Frontier Pruning for Shift-Reduce CCG Parsing

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

Abstract

We apply the graph-structured stack (GSS) to shift-reduce parsing in a Combinatory Categorial Grammar (CCG) parser. This allows the shift-reduce parser to explore all possible parses in polynomial time without resorting to heuristics, such as beam search. The GSS-based shift-reduce parser is 34% slower than CKY in the finely-tuned C&C parser. We perform *frontier pruning* on the GSS, increasing the parsing speed to be competitive with the C&C parser with a small accuracy penalty.

1 Introduction

Parsing is a vital component of sophisticated natural language processing (NLP) systems that require deep and accurate semantic interpretation, including question answering and summarisation. Unfortunately, the complexity of natural languages results in substantial ambiguity. For even a typical sentence, thousands of potential analyses may be considered by a wide-coverage parser, making parsing impractical for large-scale applications.

Several methods have been proposed to improve parsing speed, including supertagging (Bangalore and Joshi, 1999; Clark and Curran, 2004; Kummerfeld et al., 2010), coarse-to-fine parsing (Charniak and Johnson, 2005; Pauls and Klein, 2009), chart repair (Djordjevic, 2006), chart constraints (Roark and Hollingshead, 2009), structure caching (Dawborn and Curran, 2009) and chart pruning (Zhang et al., 2010). These heuristic methods offer a trade-off between accuracy and speed. A* parsing (Klein and Manning, 2003) offers speed increases with no reduction in accuracy. For parsers optimised for speed, the overhead required by additional efficiency techniques can exceed the speed gains they provide (Dawborn and Curran, 2009). As mistakes made in the parsing phase propagate to later stages, high speed but low accuracy parsers may not be useful in NLP systems (Chang et al., 2006).

In this paper, we modify the C&C (Clark and Curran, 2007) Combinatory Categorial Grammar (CCG) parser to enable shift-reduce (SR) parsing. The Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967) is replaced with the shift-reduce algorithm (Aho and Ullman, 1972). However, back-tracking in shift-reduce parsers make them exponential in the worst case.

To eliminate this duplication of work, a graph-structured stack (GSS; Tomita, 1988) is employed. This is the equivalent, for shift-reduce parsing, of the chart in CKY parsing, which stores all possible parse states compactly and enables polynomial time worst-case complexity. Due to the incremental nature of shift-reduce parsing, we can perform pruning of the parse state in the process of considering the next word (the frontier). Our *frontier pruning* model is an averaged perceptron trained to recognise the highest-scoring derivation that the C&C parser would have selected. By eliminating unlikely derivations , we substantially decrease the amount of ambiguity that the parser is required to handle.

The GSS SR parser considers all the derivations that the C&C parser would consider, but is 34% slower. When frontier pruning is applied, incremental parsing speed is improved by 39% relative to the GSS parser with a negligible impact on accuracy.

2 CCG Parsing

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a lexicalised grammar formalism that incorporates both constituent structure and dependency information into its analyses.

In CCG, each word is assigned a category which encodes sub-categorisation information. Categories may be *atomic*, such as N and S; or *complex*, such as NP/N for a word that requires an N to the right to produce an NP. Similarly, $S\backslash NP$ is an intransitive verb and produces a sentence when an NP is found to the left. Finally, a transitive verb receives $(S\backslash NP)/NP$ as it consumes an NP on the right, producing a verb phrase. Figure 1 shows two examples of CCG derivations with lexical categories assigned to each word. Both examples also provide the word saw with the $(S\backslash NP)/NP$ category.

Lexicalised grammars typically have a small set of rules (the *combinatory rules* in CCG) and instead rely on categories that describe a word's syntactic role in a sentence. In Figure 1, the word with contains two separate categories indicating whether it modifies saw (first example) or John (second example). In a highly lexicalised grammar, a parser may need to explore a large search space of categories in order to select the correct category for each word.

Bangalore and Joshi (1999) proposed *supertagging*, where each word is assigned a reduced set of categories by a sequence tagger, rather than all of the categories previously seen with that word. Our supertags are CCG categories, and so are much more detailed than POS tags. By limiting the number of supertags for each word, there is a massive reduction in the number of derivations. The effectiveness of supertagging (Clark and Curran, 2004) demonstrates the influence of lexical ambiguity on parsing complexity for lexicalised grammars.

