Syntax-Based Grammaticality Improvement using CCG and Guided Search

Yue Zhang
University of Cambridge
Computer Laboratory
yue.zhang@cl.cam.ac.uk

Stephen Clark
University of Cambridge
Computer Laboratory
stephen.clark@cl.cam.ac.uk

Abstract

Machine-produced text often lacks grammaticality and fluency. This paper studies grammaticality improvement using a syntax-based algorithm based on CCG. The goal of the search problem is to find an optimal parse tree among all that can be constructed through selection and ordering of the input words. The search problem, which is significantly harder than parsing, is solved by guided learning for best-first search. In a standard word ordering task, our system gives a BLEU score of 40.1, higher than the previous result of 33.7 achieved by a dependency-based system.

1 Introduction

Machine-produced text, such as SMT output, often lacks grammaticality and fluency, especially when using n-gram language modelling (Knight, 2007). Recent efforts have been made to improve grammaticality using local language models (Blackwood et al., 2010) and global dependency structures (Wan et al., 2009). We study grammaticality improvement using a syntax-based system.

The task is effectively a text-to-text generation problem where the goal is to produce a grammatical sentence from an ungrammatical and fragmentary input. The input can range from a bag-of-words (Wan et al., 2009) to a fully-ordered sentence (Blackwood et al., 2010). A general form of the problem is to construct a grammatical sentence from a set of un-ordered input words. However, in cases where the base system produces fluent subsequences within the sentence, constraints on the choice and order of certain words can be fed to the grammaticality improvement system. The input may also include words beyond the output of the base system, e.g. extra words from the SMT lattice, so that content word insertion and deletion can be performed implicity via word selection.

We study the above task using CCG (Steedman, 2000). The main challenge is the search problem, which is to find an optimal parse tree among all that can be constructed with any word choice and order from the set of input words. We use an approximate best-first algorithm, guided by learning, to tackle the more-than-factorial complexity. Beam-search is used to control the volume of accepted hypotheses, so that only a very small portion of the whole search space is explored. The search algorithm is guided by perceptron training, which ensures that the explored path in the search space consists of highly probable hypotheses. This framework of best-first search guided by learning is a general contribution of the paper, which could be applied to problems outside grammaticality improvement.

We evaluate our system using the generation task of word-order recovery, which is to recover the original word order of a fully scrambled input sentence (Bangalore et al., 2000; Wan et al., 2009). This problem is an instance of our general task formulation, but without any input constraints, or content word selection (since all input words are used). It is straightforward to use this task to evaluate our system and compare with existing approaches. Our system gave 40.1 BLEU score, higher than the dependency-based system of Wan et al. (2009), for which a BLEU score of 33.7 was reported.

2 The Grammar

Combinatory Categorial Grammar (CCG; Steedman (2000)) is a lexicalized grammar formalism, which associates words with lexical categories. Lexical categories are detailed grammatical labels, typically expressing subcategorisation information. CCG, and parsing with CCG, has been described in detail elsewhere (Clark and Curran, 2007; Hockenmaier, 2003); here we provide only a short description.

During CCG parsing, adjacent categories are combined using CCG's combinatory rules. For example, a verb phrase in English ($S\backslash NP$) can combine with an NP to its left:

$$NP \;\; S\backslash NP \Rightarrow S$$

In addition to binary rule instances, such as the one above, there are also unary rules which operate on a single category in order to change its type. For example, forward type-raising can change a subject NP into a complex category looking to the right for a verb phrase:

$$NP \Rightarrow S/(S\backslash NP)$$

Following Hockenmaier (2003), we extract the grammar by reading rule instances directly from the derivations in CCGbank (Hockenmaier and Steedman, 2007), rather than defining the combinatory rule schema manually as in Clark and Curran (2007).

3 The Search Algorithm

The input to the search algorithm is a set of words, each word having a count that specifies the maximum number of times it can appear in the output. Typically, most input words can occur only once in the output. However, punctuation marks and function words can be given a higher count. Depending on the fluency of the base output (e.g. the output of the base SMT system), some constraints can be given to specific input words, limiting their order or identifying them as an atomic phrase, for example.

