Lexicalized Semi-Incremental Dependency Parsing

Hany Hassan, Khalil Sima'an and Andy Way

Abstract

Even leaving aside concerns of cognitive plausibility, incremental parsing is appealing for applications such as speech recognition and machine translation because it could allow for incorporating syntactic features into the decoding process without blowing up the search space. Yet, incremental parsing is often associated with greedy parsing decisions and intolerable loss of accuracy. Would the use of lexicalized grammars provide a new perspective on incremental parsing?

In this paper we explore incremental left-to-right dependency parsing using a lexicalized grammatical formalism that works with lexical categories (supertags) and a small set of combinatory operators. A strictly incremental parser would conduct only a single pass over the input, use no lookahead and make only local decisions at every word. We show that such a parser suffers heavy loss of accuracy. Instead, we explore the utility of a two-pass approach that incrementally builds a dependency structure by first assigning a supertag to every input word and then selecting an incremental operator that allows assembling every supertag with the dependency structure built so-far to its left. We instantiate this idea in different models that allow a trade-off between aspects of full incrementality and performance, and explore the differences between these models empirically. Our exploration shows that a semi-incremental (two-pass), linear-time parser that employs fixed and limited look-ahead exhibits an appealing balance between the efficiency advantages of incrementality and the achieved accuracy. Surprisingly, taking local or global decisions matters very little for the accuracy of this linear-time parser. Such a parser fits seemlessly with the currently dominant finite-state decoders for machine translation.

1 Introduction

As it processes an input sentence word-by-word in some order (e.g., left-to-right for a language like English), an *incremental* parser builds for each prefix of the input sentence a partial parse that is a subgraph of the partial parse that it builds for a longer prefix. An incremental parser may have access only to a fixed, limited window of lookahead words. The lookahead is equivalent to buffering a number of words before processing them; as stated by [Marcus et. al., 1983], a deterministic parser can buffer and examine a small number of words before adding them to the existing structure. A parser might also make multiple incremental passes over the input sentence where decisions made in an earlier pass are refined in a next pass. In this paper we refer to an incremental, left-to-right parser without lookahead information taking a single pass over the input a *fully incremental parser*. We refer to an incremental left-to-right parser with limited lookahead capability as *weakly* incremental parser. And when the incremental parser makes a two-passes over the input we refer to it with the term *semi-incremental*.

Besides being cognitively plausible, an incremental parser is more appealing for applications if its time and space (worst-case) complexities are close to linear in input length. For example, an incremental, linear-time parser should constitute a natural match for the word-by-word decoding and pruning schemes used within phrase-based statistical machine translation (PB-SMT) and speech recognition. It is worth noting that fully incremental parsers are cognitively plausible [Marslen-Wilson, 1973, Sturt & Lombardo, 2004], while weakly incremental parsers can serve well for syntax-based language modeling where a local context of the word is usually provided for scoring.

The degree of possible incrementality in parsing depends largely on the syntactic formalism underpinning the parsing process. In this work, we adopt Combinatory Categorial Grammar (CCG) [Steedman, 2000], which seems to represent an appealing grammatical representation for incremental parsing due to two main reasons. Firstly, as highlighted in [Steedman, 2000], CCG can represent every leftmost string as a constituent even if it is not a syntactic constituent. This enables any left branching (left-to-right) parser to work fully incrementally. Secondly, a fully incremental dependency parser is only possible if the leftmost graph is fully connected at each parse state, which has been highlighted in [Nivre, 2004]. This is especially possible with grammars like CCG where the type raising and compositional capabilities can be utilized to keep the graph connected even when not resolving a dependency relation.

In this paper we present a new approach to linear-time, semi-incremental CCG parsing and explore the trade-off between the degree of incrementality and parse accuracy. On the one hand, it turns out not so easy to obtain a fully incremental parser that maintains the same accuracy level as a parser that explores the full parse-forest without worries about incrementality or linear-time processing. On the other hand, we show that reasonable levels of accuracy can be achieved when the requirements of incrementality are somewhat relaxed: allowing a two pass classification approach (a supertagger followd by operators tagger) and a limited use of look-ahead. Furthermore, when allowing for global search (as opposed to local classification) during both supertagging and operator-tagging (thereby sacrificing a major aspect of incrementality), small accuracy improvements can be obtained. The best performing models would fit seemlessly with linear-time, finite state decoders used in machine translation and speech recognition because they predict a single partial parse at every word position in the input.