Hockenmaier and Steedman (2007) developed CCGbank, a semi-automated conversion of the Penn Treebank (Marcus et al., 1993) to the CCG formalism. A number of statistical parsers (Hockenmaier and Steedman, 2002; Clark et al., 2002) have been created for CCG parsing using CCGbank.

2.1 The C&C Parser

Clark and Curran (2007) describe the three stages of the high-performance C&C CCG parser. First, the supertagger provides each word with a set of likely categories, reducing the search space considerably. Second, the parser combines the categories, using the CKY chart-parsing algorithm and CCG's combinatory rules, to produce all derivations that can be constructed with the given categories. Finally, the decoder finds the best derivation from amongst the spanning analyses in the chart.

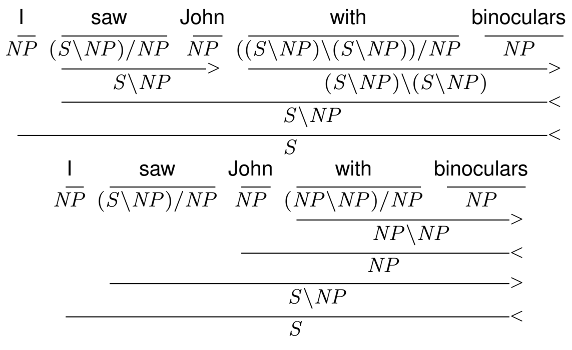
I saw John with binoculars
NP $(S\backslash NP)/NP$ NP $((S\backslash NP)\backslash(S\backslash NP))/NP$ NP
$S\backslash NP$ $(S\backslash NP)\backslash(S\backslash NP)$
$S\backslash NP$
S
I saw John with binoculars
NP $(S\backslash NP)/NP$ NP $(NP\backslash NP)/NP$ NP
$NP\backslash NP$
NP
$S\backslash NP$
S

Figure 1: Two CCG derivations with PP ambiguity.

The C&C parser uses a maximum-entropy model to score each derivation, using a wide range of features defined over local sub-trees in the derivation, including the head words and their POS tags, the local categories, and word-word dependencies. We use the default normal-form mode with the derivations decoder (Clark and Curran, 2007) and a maximum of 1,000,000 categories in the chart.

Clark and Curran (2004) describe the role of supertagging in the C&C parser and its impact on parser speed. The supertagger initially assigns as few supertags as possible per word. If the parser is unable to provide a spanning analysis, the parser requests more supertags for each word. By restricting the number of supertags considered, this provides substantial pruning at the lexical level. Recent work by Kummerfeld et al. (2010) has shown that by training the supertagger on parser output, the parser's speed can be substantially increased whilst achieving the same accuracy as the baseline system. This exploits the idea that the only supertags the parser needs are those used by the highest-scoring derivation, reducing the search space even more than traditional supertagging.

Whilst the approach we present here focuses on CCG parsing, the techniques apply equally to any other binary branching or binarised grammars.

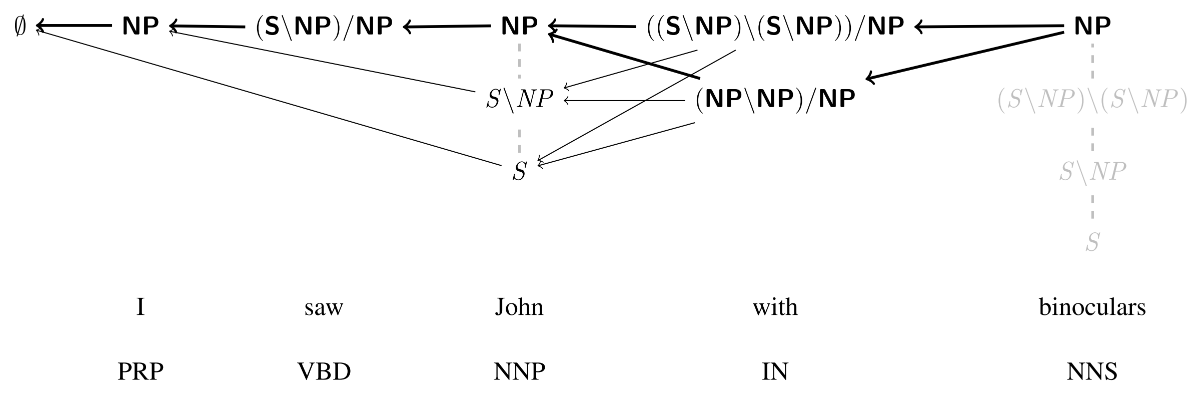

Figure 2: A graph-structured stack (GSS) representing an incomplete parse of the sentences found in Figure 1. The nodes and lines in bold were provided by the supertagger, whilst the non-bold nodes and lines have been created during parsing. The light gray lines represent what reduce operation created that lexical category.