The goal of the search algorithm is to find an optimal parse tree (including the surface string) among all that can be constructed via selecting and ordering a subset of words from the input multiset. The complexity of this problem is much higher than a typical parsing problem, since there is an exponential number of word choices for the output sentence, each with a factorial number of orderings. Moreover, dynamic programming packing for parsers, such as a CKY chart, is not applicable, because of the lack of a fixed word order.

We perform approximate search using a best-first algorithm. Starting from single words, candidate parses are constructed bottom-up. Similar to a best-first parser (Caraballo and Charniak, 1998), the highest scored hypothesis is expanded first. A hypothesis is expanded by applying CCG unary rules to the hypothesis, or by combining the hypothesis with existing hypotheses using CCG binary rules.

We use beam search to control the number of accepted hypotheses, so that the computational complexity of expanding each hypothesis is linear in the size of the beam. Since there is no guarantee that a goal hypothesis will be found in polynomial time, we apply a robustness mechanism (Riezler et al., 2002; White, 2004), and construct a default output when no goal hypothesis is found within a time limit.

3.1 Data Structures

Edges are the basic structures that represent hypotheses. Each edge is a CCG constituent, spanning a sequence of words. Similar to partial parses in a typical chart parser, edges have recursive structures. Depending on the number of subedges, edges can be classified into *leaf edges*, *unary edges* and *binary edges*. Leaf edges, which represent input words, are constructed first in the search process. Existing edges are expanded to generate new edges via unary and binary CCG rules. An edge that meets the output criteria is called a *goal edge*. In the experiments of this paper, we define a goal edge as one that includes all input words the correct number of times.

The *signature* of an edge consists of the category label, surface string and head word of the constituent. Two edges are *equivalent* if they share the same signature. Given our feature definitions, a lower scoring edge with the same signature as a higher scoring edge cannot be part of the highest scoring derivation.

The number of words in the surface string of an edge is called the *size* of the edge. Other important substructures of an edge include a bitvector and an array, which stores the indices of the input words that the edge contains. Before two edges are combined using a binary CCG rule, an *input check* is performed to make sure that the total count for a word from the two edges does not exceed the count for that word in the input. Intuitively, an edge can record the count of each unique input word it contains, and perform the input check in linear time. However, since most input words typically occur once, they can be indexed and represented by a bitvector, which allows a constant time input check. The few multiple-occurrence words are stored in a count array.

In the best-first process, edges to be expanded are ordered by their scores, and stored in an *agenda*. Edges that have been expanded are stored in a *chart*. There are many ways in which edges could be ordered and compared. Here the chart is organised as a set of beams, each containing a fixed number of edges with a particular size. This is similar to typical decoding algorithms for phrase-based SMT (Koehn, 2010). In each beam, edges are ordered by their scores, and low score edges are pruned. In addition to pruning by the beam, only the highest scored edge is kept among all that share the same signature.

3.2 The Search Process

Figure 1 shows pseudocode for the search algorithm. During initialization, the agenda (a) and chart (c) are cleared. All candidate lexical categories are assigned to each input word, and the resulting leaf edges are put onto the agenda.

In the main loop, the best edge (e) is popped from the agenda. If e is a goal hypothesis, it is appended to a list of goals (*goal*), and the loop is continued without e being expanded. If e or any equivalent edge $\tilde{e}$ of e is already in the chart, the loop continues without expanding e. It can be proved that any edge in the chart must have been combined with $\tilde{e}$, and therefore the expansion of e is unnecessary.

Edge e is first expanded by applying unary rules, and any new edges are put into a list (*new*). Next, e is matched against each existing edge $\tilde{e}$ in the chart. e and $\tilde{e}$ can be combined if they pass the input check, and there is a binary rule in which the constituents are combined. e and $\tilde{e}$ are combined in both possible orders, and any resulting edge is added to *new*.

