Mink: An Incremental Data-Driven Dependency Parser with Integrated Conversion to Semantics

Rachael Cantrell
Indiana University
Bloomington, Indiana
rcantrel@indiana.edu

Abstract

While there are several data-driven dependency parsers, there is still a gap with regards to incrementality. However, as shown in Brick and Scheutz [3], incremental processing is necessary in human-robot interaction. As is shown in Nivre et al. [12], dependency parsing is well-suited for mostly incremental processing. However, there is as of yet no dependency parser that combines syntax and semantics by including traditional dependency parsing, CCG tagging, and lambda-logical structures in one fast, accurate application suitable for embodied natural language processing.

This paper addresses that gap by introducing Mink, an incremental data-driven dependency parser with integrated conversion to semantics. We show that Mink is comparable to similar but non-incremental parsers, and that it succeeds at performing some semantic analysis.

Keywords

incremental parsing, dependency parsing, CCG, semantics conversion

1 Introduction

Dependency parsing has gained in popularity in the last few years. However, there is still a gap with regards to incrementality. As is shown in Nivre et al. [12], dependency parsing is well-suited for mostly incremental processing. However, there is as of yet no dependency parser that combines syntax and semantics by including traditional dependency parsing, CCG tagging, and lambda-logical structures in one fast, accurate application suitable for embodied natural language processing.

Our motivation for creating Mink was to meet the need for incremental processing in robots that must interact with humans using natural language. Most natural language processing (NLP) is performed on written texts. However, in embodied NLP systems such as robots, the system needs to understand incoming speech from humans. As discussed by Brick and Scheutz [3], humans do not wait for a full sentence in order to begin processing the sentence, parsing it, and resolving references. We immediately begin to process whatever we can as soon as we can, which allows us to look at referents, or reach for things before we know where we'll be told to put them, etc. Because humans have this capability, we become impatient when dealing with conversation partners that are slower. Mink takes us one step closer toward achieving a level of incremental processing in robots that will allow humans and robots to communicate easily and naturally.

Because we are working to help our robots "understand" verbal input, we are not interested in the parses themselves, but rather in what they can tell us semantically. Thus we have integrated semantic output into the system itself. Mink outputs three different types of information: dependency arcs, combinatorial categorial grammar (CCG) tags, and, when possible, lambda-logical semantic conversions.

In this paper, we evaluate Mink based on accuracy and speed. We evaluate the CCG tagging and semantic conversion capabilities together, by using them to attempt to correctly classify utterances into either sentence, question, or command. We chose these simple types because they are discourse types that are, in general, distinguishable from the syntax of the sentence.

In section 2, we discuss currently available dependency parsers, CCG taggers and parsers, and relevant work on semantics, particularly the interface between syntax and semantics. In section 3, we discuss the details of our parser. In section 4, we evaluate our parser. Finally, in section 5, we discuss how we are currently able to use the parser and where we plan to go with it.

2 Related Work

There are three main bodies of knowledge from which this project draws: dependency parsing, CCG tagging and parsing, and semantic representation of parser output.

2.1 Dependency Parsing

One often-used data-driven dependency parser is MaltParser [13], a shift-reduce dependency parser that uses support vector machines to deduce, from training examples, what next action to take in parsing a sentence. Nivre [12] discusses the potential for incrementality in MaltParser. It is determined that the algorithm is the best suited for incremental parsing that has been developed to date; however an incremental version of MaltParser has not been developed by those researchers.

MSTParser (c.f. McDonald et al. [10]) approaches the problem of finding dependency trees as one of finding maximum spanning trees. While this method is shown to be successful, the fact that it searches the entire space of possible dependency trees renders it inefficient for our purposes.

Johannsen and Nugues [8] combine aspects of MaltParser and MSTParser to form a dependency parser that, rather than maximizing the probability of each individual action (like MaltParser), maximizes the probability over a complete sentence. While this gives good results, it is a step away from incrementality.

2.2 CCG Tagging and Parsing

An utterance that is tagged with CCG tags contains more information about the parse tree of the utterance than one tagged with simple POS tags. This is because each tag contains information about both the POS (the return type), and about the token's children (the argument types). In an utterance with just one possible parse, this is enough information to describe the correct parse tree. For example: $the_{NP/NP}$ $child_{NP}$ $ate_{S\backslash NP/NP}$ $a_{NP/NP}$ $snack_{NP}$.