The structure of this paper is as follows. In section 2, we discuss related work on incremental parsing, and introduce our model of incremental CCG parsing in section 3. We then detail our method for transforming the canonical derivations in the CCGbank into left-to-right, incremental derivations (section 4). Given this incremental version of the CCGbank, we study in section 5 a range of linear-time parsing models, with varying degrees of incrementality, and provide the results of a number of experiments in section 6. We discuss our findings, and offer some conclusions and avenues for further research in section 7.

2 Related Work

While there exist numerous recent efforts at (deterministic) incremental (shallow) dependency parsing (e.g. [Nivre, 2004, Shen & Joshi, 2005]), most current constituency parsers are hard to classify as incremental. State-of-the-art parsers (e.g. [Collins, 1999, Charniak, 2000]) select the Viterbi parse only when the parse-space has been spanned for the whole sentence; the Viterbi parse may change as the prefix grows, violating incrementality. In contrast, partial parsers such as [Abney, 1991] by default do not output a full sequence of connected phrases, which causes the constraint of incrementality to fail for a quite different reason.

Among the many attempts at efficient (often deterministic) and incremental (dependency) parsing, we concentrate on the quite small number of research papers which are most similar to our work [Yamada & Matsumoto, 2003, Nivre, 2004, Shen & Joshi, 2005, Sagae & Lavie, 2006]. [Nivre, 2004] suggests that deterministic dependency parsing (e.g. [Yamada & Matsumoto, 2003]) is a halfway house between full and partial parsing, in that the building of a full parse of the input string is the aim, whilst at the same time remaining robust, efficient and deterministic. [Nivre, 2004] then describes his own approach to incremental deterministic dependency parsing. While strict incrementality was not possible using his framework, as far as well-formed utterances are concerned, the degree of incrementality achievable approaches 90%. In a similar manner, [Sagae & Lavie, 2006] introduce a statistical shift-reduce parser that uses a probabilistic framework to determine the shift/reduce actions. Multiple possible parse decisions are handled by a beam strategy.

[Shen & Joshi, 2005] use the term ‘semi-incremental’ to refer to parsers (both left-corner [Collins & Roark, 2004] and head-corner [Yamada & Matsumoto, 2003]) which permit multiple iterations of left-to-right scans, rather than just one. In contrast to these models, [Shen & Joshi, 2005] introduce an approach for incremental parsing of spinal Lexicalized Tree Adjoining Grammar, which supports full adjunction, a dynamic treatment of coordination, as well as non-projective dependencies. This can be see as an incremental version of Supertagging [Bangalore & Joshi, 1999].

Incremental dependency parsing of Categorial Grammar was attempted in [Milward, 1995] using a state-transition (or dynamic) processing model, where each state consists of a syntactic type together with an associated semantic value. In [Milward, 1994, Tugwell, 1998], a generic approach for incremental parsing is proposed based on an infinite state Markovian representation. The model of incremental parsing for CCG that we propose here is largely inspired by ideas presented in the latter three papers.

3 Semi-Incremental CCG Parsing

As it processes the sentence left-to-right, word-by-word, our parser specifies for every word a supertag and a combinatory operator, and maintains a parse-state (henceforth ‘state’). Each state is represented by a composite CCG category. Apart from the first word in the sentence, this composite CCG category is the result of applying the combinatory operator to the preceding state and current supertag; at the first word in the sentence, the state consists solely of the supertag of that word. In terms of CCG representations, a CCG composite category specifies a functor and the arguments that are expected to the right of the current word. Crucially, given a sentence and its state sequence, the dependency structure can be retrieved unambiguously. At each state the partial dependency structure can be represented as a directed graph with nodes representing words and arcs representing dependency relations. Figure 1 illustrates the workings of this incremental parser, where the sequence of supertags is shown, then the operators that apply from left to right in order to build the state sequence incrementaly.