At the end of each loop, edges from *new* are added to the agenda, and *new* is cleared. The loop continues until a stopping criterion is met. A typical stopping condition is that *goal* contains N goal edges.

```
a ← INITAGENDA(input)
c ← INITCHART()
new ← []
goal ← []
while not STOP(goal, time):
    e ← POPBEST(a)
    if GOALTEST(e)
        APPEND(goal, e)
        continue
    for $\tilde{e}$ in c:
        if EQUIV($\tilde{e}$, e):
            continue
    for e' in UNARY(e, grammar):
        APPEND(new, e')
    for $\tilde{e}$ in c:
        if CANCOMBINE(e, $\tilde{e}$):
            e' ← BINARY(e, $\tilde{e}$, grammar)
            APPEND(new, e')
        if CANCOMBINE($\tilde{e}$, e):
            e' ← BINARY($\tilde{e}$, e, grammar)
            APPEND(new, e')
    for e' in new:
        ADD(a, e')
    ADD(c, e)
    new ← []
```

Figure 1: The search algorithm.

We set N to 1 in our experiments. For practical reasons we also include a timeout stopping condition. If no goal edges are found before the timeout is reached, a default output is constructed by the following procedure. First, if any two edges in the chart pass the input check, and the words they contain constitute the full input set, they are concatenated to form an output string. Second, when no two edges in the chart meet the above condition, the largest edge $\tilde{e}$ in the chart is chosen. Then edges in the chart are iterated over in the larger first order, with any edge that passes the input check with $\tilde{e}$ concatenated with $\tilde{e}$ and $\tilde{e}$ updated. The final $\tilde{e}$, which can be shorter than the input, is taken as the default output.

4 Model and Features

We use a discriminative linear model to score edges, where the score of an edge e is calculated using the global feature vector $\Phi(e)$ and the parameter vector

$\vec{w}$ of the model.

$$\textsc{Score}(e) = \Phi(e) \cdot \vec{w}$$

$\Phi(e)$ represents the counts of individual features of e. It is computed incrementally as the edge is built. At each constituent level, the incremental feature vector is extracted according to the feature templates from Table 1, and we use the term *constituent level vector* ϕ to refer to it. So for any edge e, $\phi(e)$ consists of features from the top rule of the hierarchical structure of e. $\Phi(e)$ can be written as the sum of $\phi(e')$ of all recursive subedges e' of e, including e itself:

$$\Phi(e) = \sum_{e' \in e} \phi(e')$$

The parameter update in Section 5 is in terms of constituent level vectors.

The features in Table 1 represent patterns including the constituent label; the head word of the constituent; the size of the constituent; word, POS and lexical category N-grams resulting from a binary combination; and the unary and binary rules by which the constituent is constructed. They can be classified roughly into "parsing" features (those about the parse structure, such as the binary rule) and "generation features" (those about the surface string, such as word bigrams), although some features, such as "rule + head word + non-head word", contain both types of information.

5 The Learning Algorithm

The statistical model plays two important roles in our system. First, as in typical statistical systems, it is expected to give a higher score to a more correct hypothesis. Second, it is also crucial to the speed of the search algorithm, since the best-first mechanism relies on a model to find goal hypotheses efficiently. As an indication of the impact of the model on efficiency, if the model parameters are set to all zeros, the search algorithm cannot find a result for the first sentence in the development data within two hours.

We perform training on a corpus of CCG derivations, where constituents in a gold-standard derivation serve as gold edges. The training algorithm runs the decoder on each training example, updating the model when necessary, until the gold goal edge is recovered. We use the perceptron (Rosenblatt, 1958) to perform parameter updates. The traditional perceptron has been adapted to structural prediction (Collins, 2002) and search optimization problems (Daumé III and Marcu, 2005; Shen et al., 2007). Our training algorithm can be viewed as an adaptation of the perceptron to our best-first framework for search efficiency and accuracy.

condition	feature
all edges	constituent + size
	constituent + head word
	constituent + size + head word
	constituent + head POS
size > 1	constituent + leftmost word
	constituent + rightmost word
	consti. + leftmost POS bigram
	consti. + rightmost POS bigram
	consti. + lmost POS + rmost POS
binary edges	the binary rule
	the binary rule + head word
	rule + head word + non-head word
	bigrams resulting from combination
	POS bigrams resulting from combi.
	word trigrams resulting from combi.
	POS trigrams resulting from combi.
	resulting lexical category trigrams
	resulting word + POS bigrams
	resulting POS + word bigrams
	resulting POS + word + POS trigrams
unary edges	unary rule
	unary rule + headw

Table 1: Feature template definitions.