3 Shift-Reduce Parsing

In its deterministic form, a shift-reduce parser performs a single left-to-right scan of the input sentence, selecting one or more actions at each step. The current state of the parser is stored in a stack, where the partial derivation is stored and the parsing operations are performed. For the actions, either we *shift* the current word onto the stack or *reduce* the top two (or more) items at the top of the stack (Aho and Ullman, 1972). As the scoring model can be defined over actions, this can allow for highly efficient parsing through greedy search (Sagae and Lavie, 2005). This has made shift-reduce parsing popular for high-speed dependency parsers (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004).

Unfortunately, a deterministic shift-reduce parser cannot handle ambiguity because it only considers a single derivation. A simple extension is to eliminate determinism and perform a best-first search, backtracking if the parser reaches a dead end. This backtracking leads to duplicate construction of substructures and complete exploration is exponential in the worst case. Beam search has been used to handle this exponential explosion by discarding a large portion of the search space.

In Zhang and Clark (2011), a direct comparison is made between their shift-reduce CCG parser and the chart-based C&C parser. As CCG allows for a limited number of unary rules, specifically type-changing and type-raising, Zhang and Clark extend the shift-reduce algorithm to consider unary actions. In order to handle the exponential search space, their parser performs beam search, only keeping the top 16 scoring states. Whilst this approximate search may potentially lose the best scoring parse, they achieve competitive accuracies compared to the C&C chart parser.

3.1 Advantages of Semi-Incremental Parsing

Shift-reduce parsing allows for fully incremental parsing that does not require the full sentence. Whilst the C&C parser could be modified to perform in this fashion, POS tagging and supertagging accuracy would likely decrease, leading to lower overall parsing accuracy as mistakes propagate up the parsing pipeline.

Semi-incremental parsing can still be advantageous compared to non-incremental parsing. By using features to provide a partial understanding of the sentence structure to components not traditionally integrated with the parser, such as the POS tagger and supertagger, improved accuracy is possible. This is because these components currently only use the orthographic properties of the input text as features, with no understanding of how each word may be potentially used during parsing. In Merity (2011), we have begun exploring tightly integrating parsing and tagging, specifically for POS tags and supertags, by using semi-incremental parsing and shown improved tagging accuracy is possible.

3.2 Graph-Structured Stack

Back-tracking shift-reduce parsers are worst case exponential, preventing a full exploration of the search space. A graph-structured stack (GSS) is a general structure that allows for the efficient handling of non-determinism in shift-reduce parsing (Tomita, 1988). The GSS allows for polynomial time non-deterministic shift-reduce parsing and has been shown to be highly effective for dependency parsing (Huang and Sagae, 2010). The use of GSS allows for the incremental construction of the parse tree without being forced to discard large segments of the search space.

Here we will show an example of using a GSS to augment shift-reduce parsing and then show how it can be applied to CCG parsing. In the example grammar below, all three reduction rules are possible on the given stack. By performing backtracking and pursuing all possible reductions, shift-reduce parsing becomes worst-case exponential as previous results must be re-computed.

$$\begin{tikzcd}
\emptyset & A \arrow[l] & B \arrow[l] & C \arrow[l] & D \arrow[l] & E \arrow[l]
\end{tikzcd}$$

Reduction Rules				
F	$\leftarrow$	D	E	
G	$\leftarrow$	D	E	
H	$\leftarrow$	C	D	E

The GSS solves this by storing multiple possible derivations in a single structure. Note that all possible rules have been applied and are now stored in the GSS. These reduce operations are also non-destructive, leaving the original structure from the above figure in place. Thus, the GSS can store multiple possible derivations. Note that there is only a single bottom node, $\emptyset$, representing an empty stack.

$$\begin{tikzcd}
 & & & & & \mathbf{F} \arrow[ddll] \\
 & & & & & \mathbf{G} \arrow[dll] \\
\emptyset & A \arrow[l] & B \arrow[l] & C \arrow[l] & D \arrow[l] & E \arrow[l] \\
 & & & & & \mathbf{H} \arrow[ulll]
\end{tikzcd}$$