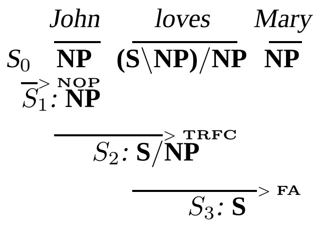


Fig. 1: *A sentence and possible supertag-, operator- and state-sequences. NOP: No Operation; BC: Backward Composition; FA: Forward Application.*

From the description above it is conceptually appealing to describe our parser in two parts: (i) a Supertag-Operator Tagger which proposes a supertag–operator pair for the current word, and (ii) a State-Realizer, which realizes the current state by applying the current operator to the previous state and the current supertag. In this work, the State-Realizer is a deterministic function, whereas the supertag-operator tagger is a statistical one trained on our own incremental version of the CCGbank. While this conceptual view describes a baseline, fully incremental version, in what follows we will consider refinements that trade off some aspects of incrementality for accuracy (i) by employing lookahead in predicting supertag–operator pairs, (ii) by predicting the supertag and using that for predicting the operator (a cascade of per-word classifiers possibly with lookahead), and (iii) by using a two-pass, semi-incremental approach where supertags and operators are chosen globally using Viterbi decoding (rather than per-word by means of classifiers). All these versions remain linear-time and space (worst-case complexity).

To train the statistical components, we transform the CCGbank normal form derivations into strictly left-to-right derivations, with operators specifically chosen to allow incrementality while satisfying the dependencies in the CCGbank. In the next section we present our transformation technique developed to obtain the appropriate training data.

4 Transforming the CCGbank into left-to-right derivations

The goal of the transformation is to obtain training data for our incremental parsing approach. The result of the transform is an incremental CCGbank where sentences are annotated with supertags as well as combinatory operators that allow left-to-right, incremental building of a parse while satisfying the dependencies specified in the CCGbank.

For each sentence in the CCGbank, we apply the following procedure:

- Initialize empty operator sequence and empty unsatisfied dependencies;
- For each word:
  1. Add current dependencies to unsatisfied dependencies.
  2. Check unsatisfied dependencies:
     (a) If adjacent dependency with simple categories, assign *application* operators;
     (b) If adjacent dependency with complex categories, assign *composition* operators;
     (c) If long-range dependency, apply *Type Raising* followed by *Forward Composition*.
  3. Handle special cases if any:
     (a) Coordination (section 4.1),
     (b) Apposition & Interruptions (section 4.2),
     (c) WH-movement (section 4.3),
  4. Update Current state;
  5. Assign selected operator to the operator sequence;
  6. Update the dependencies by removing satisfied dependencies.

This procedure deploys the dependencies available in the CCGbank in order to assign the simplest possible operator sequence that is able to satisfy, and reproduce, the dependency structure of the sentence under investigation.

Figure 2 illustrates the transformation process, step-by-step, on a sentence of the CCGbank. At the beginning of the process, we start with the words, the associated supertags and the dependency relations, indicated by curved dotted arrows in the figure. The purpose of the transformation process is to induce the state sequence and the operator sequence. These sequences should be able to reproduce the given dependency relations.

The transformation process proceeds word-by-word, and at each word position we check all previous and current unsatisfied dependencies. The transformation proceeds as follows:

1. State *S1* being an initial state, it is associated with operator *NOP*.

2. Moving to State *S2*, there is a dependency between the previous word *Mr.* and the current word *Warren*. As this dependency relation is adjacent and in the forward direction, the *FA* operator is associated and so the state is transferred to *S2* with category *NP*.

3. Moving to State *S3* is triggered by the word *will*, which has both backward and forward dependencies. Thus the operator *BC* is applied.

4. Moving to State *S4* is triggered by the word *remain*, which being linked with the word *will* by a forward dependency relation, causes the *FC* operator to be assigned. The state becomes *(S/PP)*, indicating that a prepositional phrase is required to the right.

5. Moving to State *S5* is triggered by the word *on* which is linked to the previous verb *remain*, and so the *FC* operator is assigned.

6. Moving to State *S6* is triggered by the word *the*, which has neither backward nor forward dependencies; however, it is linked through a chain of dependencies with a future word *board*. Thus we apply the composite *TRFC* operator to type raise the current word to the required dependency category and then perform forward composition.

7. Moving to State *S7* is triggered by the word *company*, which has a forward dependency with the previous position, and so the *FA* operator is applied.

8. Moving to State *S8* is triggered by the word *'s* which has adjacent forward and backward dependencies, so the *FC* operator is applied. This changes the state to *(S/NP)*.

9. Finally, state *S9* is triggered by the word *board*. The *FA* operator is finally applied to construct the complete category *S*.