We choose to update parameters as soon as the best edge from the agenda is not a gold-standard edge. The intuition is that all gold edges are forced to be above all non-gold edges on the agenda. This is a strong precondition for parameter updates. An alternative is to update when a gold-standard edge falls off the chart, which corresponds to the precondition for parameter updates of Daumé III and Marcu (2005). However, due to the complexity of our search task, we found that reasonable training efficiency cannot be achieved by the weaker alternatives. Our updates lead both to correctness (edges in the chart are correct) and efficiency (correct edges are found at the first possible opportunity).

During a perceptron update, an incorrect prediction, corresponding to the current best edge in the agenda, is penalized, and the corresponding gold edge is rewarded. However, in our scenario it is not obvious what the corresponding gold edge should be, and there are many ways in which the gold edge could be defined. We investigated a number of alternatives, for example trying to find the "best match" for the incorrect prediction. In practice we found that the simple strategy of selecting the lowest scored gold-standard edge in the agenda was effective, and the results presented in this paper are based on this method.

After an update, there are at least two alternative methods to continue. The first is to reinitialize the agenda and chart using the new model, and continue until the current training example is correctly predicted. This method is called *aggressive training* (Shen et al., 2007). In order to achieve reasonable efficiency, we adopt a second approach, which is to continue training without reinitializing the agenda and chart. Instead, only edges from the top of the agenda down to the lowest-scoring gold-standard edge are given new scores according to the new parameters.

Figure 2 shows pseudocode for the learning algorithm applied to one training example. The initialization is identical to the test search, except that the list of goal edges is not maintained. In the main loop, the best edge e is popped off the agenda. If it is the gold goal edge, the training for this sentence finishes. If e is not a gold edge, parameter updates are performed and the loop is continued with e being discarded. Only gold edges are pushed onto the chart throughout the training process.

When updating parameters, the current non-gold edge (e) is used as the negative example, and the smallest gold edge in the agenda (*minGold*) is used as the corresponding positive example. The model parameters are updated by adding the constituent level feature vector (see Section 4) of *minGold*, and subtracting the constituent level feature vector of e. Note that we do not use the global feature vector in the update, since only the constituent level parameter vectors are compatible for edges with different sizes. After a parameter update, edges are rescored from the top of the agenda down to *minGold*.

The training algorithm iterates through all training examples N times, and the final parameter vector is used as the model. In our experiments, N is chosen according to results on development data.

```
$a \leftarrow$ INITAGENDA(input)
$c \leftarrow$ INITCHART()
new $\leftarrow$ []
while true:
  $e \leftarrow$ POPBEST($a$)
  if GOLD($e$) and GOALTEST($e$):
    return
  if not GOLD($e$):
    popped $\leftarrow$ []
    $n \leftarrow 0$
    while $n <$ GOLDCOUNT($a$):
      $\tilde{e} \leftarrow$ POPBEST($a$)
      APPEND(popped, $\tilde{e}$)
      if GOLD($\tilde{e}$):
        minGold $\leftarrow \tilde{e}$
        $n \leftarrow n + 1$
    $\vec{w} \leftarrow \vec{w} - \phi(e) + \phi($minGold$)$
    for $\tilde{e}$ in popped:
      RECOMPUTESCORE($\tilde{e}$)
      ADD($a$, $\tilde{e}$)
    for $\tilde{e}$ in $c$:
      RECOMPUTESCORE($\tilde{e}$)
    continue
  for $e'$ in UNARY($e$, grammar):
    APPEND(new, $e'$)
  for $\tilde{e}$ in $c$:
    if CANCOMBINE($e$, $\tilde{e}$):
      $e' \leftarrow$ BINARY($e$, $\tilde{e}$, grammar)
      APPEND(new, $e'$)
    if CANCOMBINE($\tilde{e}$, $e$):
      $e' \leftarrow$ BINARY($\tilde{e}$, $e$, grammar)
      APPEND(new, $e'$)
  for $e'$ in new:
    ADD($a$, $e'$)
  ADD($c$, $e$)
  new $\leftarrow$ []
```

Figure 2: The learning algorithm.

