Incremental and Predictive Dependency Parsing under Real-Time Conditions

Arne Köhn and Wolfgang Menzel
Fachbereich Informatik
Universität Hamburg
{koehn, menzel}@informatik.uni-hamburg.de

Abstract

We present an incremental dependency parser which derives predictions about the upcoming structure in a parse-as-you-type mode. Drawing on the inherent strong anytime property of the underlying transformation-based approach, an existing system, jwcdg, has been modified to make it truly interruptible. A speed-up was achieved by means of parallel processing. In addition, MaltParser is used to bootstrap the search which increases accuracy under tight time constraints. With these changes, jwcdg can effectively utilize the full time span until the next word becomes available which results in an optimal quality-time trade-off.

1 Introduction

Users prefer incremental dialogue systems to their non-incremental counterparts (Aist et al., 2007). For a syntactic parser to contribute to an incremental dialogue system or any other incremental NLP application, it also needs to work incrementally. However, parsers usually operate on whole sentences only and few parsers exist that are capable of incremental parsing or are even optimized for it.

This paper focuses on using a parser as part of an incremental pipeline that requires timely response to natural language input. In such a scenario, delay imposed by a parser's lookahead is more severe than delay caused by parsing speed since the parsing speed is capped by the user's input speed. Depending on the input method, the maximum typing speed varies between 0.75 seconds per word (qwerty keyboard) and 6 seconds per word (mobile 12-key multi-tap) (Arif and Stuerzlinger, 2009) and is usually lower if the sentence has to be phrased while typing.

In such a scenario, the objective of the parser is to yield high quality results and produce them as soon as they are needed by a subsequent component. It is rarely known beforehand when the next word will be available for processing. Therefore, in an incremental pipeline a) computation should continue until the next word occurs if this might contribute to a better result, and b) a new word should be included immediately to avoid delays. A parser which works pull-based, i.e. processes one prefix until it is deemed finished and then pulls the next word, is insufficient under conditions, since it would require to determine the time used for processing before the processing can even start. Either the estimated processing time will be too short, violating a), or it will be too long, violating b). In contrast, push-based architectures can meet both requirements since the processing of the prefix can be interrupted when new input is available.

Beuck et al. (2011) showed that Weighted Constraint Dependency Grammar-based parsers are capable of producing high-quality incremental results but neglected the processing time needed for each increment. In this paper, we will use jwcdg[1], a reimplementation of the WCDG parser written in Java. jwcdg uses a transformation-based algorithm. It comes equipped with a strong anytime capability, causing the quality of the results to depend on the processing time jwcdg is allowed to consume. We will show that jwcdg can produce high quality results even if only granted fairly low amounts of processing time.

1.1 Incremental Predictive Dependency Parsing

Dependency parsing assigns every word to another word or NIL as its regent and the resulting edges are annotated with a label. If dependency analyses

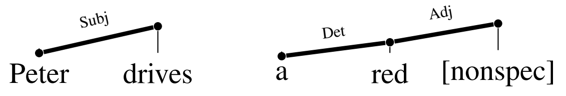
[1]`http://nats-www.informatik.uni-hamburg.de/CDG/`; detailed resources for the experiments in this paper can be found there as well.

are used to describe the syntactic structure of sentence prefixes, different amounts of prediction can be provided. The interesting cases are those where either the regent or a dependent is not yet part of the sentence prefix.

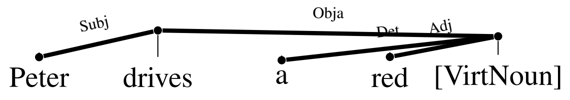
If the regent of a word *w* is not yet available, the parser can make a prediction about where and how *w* should be attached. One possibility is to simply state that the regent of *w* lies somewhere in the future without giving any additional information. This can be modelled by attaching *w* to a generic *nonspec* node (Daum, 2004). Beuck et al. (2011) call this *minimal prediction*.

However, it is usually possible to predict more: The existence of upcoming words can be anticipated and *w* can then be attached to one of these words. Of course, most of the time it will not be possible to predict exact words but abstract pseudo-words can be used instead that stand for a certain type such as nouns or verbs. Beuck et al. (2011) call these pseudo-words *virtual nodes* and the approach of using virtual nodes *structural prediction* (because the virtual nodes accommodate crucial aspects of the upcoming structure of the sentence). A virtual node can be included into an analysis to represent words that are expected to appear in later increments.

As an example, "Peter drives a red" can be analyzed as



using minimal prediction or as



using structural prediction. Minimal prediction leads to disconnected analyses while structural prediction allows for connected analyses which resemble the structure of whole sentences. Tn this case, the analysis includes the information that the regent of "red" is the object of "drives", which is missing in the analysis using minimal prediction.

1.2 Challenges in Incremental Predictive Parsing

The key difference between non-incremental and incremental parsing is the uncertainty about the continuation of the sentence. If a prediction about upcoming structure is being made, there is no guarantee that this prediction will be accurate.