This detailed illustration shows how the CCGbank is transformed. We started with a supertag sequence and a dependency graph, and ended with the corresponding operator and state sequences. However, the same procedure applies during parsing, i.e. if we have the supertag and the operator sequences, then we are able to construct both the incremental states and the dependency graph. As an indication of the performance of our transformation process, on section 00 of the WSJ—our dev data (see section 6)—we managed to successfully extract 97% of the CCGbank dependencies, with the incremental approach failing to restore just 3% of the dependencies.

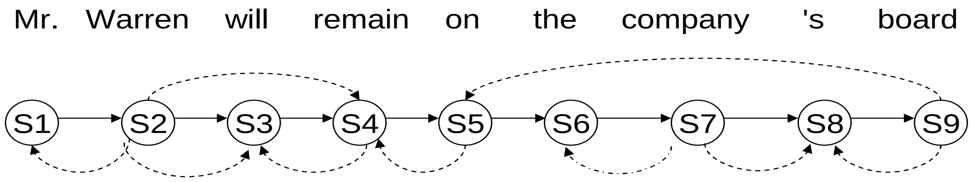
In the next three sections we describe the special operators we added for cases of coordination, apposition, interruptions and WH-movement.

4.1 Coordination

Cognitive studies [Sturt & Lombardo, 2004] view coordination as an incremental operation. Unfortunately, the CCGbank uses a simple category *conj* for coordination instead of the more elaborate category *(X\X)/X*, which was originally defined for coordination in CCG [Steedman, 2000]. The atomic *conj* operator is not efficient for incremental parsing, because it does not provide information on the coordinated elements. Therefore, we use the dependency information to assign more elaborate coordination categories. For example, if coordination is performed between two noun phrases, the appropriate category is *(NP\NP)/NP*. Furthermore, we added a new coordination operator (*COORD*) to handle coordination in the state-realizer. When such a conjunction is encountered, the nearest previous state that is able to satisfy the coordination is retrieved from the state stack to become the current state.[1]

The example shown in Figure 3 illustrates the handling of coordination. At the transition from *S3* to *S4* a coordination operator *COORD* is encountered, which causes the current state *S4* to 'go back' to state *S2* with structure *S/NP*, i.e. expecting an *NP* to the right.

[1] This retrieval action of a previous state remains incremental because the previous state sequence is not altered, and the retrieved state is always an extension of the last state (it always adds arguments to a given category rather than changing it to a different one).



Supertag	NP/NP	NP	(S\NP)/ (S\NP)	(S\NP) /PP	PP/NP	NP/NP	NP	(NP/NP) \NP	NP
Operator	NOP	FA	TRFC	FC	FC	TRFC	FA	FC	FA
State Cat.	NP/NP	NP	S/(S\NP)	(S/PP)	(S/NP)	(S/(NP\NP)) /NP	(S/(NP \NP))	(S/NP)	S

Fig. 2: *Illustration of the CCGbank transformation process into incremental derivations.*

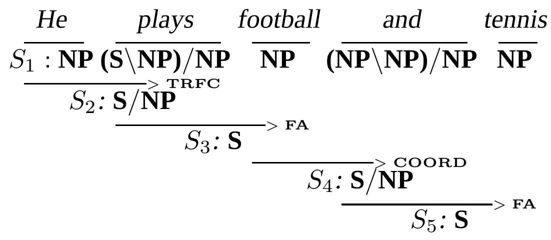


Fig. 3: *Coordination Handling.*

4.2 Apposition and Interruptions

Neither the CCGbank nor the WSJ treebank distinguish between the appositive and coordination commas [Hockenmaier & Steedman, 2007]. The comma mostly has a single supertag in the CCGbank that does not indicate its actual role in the syntactic structure of the sentence at hand. We adopted the syntactic patterns in [Bayraktar et al., 1998] as a means of identifying the comma's different possible syntactic categories. Based on these syntactic patterns, we enriched the supertags associated with the comma to indicate the correct syntactic role. Furthermore, we added a new operator INTR for handling cases of both apposition and interruptions.

4.3 WH-movement

A new operator is added to handle cases of WH-movement where a syntactic category is required on the right but, having moved, is available only on the left. Consider the sentence *He believes in what he plays.* Here *plays* is of category (S\NP)/NP, i.e. it is a transitive verb, where if it finds an object NP to its right, and a subject NP to its left, a sentence will have been formed. The NP *what* is expected as the object somewhere to the right of *plays*, but has already moved to an appropriate position somewhere to its left. Accordingly, we added a new operator WHMV to handle such cases in our framework.