6 Experiments

We use CCGBank (Hockenmaier and Steedman, 2007) for experimental data. CCGbank is the CCG version of the Penn Treebank. Sections 02–21 are used for training, section 00 is used for development and section 23 for the final test.

Original sentences from CCGBank are transformed into bags of words, with sequence information removed, and passed to our system as input data. The system outputs are compared to the original sentences for evaluation. Following Wan et al. (2009), we use the BLEU metric (Papineni et al., 2002) for string comparison. Whilst BLEU is not an ideal measure of fluency or grammaticality, being based on n-gram precision, it is currently widely used for automatic evaluation and allows us to compare directly with existing work (Wan et al., 2009).

In addition to the surface string, our system also produces the CCG parse given an input bag of words. The quality of the parse tree can reflect both the grammaticality of the surface string and the quality of the trained grammar model. However, there is no direct way to automatically evaluate parse trees since output word choice and order can be different from the gold-standard. Instead, we indirectly measure parse quality by calculating the precision of CCG lexical categories. Since CCG lexical categories contain so much syntactic information, they provide a useful measure of parse quality. Again because the word order can be different, we turn both the output and the gold-standard into a bag of word/category pairs, and calculate the percentage of matched pairs as the lexical category precision.

For fair comparison with Wan et al. (2009), we keep base NPs as atomic units when preparing the input. Wan et al. (2009) used base NPs from Penn Treebank annotation, while we extract base NPs from the CCGBbank by taking as base NPs the NPs that do not recursively contain other NPs. These base NPs mostly correspond to the base NPs from the Penn Treebank. In the training data, there are 242,813 Penn Treebank base NPs with an average size of 1.09, and 216,670 CCGBank base NPs with an average size of 1.19.

6.1 Development Tests

Table 2 shows a set of development experiment results after one training iteration. Three different methods of assigning lexical categories are used. The first ("dictionary") is to assign all possible lexical categories to each input word from the dictionary. The lexical category dictionary is built using the training sections of CCGBank. For each word occurring more than 20 times in the corpus, the dictionary has an entry with all lexical categories the word has been seen with. For the rest of the words, the dictionary maintains an entry for each POS which contains all lexical categories it has been seen with. There are on average 26.8 different categories for each input word by this method.

Method	Timeout	BLEU	Length ratio	Timeout ratio
dictionary	0.5s	34.98	84.02	62.26
	1s	35.40	85.66	57.87
	5s	36.27	89.05	45.79
	10s	36.45	89.13	42,13
	50s	37.07	92.52	32.41
$\beta = 0.0001$	0.5s	36.54	84.26	66.07
	1s	37.50	86.69	58.22
	5s	38.75	90.15	43.23
	10s	39.14	91.35	38.36
	50s	39.58	93.09	30.53
$\beta = 0.075$	0.5s	40.87	85.66	61.27
	1s	42.04	87.99	53.11
	5s	43.99	91.20	40.30
	10s	44.23	92.14	35.70
	50s	45.08	93.70	29.43

Table 2: Development tests using various levels of lexical categories and timeouts, after one training iteration.

In practice, it is often unnecessary to leave lexical category disambiguation completely to the grammaticality improvement system. When it is reasonable to assume that the input sentence for the grammaticality improvement system is sufficiently fluent, a list of candidate lexical categories can be assigned automatically to each word via supertagging (Clark and Curran, 2007) on the input sequence. We use the C&C supertagger[1] to assign a set of probable lexical categories to each input word using the gold-standard order. When the input is noisy, the accuracy of a supertagger tends to be lower than when the input is grammatical. One way to address this problem is to allow the supertagger to produce a larger list of possible supertags for each input word, and leave the ambiguity to the grammatical improvement system. We simulate the noisy input situation by using

[1]http://svn.ask.it.usyd.edu.au/trac/candc/wiki/Download.