There is just one way to create a tree from this information, so, even if this utterance came from a tagger rather than a parser, it is essentially parsed already. However, many utterances have ambiguity, which a tagger is not intended to resolve. For our purposes, in most cases, a tagger gives us the information we need; therefore we consider both taggers and parsers as being of equal utility to us.

There are two basic types of CCG taggers/parsers, statistical and rule-based. An example of a small rule-based system is found in [1]. It looks up the CCG tag(s) of each word and builds a parse based on combinatory rules. Another rule-based system is OpenCCG, a freely-available CCG parser written in Java. While it has been widely used in many projects[1], the nature of a rule-based system limits its robustness, particularly for processing spoken data, which often contains unknown words and disfluencies.

There are several statistic systems, which we would prefer for greater robustness. For example, Clark [5] uses a maximum-entropy based statistical tagger to select CCG tags based on context.

The problem with statistical systems, for our purposes, is that they are based on probabilities of full tags, rather than on the parts.

Because we wanted to be able to build the tags by combining individual parts rather than selecting from a set of full CCG tags, we decided to use a dependency parser to build the tags based on dependency arcs.

2.3 Semantic Representation

Che et al. [4] integrates syntax and semantics. They parse the sentence using MSTParser, then use a series of classifiers to identify predicates, classify them according to senses, and assign semantic roles to different elements in the sentence. However, they do not integrate the results into any sort of logical framework.

Bos et al. [2] discusses a method of converting output from a wide-coverage CCG parser into semantic representation using lambda calculus, from which our semantic conversion method draws heavily.

Lambda conversion has been shown to be semantically useful in robotic NLP systems, c.f. Gold and Scassellati [6].

3 An incremental architecture for dependency parsing and integrated semantics conversion

3.1 Parsing Algorithm

We used the Nivre algorithm, which was previously implemented in MaltParser [11], which itself is an adaptation of the shift-reduce parsing algorithm for constituency parsers.

The parser keeps track of what has already been input in a stack. Each time a token is input, the parser must decide whether to shift, reduce, create a left arc, or create a right arc. These actions are described in Table 1.

The parser continues processing until there is no more input and the stack is empty again, resulting in a connected, non-projective dependency graph. Currently, the parser needs to know when there is a pause or other possible indicator of a sentence boundary; however, an action is being developed that allows the parser to guess that it should terminate the current graph and begin work on a new one. This, rather than separating actual sentences, is intended to separate semantically meaningful fragments, since spoken data often consists of such fragments, rather than of complete sentences.

The main difference between our implementation and others is its incrementality. While the other implementations accept only completed text files and output the same, Mink accepts input from a stream, which in our case is the output of a speech recognizer, and outputs partial analyses as soon as they are available. As soon as it pops a token off the stack–i.e. as soon as it is clear the token will have no more dependents–it outputs the CCG tag and begins semantic conversion. It outputs each conversion as it makes it so that the module that mediates between semantic representation and action can immediately begin to process the semantics of the input to see what it can do with what it knows so far.

3.2 Machine Learning Algorithm and Features

To decide which actions to perform, we used a maximum-entropy based classifier, namely, the Logistic algorithm from the Weka Java-based machine learning library [15]. This classifier decides, for a pair of an input token and a token on the top of the stack, which action to perform.

We trained the classifier on the dependency graph version of the Penn Treebank, created with pennconverter [7].

[1] `http://comp.ling.utexas.edu/wiki/doku.php/openccg/projects_using_openccg`

Table 1: *Parser Actions*

Action	Description
Shift	Move the input token to the stack
Reduce	Pop the top token off the stack
Left-arc	Create a dependency arc pointing from the input to the token on the top of the stack; this is followed by a reduce action
Right-arc	Create a dependency arc pointing from the token on top of the stack to the input token; this is followed by a shift action

For features, we tested several different sets. First, we used the standard set of features used by MaltParser, since they have proven to be ideal for parsing English. These features are: the token on top of the stack (TOP), its part-of-speech tag and dependency type, the dependency types of its leftmost and rightmost dependents, the input token (NEXT), and its part-of-speech tag and dependency type, and the part-of-speech of the next + 1 input. Next, we eliminated the lookahead feature. Last, we eliminated labels and used only unlabeled arcs. See Table 2 for a complete listing of feature sets.