5 Implementation Detail

After POS tagging, the parser works its way through the sentence, left-to-right, assigning for every word a supertag–operator pair, and deciding on a state using the deterministic state-realizer. We describe the state-realizer

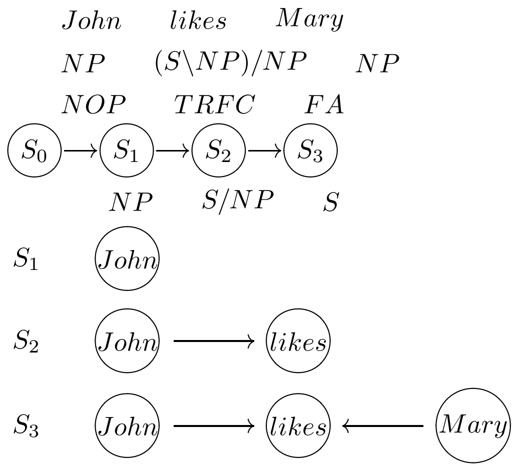


Fig. 4: *Illustration of the operation of the incremental parse-state realizer and the associated intermediate dependency graphs at each state.*

before delving into the implementation of different versions of the supertag-operator tagger.

Parse-State Realizer After assigning supertag–operator pairs for the words of the input sentence (described in the next section), the *state-realizer* deterministically realizes the parse-states as well as the intermediate dependency graphs between words using the CCG incremental operators (as defined in our incremental version of the CCGbank). Starting at the first word with (obviously) a null previous state, the realizer performs the following steps for each word in turn: (i) set the current supertag and operator to those of current word; (ii) at the current state apply the current operator to the previous state and current supertag; (iii) add edges to the dependency graphs between words that are linked as CCG arguments; and (iv) if not at end of sentence, set previous state to current one, then set current word to next one, and iterate from (i).

Figure 4 illustrates the realizer operation along with the incrementally constructed partial dependency graphs at each state. At the initial state S_0, the *NOP* Operator is applied to the previous state, a Null state, and the current supertag *NP*; the resulting state is S_1 with category *NP* and the resulting dependency graph is simply the node representing the first word *John*. The transition to the next state S_2 is triggered by the verb *likes*, where the *TRFC* operator is applied to the previous state and the current supertag, resulting in a new state *S/NP*. The dependency graph associated with state S_2 shows the realized dependency between *likes* and *John* which has resulted from the previous operation. Finally the last word triggers the final state, and the realizer is able to construct the full dependency graph which is associated with the last state S_3.

Maximum Entropy Taggers We build different linear-time models for assigning supertag–operator pairs to words in order to explore the effect of the different gradations of incrementality on parsing accuracy. All models present in this paper are based on MaxEnt classifiers [Berger et al., 1996]. A MaxEnt classifier selects the class that gives the highest conditional probability of any class given a set of features of the input, where the probability is expressed as a log-linear interpolation of weights of features. The weights are trained in order to maximize the likelihood of the given training data. In this work we use sequential conditional generalized iterative scaling [Goodman, 2002].

In building classifiers for sequence-tagging, as in the present case, choices arise regarding the search (or decoding) strategy as well as regarding the use of lookahead features:

- *Search (GLOBAL/LOCAL):* A MaxEnt model can be used for sequence classification, either as a classifier (LOCAL) or by converting the classification scores into probabilities and then using standard dynamic programming (Viterbi search, GLOBAL). Naturally, per-word (left-to-right) LOCAL classifiers facilitate a more incremental parser than GLOBAL classifiers; in the latter case, the state-realizer has to wait till the end of the sentence before it can realize the states.
- *Lookahead: (Y-LH/N-LH):* For training the classifiers, lookahead features could either be included (Y-LH) or not (N-LH). When lookahead is used, it refers to the lexical and POS features from a window of two words two words to the right of the focus word. Note that those features to the left are always included in all models as history.

Crucially, the choice for a global search procedure is orthogonal to using a lookahead. This is because in a global search the disambiguation weights are the accumulation (assuming independence given the overlapping features) of the weights of local classification decisions (per word) for the whole input sentence, whereas the lookahead affects the features used to estimate the local weights themselves. Hence, when assuming independence (multiplying) between the local decisions, global search cannot substitute for the use of lookahead.