When a new node is pushed onto the stack, we combine it with the heads of all of the existing stacks stored in the GSS. This means that only a single shift action is necessary for the GSS instead of one for each possible derivation.

$$\begin{tikzcd}
 & & & & & F \arrow[ddll] & \\
 & & & & & G \arrow[dll] & \\
\emptyset & A \arrow[l] & B \arrow[l] & C \arrow[l] & D \arrow[l] & E \arrow[l] & \mathbf{I} \arrow[l] \arrow[uul] \arrow[ul] \arrow[dl] \\
 & & & & & H \arrow[ulll] &
\end{tikzcd}$$

Finally, to prevent an exponential explosion due to local ambiguity, we check if a new partial derivation is equivalent to any existing partial derivations. If it is, we keep track of the ways the given node can be generated and then merge them into a single node. This is referred to as local ambiguity packing by Tomita (1988) and allows shift-reduce parsing to be performed in polynomial time. In the example below, the new reduction rules result in two new J nodes. These two nodes are merged to form a single node as they are equivalent.

$$\begin{tikzcd}
 & & & & & F \arrow[ddll] & \\
 & & & & & G \arrow[dll] & \mathbf{J} \arrow[dlll, bend right] \\
\emptyset & A \arrow[l] & B \arrow[l] & C \arrow[l] & D \arrow[l] & E \arrow[l] & I \arrow[l] \arrow[uul] \arrow[ul] \arrow[dl] \\
 & & & & & H \arrow[ulll] &
\end{tikzcd}$$

Reduction Rules			
J	$\leftarrow$	F	I
J	$\leftarrow$	G	I

When parsing an n word sentence, there are n possible stages in the GSS. We refer to these stages as *frontiers*, with the k^{th} frontier containing all partial derivations that contain a total span of k. In CKY chart terms, a frontier can be considered as representing all cells on the diagonal from the top left to the bottom right, as seen in Figure 3.

Figure 2 represents an incomplete sentence processed using a GSS-based shift-reduce CCG parser. The frontier for the word with contains two heads, $((S\backslash NP)\backslash(S\backslash NP))/NP$ and $(NP\backslash NP)/NP$. When the CCG category for the

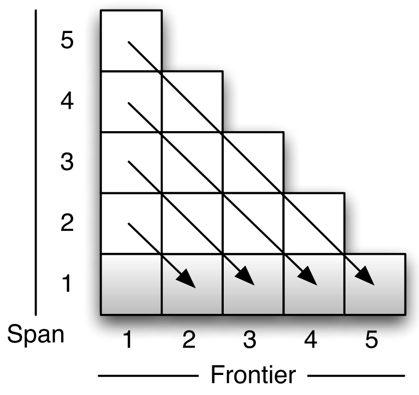

Figure 3: An illustration of the relation between the chart in CKY and the graph-structured stack in SR

word binoculars is shifted on to the GSS, it connects to both of the previous heads. As the category for the word binoculars is an NP, we can then reduce the stack by applying combinatory rules from CCG to both of the heads found in the previous frontier. In light gray, we show the full derivation for "John with binoculars".

During the parsing process, we start with an empty GSS. During the *shift* step, we add all the possible CCG categories provided by the supertagger for the k^{th} word to the GSS and connect each category to all of the head categories on the GSS. Next, we attempt all possible *reduce* operations on the partial derivations in the current frontier. In CCG shift-reduce parsing, these reduce operations are the CCG combinatory rules. If a reduction is possible, we create a new top partial derivation from the result and place it in the k^{th} frontier.

4 Frontier Pruning

The purpose of frontier pruning is to cut down the search space of the parser by only considering partial derivations that are likely to be in the highest-scoring derivation. Like adaptive supertagging, it exploits the idea that the only partial derivations the parser needs to generate are those used by the highest-scoring derivation. The model is trained using the parser's initial unpruned output and aims to distinguish between partial derivations that are necessary and those that are not. By eliminating a large number of those unnecessary partial derivations, parsing ambiguity is significantly decreased.

This approach is similar to beam search as frontier pruning removes partial derivations once it is likely they will not be used in the highest-scoring derivation. Beam search prunes nodes that are below a multiple (β) of the highest-scoring node in the frontier. For certain instances, such as n-best re-ranking, beam search would be preferred as derivations without the highest score are still useful in the parsing process. For one best parsing, however, the parser may waste time generating these additional derivations when it could be known in advanced that they will not be used. This could occur during attachment ambiguity where, although the parser is guaranteed to select one attachment, the other attachment may be constructed as it is valid and still competitive when considered by beam search's criteria.