We evaluate each of these feature sets in section 4.

3.3 Building CCG tags

CCG tags give a bit more information than part-of-speech tags. Namely, they give both the return type, and the arguments. The return type is basically the part-of-speech tag, while the arguments are similar to phrasal constituents. Our method of constructing CCG tags from dependency arcs therefore involves two steps: determining the return type, and finding the arguments.

When a new token is input, an "empty" CCG tag is created. This includes the return type and list of arguments. Ultimately, it can accept 0 or more arguments. Currently, the return type is determined immediately upon creating the new tag in order to reduce the complexity of the computation. This is posed as a classification problem. The features used to make the classification are currently the token and WSJ POS tag that was already found by the tagger.

Whenever a dependency arc is discovered with a given token as the head, the dependent's return type is added as an argument to the head's CCG tag.

As an example, the phrase `the blue box` might be tagged and parsed as follows by a constituent parser using the WSJ tagset:

$(NP(the_{DT}\ blue_{JJ}\ box_{NN}))$.

However, with a CCG parser, it would be tagged and parsed

$(NP\ (the_{NP/NP}\ (NP\ blue_{NP/NP}\ (NP\ box_{NP}))))$.

In the future, we would like to evaluate the utility of determining the return type after finding the token's arguments in order to use the arguments as features; however, that has not yet been implemented.

The CCG representation of a parse tree is then used in the semantic conversion.

3.4 Semantic Conversions

The conversion to semantics relies on a semantic dictionary to translate from the parse trees to semantic representations. Each entry in the dictionary consists of a word, its possible CCG tags, and the lambda-logical expressions for each word/CCG tag combination.

The goal of the conversion is, in our case, to translate input sentences into appropriate responses to that input. For example, the question "Did you get it?" generates a *report(get+PST(I,box))* action, in which the robot reports the truth or falsity of the predicate *get+PST(I,box)*.

The conversion will fail if definitions retrieved from the dictionary do not form a connected parse. This is appropriate, since in this particular task, precision is more important than recall: if the robot does not understand an utterance, it can ask for clarification or request to have the utterance rephrased.

One problem of the conversion is that words can be ambiguous, according to human-level comprehension abilities. For example, take the sentence "there is another one in the corner to my right". This could potentially be given two different readings: 1) the existential ("there exists another one in the corner to my right"), and 2) the locative ("another one is in the corner to my right"). This is a problem because the conversion must be deterministic: given an input, it must arrive at the same output every time. Because we currently choose definitions according to rule, this means that each word can have at most one definition for each word/valency combination. However, in the future, a statistical method of choosing the correct definition will be implemented.

4 Evaluation

There are three segments of the application to evaluate: the dependency parsing, the CCG tagging, and the semantic representation. The last two we evaluate simultaneously based on their success at identifying different types of utterances.

4.1 Data Set

In order to evaluate Mink on a domain relevant to our work, we use a spoken-data corpus of human-human dialogs [14], in which one person has to navigate a labyrinth and perform a task, guided by a second person outside the labyrinth. The two persons can only communicate via handheld devices. At present, the corpus comprises transcriptions of 12 dialogues.

4.2 Dependencies

We evaluate the accuracy of the dependency arcs by training both our parser and MaltParser on the same training material, then testing against identical test sets. We test MaltParser vs. each of the three feature sets from Table 2 and show that our parser is, in each

Table 2: *Feature Sets*

Features	Description	1	2	3
TOP.TOK	The token on top of the stack	y	y	y
TOP.POS	The part of speech of TOP	y	y	y
TOP.DEP	the dependency type of TOP	y	y	
TOP.LEFT	The dependency type of TOP's leftmost dependent	y	y	
TOP.RIGHT	The dependency type of TOP's rightmost dependent	y	y	
NEXT.TOK	The next input token	y	y	y
NEXT.POS	The part of speech of NEXT	y	y	y
NEXT.LEFT	The dependency type of NEXT's leftmost dependent	y	y	
LOOK.POS	The part-of-speech of the next plus one input	y		

case, comparable to MaltParser, as one would expect from a reimplementation of the algorithm.