Furthermore, given any choice of search strategy and lookahead, there are two different architectures for assigning supertag–operator pairs to words:

- *Joint:* We train a MaxEnt module to assign supertag–operator pairs (as a single class) to words.
- *Cascaded:* We train two MaxEnt modules, one for supertags, and one for operators, and use them in a cascade. We limit the options here such that both classifiers (supertag and operator) either use lookahead or not, i.e. we do not explore a mixed situation where one does and the other does not. For the version that does use lookahead, while the supertagger is trained using the standard feature set, the operator tagger is trained with the same window but with POS and supertag features coming from the supertagger (no lexical features).

As a baseline we start out from the Joint classifier, without lookahead (N-LH) and with LOCAL search. We choose this as a baseline as it is the most straightforward implementation of an incremental, linear-time CCG parser without making any extra assumptions. The Cascaded architecture with LOCAL search and without lookahead (N-LH) might be seen as incremental but it embodies an extra set of independence assumptions (i.e. different choices of features for supertag and operator classifiers). Similarly, any model using look-ahead and LOCAL search is slightly less incremental since it employs still more resources than the preceding models. If we move one further step away from full to semi-incremental parsing, we arrive at models which employ GLOBAL search (since the supertag–operator sequences are assigned only after the whole sentence has been processed).

6 Experiments and Results

This section details a number of experiments carried out to test the effectiveness of the supertagger, the operator tagger, and our ability to capture the necessary dependencies using a range of incremental parsers. We used the same data split as in [Clark & Curran, 2007]. Sections 02–21 were used for training, section 00 for dev-testing of intermediate taggers, and section 23 for testing dependencies.

6.1 Supertagging Results

Given our introduction of new supertags for coordination, apposition, interruption, and WH-movement, we used section 00 to evaluate our supertagger's accuracy compared to the standard CCGbank set. Although our supertags are more complex, we obtain an F-score of 91.7 (cf. Table 1, last row, 'Supertagging' column), which compares favourably with the supertagger of [Clark & Curran, 2007], which scores 92.39 on the same dataset. While we have not carried out significance testing at this stage, it is clear that there is little difference between the two sets of scores, indicating that our supertagger is robust as well as accurate. As will be seen for all experiments in this section, this is only true when lookahead is utilised; note that our best score of 91.7 dips to 68.62—an absolute drop of 23.08 points, or a 33.6% relative decrease in performance—when lookahead is turned off.

6.2 Operator Tagging Results

In Table 1 we also present the results for our Operator tagger. This displays a very high accuracy (90.9%, cf. last row, 'Operator Tagging' column) even when no lexical features are used. We also contemplated a hypothetical situation in which we feed the correct (gold standard) previous syntactic state as a feature to the system. In this scenario an operator tagging score of 99.22% (8.32% absolute improvement, or 9.15% relative) was obtained, indicating that a high gain is to be expected if this state were to be made available to the operators classifier.

6.3 Dependency Results

In Table 1 we also present the results for unlabeled dependency accuracy using our method. We use the same

Architecture	Lookahead	Search	Dependency Accuracy	Supertagging Accuracy	Operator Tagging Accuracy	Incremental
Joint	NO	LOCAL	56.02	67.47		Incr.
	NO	GLOBAL	56.13	68.31		Semi
	YES	LOCAL	82.17	84.34		Incr.+LH
	YES	GLOBAL	83.20	85.02		Semi+LH
Cascaded	NO	LOCAL	59.01	68.11	76.19	Incr.
	NO	GLOBAL	59.30	68.62	76.53	Semi
	YES	LOCAL	86.31	91.62	90.76	Incr.+LH
	YES	GLOBAL	86.70	91.70	90.90	Semi+LH

Table 1: *All results (F-Score) of systems applied to input with POS tags output by POS tagger trained using Maxent with feature window (+/-2) on the CCGbank POS data. The results with no lookahead use only left window features.*

evaluation criteria as [Clark & Curran, 2007] by comparing the dependency output of the incremental parser with the predicate-argument dependencies in the CCGbank. Testing on section 23 of the WSJ, we obtain an F-score of 86.7 (last row, 'Dependency' column). The score with the gold standard POS and supertags in the input is 87.5, 0.8% absolute (or 0.92% relative) higher than the result when using the POS, supertags and operators hypothesized by the system, but not by much. While this result is considerably below the best known result for a non-incremental, bottom-up chart parser [Clark & Curran, 2007]), we think that our result is reasonably good for an extremely efficient, semi-incremental parser. To put the efficiency gains in perspective, the parsers of [Collins, 1999], [Charniak, 2000] and [Clark & Curran, 2007] take respectively 45, 28 and 1.9 mins to parse section 23 of the WSJ. By contrast, our parser takes only 11 seconds, a speed-up of around ten times relative to the fastest among these parsers [Clark & Curran, 2007] (parsing times are measured on a machine with the same specifications).