5 Experiments

The only modifications are to the core parsing algorithm, which involves replacing CKY with SR, and to the parsing process via pruning. As the decoder and base models used for selecting the best-scoring derivation remain unchanged, any improvements seen are from an improved parsing process.

The C&C code base has been optimised for CKY parsing and we have made only limited attempts to optimise specifically for the shift-reduce approach. Due to this, the speed of the SR parser is 34% slower than the CKY parser. As the frontier pruning is implemented on the SR parser, all speeds will be relative to the SR baseline. For the frontier pruning SR parser to be competitive with the CKY parser, a speed improvement of 34% or more must be achieved.

5.1 Training and Processing

We train a binary averaged perceptron model (Collins, 2002) on parser output generated by the SR C&C parser using the standard parsing model. Once the base parser has successfully processed a sentence, all partial derivations that lead to the highest-scoring derivation are marked. For each partial derivation in the GSS, the perceptron model attempts to classify whether it was part of the marked set. If the classification is incorrect, the perceptron model updates the weights appropriately.

During processing, pruning occurs as each fron-

Feature Type	Example
Category	$S\backslash NP$
Binary Composition	$(S\backslash NP)/NP$ and NP
Forward Application	True
Head Word	*saw*
Head POS	VBD
Previous Frontier	NP
Next Frontier	$((S\backslash NP)\backslash(S\backslash NP))/NP$
Next Frontier	$(NP\backslash NP)/NP$

Table 1: Example features extracted from $S\backslash NP$ in the third frontier of Figure 2. For the frontier features, bold represents the highest-scoring feature selected for contribution to the classification decision.

tier is developed. For each partial derivation, the perceptron model classifies whether the partial derivation is likely to be used in the highest-scoring derivation. If not, the partial derivation is removed from the frontier, eliminating any paths that the partial derivation would have generated. Perfect frontier pruning would allow only a single derivation, specifically the highest-scoring one, to develop.

5.2 Model Features

For frontier pruning to be effective, the model must be able to accurately distinguish between partial derivations that will be used in the highest-scoring derivation and those that shall not. As the features of the C&C parser dictate the highest-scoring derivation, the features used for frontier pruning have been chosen to be similar. For a full description of the features used in the C&C parser, refer to Clark and Curran (2007).

Each partial derivation is given a base set of features derived from the current category. The initial features include a NULL which all categories receive, the CCG category itself and whether the category was assigned by the supertagger. There are also features that encode rule instantiation, including whether the category was created by type raising, a lexical rule, or any CCG combinatory rule. If the category was created by a CCG combinatory rule, the type of rule (such as forward/backward application and so on) is included as a feature.

Features representing the past decisions the parser has made are also included. Note that *current* represents the current category and *left*/*right* is the current category's left or right child respectively. For unary categories, a tuple of [current,current→left] is included as a feature. For binary categories, a tuple of [current→left, current, current→right] is included. If a category is a leaf, then two features [current, word] and [current, POS] are included. Features representing the root category of the partial derivation are also included, encoding the category head's word and POS tag.

Finally, additional features are added that represent the possible future parsing decisions. This is achieved by adding information about the remaining partial derivations on the stack (the past frontier) and the future incoming partial derivations (the next frontier). These do not exist in the C&C parser and are only possible due to the implementation of the GSS. For each category in the previous frontier, a feature is added of the type [previous, current]. For the next frontier, which is only composed of supertags at this point, the feature is [current, next]. These features allow the pruning classifier to determine whether the current category is likely to be active in any other reductions in future parsing work. As we only want to score the optimal path using the previous and next features, only the highest weighted of these features are selected. The rest of the previous and next features are discarded and do not contribute to the classification.

An example of this can be seen in Table 1, where the features for the partial derivation of $S\backslash NP$ are enumerated.

These features differ to the traditional features used by shift-reduce parsers due to the addition of the GSS. As traditional shift-reduce parsing only considers a single derivation at a time, it is trivial to include history further back than the current category's previous frontier. As GSS-based shift-reduce parsing encodes an exponential number of states, however, the overhead of unpacking these states into a feature representation is substantial. Our approximation of selecting the highest weighted previous and next frontier features approximates the non-deterministic shift-reduce solution.