In order to evaluate, we trained both parsers on the Penn treebank (PTB) [9], and tested them on 60 sentences from our corpus. The sentences are very much out of the PTB's domain, so this is purely for parser comparison. The sentences were randomly selected and tagged with Wall Street Journal (WSJ) POS tags using acopost[2], then hand-corrected in order to ensure correct POS tagging. The results can be seen in Table 3.

Table 3: *Parser Evaluation*

Metric	Malt	Set 1	Set 2	Set 3
Lbld attachmt	0.86	0.85	0.85	N/A
Unlbld attachmt	0.88	0.88	0.87	0.85
Parsing Time (s)	22.7	32.6	31.1	29.1

As can be seen, our results did not vary significantly from MaltParser's. Our results were just slightly lower, which we attribute to different machine-learning methods. Eliminating the lookahead feature had basically no effect on results, and eliminating all labeling lowered the unlabeled attachment score somewhat, though not significantly.

We also evaluate the speed of the system when running with either Mink or Malt. Since the system is used to respond to individual utterances rather than to process large texts, we evaluated this by parsing 5 sentences 10 times each and averaging the results. We show that adding incrementality so that processes can run in parallel, rather than in sequence, speeds up the entire process. The results of this evaluation are shown in Table 4.

Table 4: *Speed*

Utterance	Mink	Malt	Ratio
what is your goal	185	478.4	0.38
keep the lights on	144	534.5	0.26
cancel keep lights off	145.5	489.1	0.29
what are your orders	173.9	516	0.33
and try to report the locations of wounded people	304.8	525	0.58

4.3 CCG Tags and Semantics

For this particular task, since we are interested only in practical application, the best way of evaluating our tags is, rather than comparing them to those generated by other applications or to gold standards, seeing how successful they are at a practical task. We chose to use the parser to distinguish between different types of utterances, namely questions, commands, and statements. We chose not to use a more complex discourse scheme because we are investigating the interface between syntax, semantics, and discourse. More complex discourse schemes often make distinctions between discourse types that cannot be distinguished by syntax. That is outside the scope of the current project. Our last reason for choosing this particular task is because it is practically useful: we would like our robot to be able to respond to these types of utterances in different ways, by following commands, answering questions, and storing statements (facts that may be useful to it).

To this end, we use three different semantic dictionaries–a statement dictionary, a command dictionary, and a question dictionary–to semantically convert each statement in parallel. See Figures 5 and 6 for an example of a small dictionary, describing one question, one statement, and one command. The conversion itself fails if a given dictionary is not able to find a complete conversion for the statement. The classification fails if the utterance is correctly converted by multiple dictionaries, and thus it is not clear which type of utterance it is.

Table 5: *Sample utterances*

Type	Example
Question	$did_{S/S}\ you_{NP}\ get_{S\backslash NP/NP}\ it_{NP}$
Statement	$I_{NP}\ got_{S\backslash NP/NP}\ that_{NP/NP}\ one_{NP}$
Command	$get_{S/NP}\ that_{NP/NP}\ one_{NP}$

The obvious base for distinguishing between these types of utterances is in the verb subcategorization: Statements will, in general, have a subject to the left of the finite verb, while questions will, in general, have a subject to the right, and commands will not have an

[2] http://acopost.sourceforge.net/

Table 6: *Sample dictionary*

Word	CCG	Lambda expression
did	S/S	λ x. report(x)
get	S/NP,	λ x. get(you,x),
	S\NP/NP	λ x. λ y. get(x, y)
got	S\NP/NP	λ x. λ y. get+PST(x, y)
I	NP	I
it	NP	it
one	NP	one
that	NP/NP	λ x. find-reference(x)
you	NP	you

overt subject at all. There are, of course, exceptions to these generalities which will ultimately require a more complex method of distinguishing between them. In each case, only one semantic conversion should succeed, thus letting us know which type of utterance it is.

We evaluate 60 sentences–20 randomly chosen from each of three human-selected subsets: questions, statements, and commands– from our experimental corpus. There are four possibilities for each test example: it does not parse with any of the dictionaries; it parses with just one dictionary, and it is correctly classified; it parses with just one dictionary and it is incorrectly classified; it parses with multiple dictionaries and thus is left unclassified.