6.4 Cascaded vs. Joint Approach

The results reported above demonstrate the accuracy of the cascaded approach using two cascaded taggers: the first for supertags, and the second the operator tagger followed by the deterministic state- realizer. In this section we compare the cascaded model with a joint model, where we train a single classifier that produces the supertags and operators simultaneously in the same step. In Table 1 we give the unlabeled dependency results for section 23 for the cascaded and joint models side-by-side for comparative purposes. The cascaded model significantly outperforms the joint model (by 3.5% absolute, or 4.2% relative; this rises to 4.3% absolute, or 5.17% relative, if we compare the joint model with the dependency score using the gold standard POS and supertags, as described in the previous section). Besides data sparseness, the joint model makes the choice of an operator at a certain position in the sentence based on supertag information only to the left of the current position because the joint model must guess supertag–operator pairs at once.

Note that our Cascaded version with lookahead and GLOBAL search is the semi-incremental model of [Shen & Joshi, 2005]. They report an F-score of 89.3 on section 23 using a semi-incremental approach, together with extra information from Propbank [Palmer et al., 2005]. While not directly comparable, we consider our performance to be on a par with theirs, with a considerable improvement in parsing time (they report a speed of 0.37 sent./sec.).

6.5 Effect of Lookahead

The present parser is just two words of lookahead away from being fully incremental. Here, we examine the effect of lookahead features on the supertagger, operator tagger and dependency results. We examine two versions of a supertag- and operator-classifier, namely a weakly incremental and a fully incremental version. The weakly incremental version deploys features in a window of two words to the left and two words to the right of the focus word. The fully incremental parser deploys features in a window of two words to the left only.

Looking at all the results in Table 1, the scores for the weakly (semi-)incremental versions of the parser barely differ from their fully incremental counterparts, whether we are concerned with dependency, supertagging or operator accuracy; the scores are higher, on the whole, but not by much.

By contrast, what is *extremely* significant is the extent to which lookahead is utilised. For all accuracy measures,huge improvements are to be seen when the parser avails of lookahead. Clearly, full incrementality at this stage comes at a high cost in accuracy, relative to the weakly incremental version, without any benefit in efficiency.

7 Conclusions and Future work

In this paper we introduced a novel, simple approach for wide-coverage CCG analysis using a weakly incremental parser. In addition to the standard CCG operators, we added extensions to efficiently handle coordination, apposition, interruptions and WH-movement. Our supertagger achieves results comparable with the state-of-the-art for CCG [Clark & Curran, 2004]. Moreover, the operator tagger, even without lexical features, performs well on our extended operator set.

Our empirical results show three main findings. Firstly, despite being just two words of lookahead away from being fully incremental, our proposed cascaded weakly incremental parser outperforms both the joint and fully incremental approaches. Nevertheless, there is perhaps less of a performance difference between the joint and cascaded architectures on the one hand, and between global search or local classification on the other. The fact that the local search performs almost as well as global search for this model implies that at least the local search could be easily integrated within speech-recognition or machine translation decoders.

Secondly, with respect to dependency parsing, while our overall result is below the result reported in [Clark & Curran, 2007] using a non-incremental bottom-up parser, it is far more efficient being (weakly) incremental. This speedup is, we feel, particularly attractive for applications that incorporate in the decoder a word-prediction (or language) model, since this semi-incremental parser works in a fashion similar to such language models, i.e. the possible states are built on-the-fly from the training data, just like any other non-parametric method.

Thirdly, our results, using state-of-the-art classifier, highlight the fact that incremental parsing using CCG is attainable at some tolerable cost in accuracy. Perhaps most importantly, incremental parsing for CCG as suggested here gives acceptable levels of accuracy only when lookahead is used. Yet, since our approach to parsing is left-to-right and makes decisions at every word in the input, one might actually question the effectiveness of taking decisions at every word. It might turn out, for example, that when the decision points are selected carefully, the need for lookahead can be less crucial. This conjecture is left for the future.

As for future research, work on an improved, fully incremental version is ongoing as well as an investigation of the effect of incremental parsing in machine translation.

