131 and DP0665973.

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

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

Fabio Ciravegna, Alberto Lavelli, and Giorgio Satta. 1997. Efficient full parsing for Information Extraction. In *Proceedings of the Meeting of the Working Groups on Automatic Learning and Natural Language of the Associazione Italiana per l'Intelligenza Artificale (AI*IA)*, Turin, Italy.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *20th International Conference on Computational Linguistics*, pages 282–288, Geneva, Switzerland.

Stephen Clark and James R. Curran. 2006. Wide-coverage statistical parsing with CCG and log-linear models. (submitted).

Stephen Clark. 2002. A supertagger for Combinatory Categorial Grammar. In *Proceedings of the TAG+ Workshop*, pages 19–24, Venice, Italy.

Ronald Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue. 1997. *Survey of the state of the art in human language technology*. Cambridge University Press, New York, NY, USA.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Ralph Grishman. 1997. Information Extraction: Techniques and challenges. In *SCIE '97: International Summer School on Information Extraction*, pages 10–27, London, UK. Springer-Verlag.

Julia Hockenmaier and Mark Steedman. 2006. CCGbank - a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. (submitted).

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of Human Language Technology and the North American Chapter of the Association for Computational Linguistics Conference*, pages 119–126, Edmond, Canada.

Klass Sikkel and Anton Nijholt. 1997. Parsing of Context-Free languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 2: Linear Modelling: Background and Application*, pages 61–100. Springer-Verlag, New York.

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

K. Vijay-Shanker and David J. Weir. 1990. Polynomial time parsing of combinatory categorial grammars. In *Proceedings of the 28th Annual Meeting on Association for Computational Linguistics*, pages 1–8, Pittsburgh, Pennsylvania.

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