For determining which category a statement belonged to, we categorized only the explicit structure rather than the utterance's discoursal meaning. For example, "if you enter the closet you should see a box" can be seen an an implicit command to enter the closet and check for a box. However, it is explicitly a simple descriptive statement. Particularly in our case, where all utterance types are translated into some kind of response, the system needs to be able to translate only the *explicit* form of the statement into an action.

As can be seen from the results in Table 7, this method of classifying sentence types was quite successful. All utterances classified as either commands or questions were correctly classified; however, some questions and commands were classified as statements; and some statements were not classified at all.

Looking at the data, it is not surprising that questions were misclassified as statements; some question phrasing is very statement-like, and some other method will have to be deployed to identify these utterances. A few examples that were misclassified are: `we are not supposed to take the blue box though right just the blocks ?` and `so you went through the second room right ?` When humans hear such sentences, prosodic contours tell us how to interpret the sentences; but in the current system, we are not able to make use of such information.

The commands that were classified as statements were all similar to the following: `you leave it` and `and then you head back`. Again, though they appeared command-like to the human classifiers, it is obvious why they were misclassified as statements.

Table 7: *Semantics Evaluation*

Utterance type	precision	recall	f-score
Commands	1.00	0.90	0.94
Questions	1.00	0.70	0.82
Statements	0.69	0.90	0.78

5 Conclusion and Future Work

In this paper, we have described an incremental data-driven dependency parser that outputs graphs, CCG tags, and semantic representations of the input. We have shown that its accuracy is comparable to that of other data-driven dependency parsers, and that it is successful at creating useful CCG tags for practical semantic tasks.

As we move forward with this project, we will investigate more feature sets for the parser, in the hopes of finding the smallest possible set that continues to achieve high accuracy. Additionally, there are several aspects of our system that are rule-based, and we will investigate the feasibility of making every aspect statistical.

Finally, our system has largely been working with features that are extractable from the text of a dialogue. Clearly, in human-robot interaction, there is much that can be learned from other aspects of the interaction, in particular intonation. In the future, we will experiment with integrating prosodic features into the classification system.

References

[1] C. Baral, J. Dzifcak, and T. C. Son. Using answer set programming and lambda calculus to characterize natural language sentences with normatives and exceptions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pages 818–823, Chicago, Illinois, 2008.

[2] J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier. Wide-coverage semantic representations from a ccg parser. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 1240, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[3] T. Brick and M. Scheutz. Incremental natural language processing for hri. In *The Second ACM IEEE International Conference on Human-Robot Interaction*, pages 263–270, 2007.

[4] W. Che, Z. Li, Y. Hu, Y. Li, B. Qin, T. Liu, and S. Li. A cascaded syntactic and semantic dependency parsing system. In *CoNLL*, pages 238–242, Manchester, England, August 2008. Coling 2008 Organizing Committee.

[5] S. Clark. Supertagging for combinatory categorial grammar. In *In Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 19–24, 2002.

[6] K. Gold and B. Scassellati. A robot that uses existing vocabulary to infer non-visual word meanings from observation. In *The Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, 2007.

[7] R. Johansson and P. Nugues. Extended constituent-to-dependency conversion for English. In *NODALIDA 2007*, 2007.

[8] R. Johansson and P. Nugues. Incremental dependency parsing using online learning. In *The CoNLL Shared Task Session of EMNLP-CoNLL*, pages 1134–1138, 2007.

[9] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2), 1993.

[10] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[11] J. Nivre. An efficient algorithm for projective dependency parsing. In *The 8th International Workshop on Parsing Technologies*, pages 149–160, 2003.

[12] J. Nivre. Incrementality in deterministic dependency parsing. *Incremental Parsing: Bringing Engineering and Cognition Together. Workshop at ACL-2004*, 2004.

[13] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov, and E. Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

[14] M. Scheutz and K. Eberhard. Towards a framework for integrated natural language processing architectures for social robots. In *The Fifth International Workshop on Natural Language Processing and Cognitive Science (NLPCS-2008)*, 2008.

[15] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.