References

[Abney, 1991] Abney, S. 1991. Parsing By Chunks. In R. Berwick, S. Abney and C. Tenny (eds.) *Principle-Based Parsing*. Kluwer, Dordrecht, pp.257–278.

[Bangalore & Joshi, 1999] Bangalore, S. and A. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics* **25**(2):237–265.

[Bayraktar et al., 1998] Bayraktar, M., B. Say and V. Akman. 1998. An Analysis of English punctuation: The Special Case of Comma. *International Journal of Corpus Linguistics* **3**(1):33–58, .

[Berger et al., 1996] Berger, A., V. Della Pietra, and S. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* **22**(1):39–71.

[Charniak, 2000] Charniak, E. 2000. A Maximum-Entropy-Inspired Parser. In *1st Annual Meeting of the North American Association for Computational Linguistics, Proceedings (NAACL 2000)*, Seattle, WA., pp.132–139.

[Chiang, 2005] Chiang, D. 2005. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, MI., pp.263–270.

[Clark & Curran, 2004] Clark, S. and J. Curran. 2004. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING'04)*, Geneva, Switzerland, pp.282–288.

[Clark & Curran, 2007] Clark, S. and J. Curran. 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics* **33**(4):493–552.

[Collins, 1999] Collins, M. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. PhD Thesis, University of Pennsylvania, Philadelphia, PA.

[Collins & Roark, 2004] Collins, M. and B. Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, pp.111-118.

[Doran and Bangalore, 1994] Doran, C. and S. Bangalore. 1994. Bootstrapping a Wide-coverage CCG from FB-LTAG. In *Proceedings of Third International Workshop on Tree-Adjoining Grammars (TAG+3)*, Paris, France [no page numbers].

[Goodman, 2002] Goodman, J. 2002. Sequential Conditional Generalized Iterative Scaling. In *40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA., pp.9–16.

[Hassan et al., 2007] Hassan, H., K. Sima'an and A. Way. 2007. Integrating Supertags into Phrase-based Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, Prague, Czech Republic, pp.288–295.

[Hassan et al., 2008] Hassan, H., K. Sima'an and A. Way. 2008. Syntactically Lexicalized Phrase-Based SMT. *IEEE Transactions on Speech and Audio Processing* (in press).

[Hockenmaier & Steedman, 2007] Hockenmaier, J. and M. Steedman. 2007. CCG-bank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics* **33**(3):355–396.

[Marslen-Wilson, 1973] Marslen-Wilson, W. 1973. Linguistic structure and speech shadowing at very short latencies. *Nature* **244**:522–533.

[Milward, 1994] Milward, D. 1994. Dynamic Dependency Grammar. *Linguistics and Philosophy* **17**:561–605.

[Milward, 1995] Milward, D. 1995. Incremental Interpretation of Categorial Grammar. In *Proceedings of 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL-95)*, Dublin, Ireland, pp.119–126.

[Nivre, 2004] Nivre, J. 2004. Incrementality in Deterministic Dependency Parsing. In *Proceedings of the ACL Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, Barcelona, Spain, pp.50–57.

[Palmer et al., 2005] Palmer, M., D. Gildea and P. Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics* **31**(1):71–106.

[Sturt & Lombardo, 2004] Sturt, P. and V. Lombardo. 2004. Processing coordinated structures: Incrementality and connectedness. *Cognitive Science* **29**(2):291–305.

[Sagae & Lavie, 2006] Sagae, K. and A. Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics–Posters*, Sydney, Australia, pp.691–698.

[Shen & Joshi, 2005] Shen, L. and A. Joshi. 2005. Incremental LTAG Parsing. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, Vancouver, Canada, pp.811–818.

[Steedman, 2000] Steedman, M. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA.

[Tugwell, 1998] Tugwell, D. 1998. A Markov Model of Syntax. In *1st Annual CLUK Research Colloquium*, Sunderland, UK.

[Yamada & Matsumoto, 2003] Yamada, H. and Y. Matsumoto. 2003. Statistical dependency Analysis with Support Vector Machines. In *Proceedings of 8th International Workshop on Parsing Technologies (IWPT-2003)*, Nancy, France, pp.195–206.

[Marcus et. al., 1983] Mitchell Marcus and Donald Hindle and Margaret Fleck. 1983. D-theory: talking about talking about trees. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics (1983)*, Cambridge, Massachusetts, pp.129–136.