5.3 Improving Marked Set Recall

Compared to the unmarked set, the marked set of partial derivations used to create the highest-scoring derivation is small. If a single CCG category from the marked set is pruned accidentally, the accuracy may be negatively impacted. The loss of a single category may even mean it is impossible to form a spanning analysis.

To prevent this loss of accuracy and coverage, the recall of the marked set needs to be improved. This can be achieved by biasing the binary perceptron algorithm towards a certain class, trading precision for recall. Traditionally, a binary perceptron classifier returns true if $w \cdot x > 0$, else false, with w being a vector of weights for each feature and x being a binary vector indicating whether a feature was active.

By providing a manual bias λ, $w \cdot x > \lambda$, we can bias the classifier towards a class. The value of λ modifies the perceptron threshold level, allowing us to improve the recall of the marked set by lowering the precision. The value for λ is obtained manually through the use of a development set.

Identifying the optimal threshold value is important. Too high a recall value would prevent pruning any parts of the parse tree whilst too low a threshold reverts back to traditional unpruned parsing. Due to the overheads involved in the frontier pruning process, ineffective frontier pruning may also be slower than traditional parsing, especially for an optimised parser such as the C&C parser. This value is determined experimentally using a development dataset.

5.4 Balancing Pruning Features and Speed

For frontier pruning to produce a speed gain, enough of the search space must be pruned in order to compensate for the additional computational overhead of the pruning step itself. This is a challenge as the C&C parser is written in *C++* with a focus on efficiency and already features substantial lexical pruning due to the use of supertagging.

For this reason, there were instances where expressive features needed to be traded for simpler features in the frontier pruning process. Whilst these simpler features may not prune as effectively, they take far less time to compute and result in higher speed gains than complex features with a further reduced search space. The complexity of the frontier pruning features may be dictated by the speed of the core parser itself, with more expressive features being possible if the core parser is slower.

The implementation of these features also had to focus on efficiency. To decrease the stress and improve memory locality of the hash table storing the feature weights, only a subset of features were stored. This feature subset was obtained from the gold standard training data as it contains far less ambiguity than the same training data which uses lexical categories supplied by the supertagger.

Hash tables were used for storing the relevant feature weights. Simple hash based feature representation were used for associating features with weights to reduce the complexity of equivalence checking. The hash values of features that were to be reused were also cached to prevent recalculation, substantially decreasing the computational overhead of feature calculation.

6 Evaluation

Our experiments are performed using CCGbank which was split into three subsets for training (Sections 02-21), development (Section 00), and the final evaluation (Section 23). The performance is measured in terms of sentence coverage, accuracy and parsing time. The accuracy is computed as F-score over the extracted labeled and unlabeled CCG dependencies found in CCGbank. All unmarked experiments use gold standard POS tags whilst experiments marked *Auto* use automatically assigned POS tags using the C&C POS tagger.

7 Results

7.1 Training the Frontier Pruning Algorithm

To establish bounds on the potential search space reduction, the size of the marked set compared to the total tree size was tracked over all sentences in the training data. This represents the size of the tree after optimal pruning occurs. Two figures are presented, one with gold supertags and the other with supertags supplied by the C&C supertagger. Gold represents the reduction in search space possible when only the correct CCG categories are used to parse the sentence. In contrast, the C&C supertagger may apply multiple CCG categories to improve supertagging accuracy, resulting in higher ambiguity and greater potential search space reductions.

As can be seen in Table 2, the size of the marked set is 10 times smaller for gold supertags and 15 times smaller for automatically supplied supertags.

Task	Acc.
Marked set recall (gold supertags)	84.4%
Marked set recall	72.9%
Average pruned size (gold supertags)	9.6%
Average pruned size	6.7%

Table 2: Recall of the marked set from the frontier pruning algorithm across all trees and the size of the pruned tree compared to the original tree.

This places an upper-bound on the potential speed improvement the parser may see due to aggressive frontier pruning.

The recall of the marked set was low for both gold supertags and automatically assigned supertags. This suggests the need for a modified perceptron threshold level in order to increase the recall of the marked set.