	Precision
dictionary	58.5%
$\beta = 0.0001$	59.7%
$\beta = 0.075\%$	77.0%

Table 3: Lexical category accuracies. Timeout = 5s. 1 training iteration.

Length	dictionary	$\beta = 0.0001$	$\beta = 0.075$
≤ 5	75.65	89.42	92.64
≤ 10	57.74	66.00	78.54
≤ 20	42.44	48.89	58.23
≤ 40	37.48	40.32	46.00
≤ 80	36.50	39.01	44.26
all	36.27	38.75	43.99

Table 4: BLEU scores measured on different lengths on development data. Timeout = 5s. 1 training iteration.

a small probability cutoff (β) value in the supertagger, and supertag correctly ordered input sentences before breaking them into bags of words. With a β value of 0.0001, there are 5.4 lexical categories for each input word in the development test (which is smaller than the dictionary case).

The average number of lexical categories per word drops to 1.3 when β equals 0.075, which is the value used for parsing newspaper text in Clark and Curran (2007). We include this β in our experiments to compare the effect of different β levels.

The table shows that the BLEU score of the grammaticality improvement system is higher when a super tagger is used, and the higher the β value, the better the BLEU score. In practice, the β value should be set in accordance with the lack of grammaticality and fluency in the input. The dictionary method can be used when the output is extremely unreliable, while a small beta value can be used if the output is almost fluent.

Due to the default output mechanism on timeout, the system can sometimes fail to produce sentences that cover all input words. We choose five different timeout settings between 0.5s to 50s, and compare the speed/quality tradeoff. In addition to BLEU, we report the percentage of timeouts and the ratio of the sum of all output sentence lengths to the sum of all input sentence lengths.

When the timeout value increases, the BLEU score generally increases. The main effect of a larger timeout is the increased possibility of a complete sentence being found. As the time increases from 0.5s to 50s using the dictionary method, for example, the average output sentence length increases from 84% of the input length to 93%.

Table 3 shows the lexical category accuracies using the dictionary, and supertagger with different β levels. The timeout limit is set to 5 seconds. As the lexical category ambiguity decreases, the accuracy increases. The best lexical category accuracy of 77% is achieved when using a supertagger with a β level 0.075, the level for which the least lexical category disambiguation is required. However, compared to the 93% lexical category accuracy of a CCG parser (Clark and Curran, 2007), which also uses a β level of 0.075 for the majority of sentences, the accuracy of our grammaticality improvement system is much lower. The lower score reflects the lower quality of the parse trees produced by our system. Besides the difference in the algorithms themselves, one important reason is the much higher complexity of our search problem.

Table 4 shows the BLEU scores measured by different sizes of input. We also give some example output sentences in Figure 3. It can be seen from the table that the BLEU scores are higher when the size of input is smaller. For sentences shorter than 20 words, our system generally produces reasonably fluent and grammatical outputs. For longer sentences, the grammaticality drops. There are three possible reasons. First, larger constituents require more steps to construct. The model and search algorithm face many more ambiguities, and error propagation is more severe. Second, the search algorithm often fails to find a goal hypothesis before timeout, and a default output that is less grammatical than a complete constituent is constructed. Long sentences have comparatively more input words uncovered in the output. Third, the upper bound is not 100, and presumably lower for longer sentences, because there are many ways to generate a grammatical sentence given a bag of words. For example, the bag $\{$ cats, chase, dogs $\}$ can produce two equally fluent and grammatical sentences.