7.2 Tuning the Perceptron Threshold Level

Tuning the perceptron threshold level, as described in the previous section, has an important impact on frontier pruning. If the baseline parser cannot form a spanning analysis with the supertags initially supplied by the supertagger, it requests more supertags. Aggressive frontier pruning may counter-intuitively result in a slower parser as the parser spends more time attempting to unsuccessfully parse the sentence with an increasingly large number of supertags. By tuning the perceptron threshold level we can prevent potential slow-downs caused by aggressive pruning.

To optimise the threshold level, experiments were performed on the development portion of CCGbank, Section 00. The results are shown in Table 3. Decreasing the perceptron thresholds level (λ) is shown to decrease the speed of the parser substantially without increasing the accuracy. For extremely low values of λ, frontier pruning will keep partial derivations previously discarded as the perceptron classifier becomes biased towards recall. For a sufficiently low value, the accuracy would reach the same levels as the CKY and SR C&C parsers, but the speed would be far too slow due to the computational overhead of frontier pruning added to the small reduction in the search space. More work on fine-tuning the feature representation and allowing for more expressive features in a faster manner will be required.

Model	Coverage	lf.	uf.	Speed
	(%)	(%)	(%)	(sents/sec)
CKY C&C	99.01	86.37	92.56	55.6
SR C&C	98.90	86.35	92.44	48.6
FP $\lambda = 0$	99.01	86.11	92.25	61.1
FP $\lambda = -1$	99.06	86.16	92.23	56.4
FP $\lambda = -2$	99.01	86.13	92.19	53.9
FP $\lambda = -3$	99.06	86.15	92.21	49.0
CKY C&C Auto	98.90	84.30	91.26	56.2
SR C&C Auto	98.85	84.27	91.10	47.5
FP $\lambda = 0$ Auto	98.80	84.09	90.97	60.0

Table 3: Comparison to baseline parsers and analysis of the impact of threshold levels on frontier pruning (FP). The perceptron threshold level is referred to as λ. All results are against the development dataset, Section 00 of CCGbank, which contains 1,913 sentences.

For $\lambda = 0$, however, frontier pruning increases the parser's speed by 25.7% compared to the baseline GSS-based SR parser on which the frontier pruning operates. There is also a small 9.8% speed increase compared to the CKY baseline parser. The F-score for both labeled and unlabeled dependencies is negatively impacted though.

7.3 Speed Improvements during Evaluation

Table 4 reports the impact frontier pruning has on speed compared to the baseline CKY and SR C&C parsers. Frontier pruning has improved the speed of the GSS-based SR C&C parser by 39%, an improvement over the speed increase seen during evaluation. Longer sentences seem to have a higher impact on the speed of the frontier pruning algorithm due to the increased computational complexity of feature generation. This indicates that implementing a form of beam search on top of this may be beneficial, keeping on the top k scoring states in a frontier. Currently all partial derivations that are greater than the perceptron threshold level λ are kept.

8 Discussion and future work

As the C&C parser is already highly tuned and thus extremely fast, the optimal balance between feature expressiveness and accurate pruning is difficult to achieve. However, there was still room for improvement. This suggests that on slower parsers than the C&C parser, frontier pruning may have a much more substantial impact on parsing speeds.

Model	Coverage (%)	lf. (%)	uf. (%)	Speed (sents/sec)
CKY C&C	99.34	86.79	92.50	96.3
SR C&C	99.58	86.78	92.41	71.3
FP $\lambda = 0$	99.38	86.51	92.25	95.4
CKY C&C Auto	99.25	84.59	91.20	82.0
SR C&C Auto	99.50	84.53	91.09	61.2
FP $\lambda = 0$ Auto	99.29	84.29	90.88	84.9

Table 4: Final evaluation on Section 23 of CCGbank for the top performing models from Table 3, containing 2,407 sentences.

More work needs to be done on reducing the number of computationally intensive feature look-ups and calculations. Even when using the gold-standard subset of the features, the feature look-up process accounts for the majority of the slow-down that the frontier pruning algorithm causes.

The C&C code has been highly optimised to suit CKY parsing. It should be possible to improve the GSS parser to be directly competitive with the CKY implementation. The frontier pruning provides speed increases for the GSS parser, allowing it to be competitive with the original CKY parser, but with an improved GSS parser, we could expect further improvements over the original CKY parser.

Finally, we are still using the separate maximum entropy model and decoder to find the best derivation. If we add more features to the perceptron model, it may be possible to use it for frontier pruning and finding the best derivation.

9 Conclusion

We present a shift-reduce CCG parser that can explore all possible analyses in polynomial time through the use of a graph-structured stack (GSS). Whilst this parser is 34% slower than the CKY parser on which it is based, it can parse 60 sentences per second whilst exploring the full search space. We show that by performing frontier pruning on the GSS and reducing this search space, the speed of the GSS parser can be improved by 39% whilst only incurring a small accuracy penalty. This allows for shift-reduce parsing to attain speeds directly competitive with the CKY parser, whilst allowing all the potential advantages of a semi-incremental parser.

We have also shown that whilst pruning is occurring at the lexical level due to supertagging, substantial speed-ups are still possible by performing pruning during the parsing process itself. This has also illustrated the difficulty in balancing expressive features and feature calculations overhead that frontier pruning needs to achieve.

Our approach uses the output of the original C&C parser as training data, and so we can use any amount of parser output to train the system. This self-training has been shown to be highly effective in adaptive supertagging for increasing parser speed (Kummerfeld et al., 2010). The final result will be a substantially faster wide-coverage CCG parser that can be used for large-scale NLP applications.

Acknowledgements

This work was supported by the Capital Markets Co-operative Research Centre, an Australian Research Council Discovery grant DP1097291 and a University of Sydney Honours Scholarship. We thank the anonymous reviewers for their insightful feedback.

References

Alred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Volume I: Parsing*. Prentice-Hall.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Ming-Wei Chang, Quang Do, and Dan Roth. 2006. Multilingual Dependency Parsing: A Pipeline Approach. In Nicolas Nicolov, editor, *Recent Advances in Natural Language Processing*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 173–180, Ann Arbor, Michigan, USA, June.

Stephen Clark and James R. Curran. 2004. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 282–288, Geneva, Switzerland, August.

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, Julia Hockenmaier, and Mark Steedman. 2002. Building Deep Dependency Structures using a Wide-Coverage CCG Parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 327–334, Philadelphia, Pennsylvania, USA, July.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-03)*, pages 1–8.

Tim Dawborn and James R. Curran. 2009. CCG parsing with one syntactic structure per n-gram. In *Proceedings of the Australasian Language Technology Association Workshop 2009 (ALTA-09)*, pages 71–79, Sydney, Australia, December.

Bojan Djordjevic. 2006. Efficient Combinatory Categorial Grammar Parsing. In *Proceedings of the 2006 Australasian Language Technology Workshop (ALTW-06)*, pages 3–10, Sydney, Australia, December.

Julia Hockenmaier and Mark Steedman. 2002. Generative Models for Statistical Parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 335–342, Philadelphia, PA.

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.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, pages 1077–1086, Uppsala, Sweden, July.

Tadao Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Dan Klein and Christopher D. Manning. 2003. A* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL-03)*, volume 3, pages 119–126.

Jonathan K. Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. 2010. Faster Parsing by Supertagger Adaptation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*, pages 345–355, Uppsala, Sweden, July.

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

Stephen Merity. 2011. *Integrated Tagging and Pruning via Shift-Reduce CCG Parsing*. Honours Thesis, The University of Sydney, Sydney, Australia.

Joakim Nivre and Mario Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proceedings of the 18nd International Conference on Computational Linguistics (COLING-04)*, pages 64–70, Geneva, Switzerland, August.

Adam Pauls and Dan Klein. 2009. K-Best A* Parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 958–966, Singapore, August.

Brian Roark and Kristy Hollingshead. 2009. Linear Complexity Context-Free Parsing Pipelines via Chart Constraints. In *Proceedings of 2009 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-09)*, pages 647–655, Boulder, Colorado, June.

Kenji Sagae and Alon Lavie. 2005. A Classifier-Based Parser with Linear Run-Time Complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT-05)*, pages 125–132, Vancouver, British Columbia, Canada, October.

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

Masaru Tomita. 1988. Graph-structured Stack and Natural Language Parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 249–257, Buffalo, New York, USA, June.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. *Proceedings of International Conference on Parsing Technologies (IWPT-03)*, pages 195–206.

Daniel H. Younger. 1967. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10(2):189–208, February.

Yue Zhang and Stephen Clark. 2011. Shift-Reduce CCG Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-11:HLT)*, pages 683–692, Portland, Oregon, USA, June.

Yue Zhang, Byung-Gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010. Chart Pruning for Fast Lexicalised-Grammar Parsing. In *Proceedings of the COLING 2010 Poster Sessions*, pages 1471–1479, Beijing, China, August.