The relatively low score for long sentences is un-

(dictionary)	`our products There is no asbestos in now .`
($\beta = 0.0001$)	`in our products now There is no asbestos .`
($\beta = 0.075$)	`There is no asbestos in our products now .`
(dictionary)	`No price for the new shares has been set .`
(both β)	`No price has been set for the new shares .`
(all)	`Federal Data Corp. got a $ 29.4 million Air Force contract for intelligence data handling .`
(dictionary)	`was a nonexecutive director of Rudolph Agnew and former chairman of Consolidated Gold Fields PLC , this British industrial conglomerate , 55 years old . named`
($\beta = 0.0001$)	`old Consolidated Gold Fields PLC , was named 55 years , former chairman of Rudolph Agnew and a nonexecutive director of this British industrial conglomerate .`
($\beta = 0.075$)	`Consolidated Gold Fields PLC , 55 years old , was named former chairman of Rudolph Agnew and a nonexecutive director of this British industrial conglomerate .`
(dictionary)	`McDermott International Inc. said its Babcock & Wilcox unit completed the sale of its Bailey Controls Operations for Finmeccanica S.p . A. to $ 295 million .`
($\beta = 0.0001$)	`$ 295 million McDermott International Inc. for the sale of its Babcock & Wilcox unit said its Bailey Controls Operations completed to Finmeccanica S.p . A. .`
($\beta = 0.075$)	`McDermott International Inc. said its Bailey Controls Operations completed the sale of Finmeccanica S.p . A. for its Babcock & Wilcox unit to $ 295 million .`

Figure 3: Example outputs on development data.

likely to be such a problem in practice, because the base system (e.g. an SMT system) is likely to produce sentences with locally fluent subsequences. When fluent local phrases in the input are treated as atomic units, the effective sentence length is shorter.

All the above development experiments were performed using only one training iteration. Figure 4 shows the effect of different numbers of training iterations. For the final test, based on the graphs in Figure 4, we chose the training iterations to be 8, 6 and 4 for the dictionary, $\beta = 0.0001$ and $\beta = 0.075$ methods, respectively.

6.2 Final Accuracies

Table 5 shows the final results of our system, together with the MST-based ("Wan 2009 CLE") and assignment-based ("Wan 2009 AB") systems of Wan et al. (2009). Our system outperforms the dependency grammar-based systems, and using a supertagger with small β value produces the best BLEU. Note that through the use of a supertagger, we are no longer assuming that the input is a bag of words without any order, and therefore only the dictionary results are directly comparable with Wan et al. (2009)[2].

	BLEU
Wan 2009 CLE	26.8
Wan 2009 AB	33.7
This paper dictionary	40.1
This paper $\beta = 0.0001$	43.2
This paper $\beta = 0.075$	50.1

Table 5: Final accuracies.

[2]We also follow Wan et al. (2009) by assuming each word is associated with its POS tag.

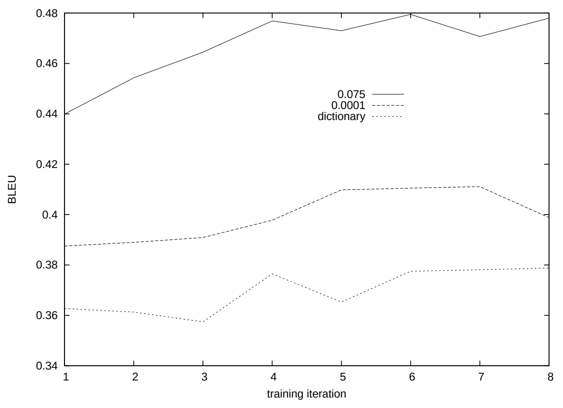

Figure 4: The effect of training iterations.

7 Related Work

Both Wan et al. (2009) and our system use approximate search to solve the problem of input word ordering. There are three differences. First, Wan et al. use a dependency grammar to model grammaticality, while we use CCG. Compared to dependency trees, CCG has stronger category constraints on the parse structure. Moreover, CCG allows us to reduce the ambiguity level of the search algorithm through the assignment of possible lexical categories to input words, which is useful when the input has a basic degree of fluency, as is often the case in a grammaticality improvement task.

Second, we use learning to optimise search in order to explore a large search space. In contrast, Wan et al. break the search problem into a sequence of sub tasks and use greedy search to connect them. Finally, in addition to ordering, our algorithm further allows word selection. This gives our system the flexibility to support word insertion and deletion.

White (2004) describes a system that performs CCG realization using best-first search. The search process of our algorithm is similar to his work. The problem we solve is different from realization, which takes an input in logical form and produces a corresponding sentence. Without constraints, the word order ambiguities can be much larger with a bag of words, and we use learning to guide our search algorithm. Espinosa et al. (2008) apply hypertagging to logical forms to assign lexical categories for realization. White and Rajkumar (2009) further use perceptron reranking on N-best outputs to improve the quality.

The use of perceptron learning to improve search has been proposed in guided learning for easy-first search (Shen et al., 2007) and LaSO (Daumé III and Marcu, 2005). LaSO is a general framework for various search strategies. Our learning algorithm is similar to LaSO with best-first inference, but the parameter updates are different. In particular, LaSO updates parameters when all correct hypotheses are lost, but our algorithm makes an update as soon as the top item from the agenda is incorrect. Our algorithm updates the parameters using a stronger precondition, because of the large search space. Given an incorrect hypothesis, LaSO finds the corresponding gold hypothesis for perceptron update by constructing its correct sibling. In contrast, our algorithm takes the lowest scored gold hypothesis currently in the agenda to avoid updating parameters for hypotheses that may have not been constructed.

Our parameter update strategy is closer to the guided learning mechanism for the easy-first algorithm of Shen et al. (2007), which maintains a queue of hypotheses during search, and performs learning to ensure that the highest scored hypothesis in the queue is correct. However, in easy-first search, hypotheses from the queue are ranked by the score of their next action, rather than the hypothesis score. Moreover, Shen et al. use aggressive learning and regenerate the queue after each update, but we perform non-agressive learning, which is faster and is more feasible for our complex search space. Similar methods to Shen et al. (2007) have also been used in Shen and Joshi (2008) and Goldberg and Elhadad (2010).

8 Conclusion

We proposed a grammaticality improvement system using CCG, and evaluated it using a standard input word ordering task. Our system gave higher BLEU scores than the dependency-based system of Wan et al. (2009). We showed that the complex search problem can be solved effectively using guided learning for best-first search.

Potential improvements to our system can be made in several areas. First, a large scale language model can be incorporated into our model in the search algorithm, or through reranking. Second, a heuristic future cost (e.g. Varges and Mellish (2010)) can be considered for each hypothesis so that it also considers the words that have not been used, leading to better search. Future work also includes integration with an SMT system, where content word selection will be applicable.

Acknowledgements

We thank Graeme Blackwood, Bill Byrne, Adrià de Gispert, Stephen Wan and the anonymous reviewers for their discussions and suggestions. Yue Zhang and Stephen Clark are supported by the European Union Seventh Framework Programme (FP7-ICT-2009-4) under grant agreement no. 247762.

References

Srinivas Bangalore, Owen Rambow, and Steve Whittaker. 2000. Evaluation metrics for generation. In *Proceedings of the First International Natural Language Generation Conference (INLG2000), Mitzpe*, pages 1–8.

Graeme Blackwood, Adrià de Gispert, and William Byrne. 2010. Fluency constraints for minimum bayes-risk decoding of statistical machine translation lattices. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 71–79, Beijing, China, August. Coling 2010 Organizing Committee.

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Comput. Linguist.*, 24:275–298, June.

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

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *ICML*, pages 169–176.

Dominic Espinosa, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proceedings of ACL-08: HLT*, pages 183–191, Columbus, Ohio, June. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June. 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. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Meeting of the ACL*, pages 359–366, Sapporo, Japan.

Kevin Knight. 2007. Automatic language translation generation help needs badly. In *MT Summit XI Workshop on Using Corpora for NLG: Keynote Address*.

Phillip Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. III Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 271–278, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Libin Shen and Aravind Joshi. 2008. LTAG dependency parsing with bidirectional incremental construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii, October. Association for Computational Linguistics.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*, pages 760–767, Prague, Czech Republic, June.

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

Sebastian Varges and Chris Mellish. 2010. Instance-based natural language generation. *Natural Language Engineering*, 16(3):309–346.

Stephen Wan, Mark Dras, Robert Dale, and Cécile Paris. 2009. Improving grammaticality in statistical sentence generation: Introducing a dependency spanning tree algorithm with an argument satisfaction model.

In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 852–860, Athens, Greece, March. Association for Computational Linguistics.

Michael White and Rajakrishnan Rajkumar. 2009. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Singapore, August. Association for Computational Linguistics.

Michael White. 2004. Reining in CCG chart realization. In *Proc. INLG-04*, pages 182–191.