Using a beam of possible prefixes, as done in

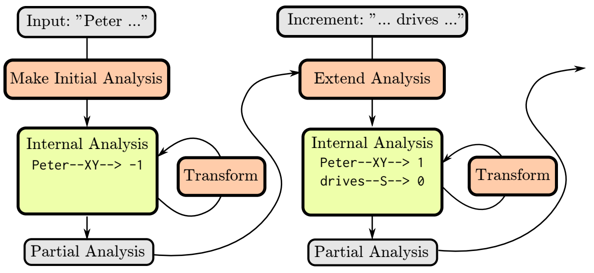


Figure 1: Incremental Parsing with jwcdg

Demberg-Winterfors (2010), is a strategy to deal with this uncertainty. It guarantees that each new analysis is an extension of an old one. With this approach, the whole beam becomes incompatible with the observed sentence continuation if no fitting prediction is contained in the beam. Thus, such sentences can not be parsed.

Minimal prediction, as another option, largely abstains from prediction. This allows for monotonous extensions even without a beam since the analysis of a prefix will not be incompatible with the continuation of the sentence. This approach is used by MaltParser (Nivre et al., 2007).

A transformation-based parser, finally, can also deal with non-monotonic extensions. In contrast to beam search, only a single analysis is generated for each prefix and there is no guarantee that the analysis of a prefix p_n is a monotonic extension of p_{n-1}. Because the analysis of p_{n-1} is only used to initialize the transformation process, the search space is not restricted by the results that were obtained in former prefixes although they still guide the analysis.

2 WCDG Parsing

In the Weighted Constraint Dependency Grammar formalism, a grammar is used to judge the quality of analyses. The grammar consists of constraints that are evaluated on one or more edges of an analysis. If a constraint is violated, a penalty is computed. Constraints can incorporate more edges into their computation than the edges they are evaluated on (McCrae et al., 2008). They can traverse the current analysis by using special predicates. This way, a constraint evaluated on one edge could, for example, check if that edge is adjacent to an edge with certain properties. If a constraint uses such predicates, it is considered *context sensitive*.

In the WCDG formalism, the best analysis of a sentence is defined by

$$ba = \underset{a \in Analyses}{\arg\max} \prod_{c \in Conflicts(a)} penalty(c)$$

where the *conflicts* are the parts of an analysis that stand in conflict with the grammar and $penalty(c)$ is the penalty that the grammar assigns to the conflict c.

2.1 The Frobbing Algorithm

jwcdg tries to find an analysis by transforming a given one until it cannot be improved further. The algorithm employed for this purpose is called frobbing (Foth et al., 2000).

Frobbing consists of two phases: first, the problem is initialized. In this phase, all possible edges are constructed and the constraints defined for a single edge are evaluated on them. An initial analysis is constructed using the best-scored edge for every word, which is repeatedly transformed in the second phase. Frobbing is described as pseudocode in Algorithm 1. A set of conflicts (constraints that are violated on specific edges) is computed and the most severe of them is attacked by transforming the analysis (attackConflict, line 7). This either results in a (not necessarily better) analysis that does not have this conflict or in the insight that the conflict cannot be removed (line 12). The algorithm repeatedly tries to remove the most severe conflict. If in this process a new best analysis is found, it is marked as the new starting point. If the conflict can not be removed, the algorithm tracks back to the starting point. The procedure can be interrupted at any time and the best analysis that was found up to that point will be returned.

In its incremental mode, jwcdg works as depicted in Figure 1. The new word is added to the previous analysis using the best edge. The frobbing algorithm is then run until no better result can be found or the parser is interrupted. To allow for prediction, either a *nonspec* node (Daum, 2004) or a set of virtual nodes (Beuck et al., 2011) is added to the set of words. This way, all changes regarding incrementality are completely transparent to the frobbing algorithm, only the constraints in the grammar need to be aware of virtual nodes and `nonspec`.

3 Related Work

One of the few broad-coverage parsers that are capable of incremental processing is PLTAG, a Tree

```
   Data: sentence S
   Result: Analysis of S
1  List removedConflicts ← [ ];
2  Analysis best ← makeInitialAnalysis(S);
3  Analysis current ← best;
4  Conflict initialConflict ←
   getHardestSolvableConflict(current);
5  while solvable conflicts remain do
6      Conflict c ←
       getHardestSolvableConflict(current);
7      current ← attackConflict(c, current,
       removedConflicts);
8      if score(current) > score(best) then
9          best ← current;
10         reset();
11         initialConflict ←
           getHardestSolvableConflict(current);
12     else if current == null then
13         setUnresolvable(initialConflict);
14         current ← best;
15         removedConflicts ← [ ];
16     end
17 end
18 return best;
```

Algorithm 1: `frobbing`

Adjoining Grammar parser that tries to model psycholinguistic phenomena such as surprisal, paying less attention to high parsing accuracy or fast parsing speed. It works incrementally and provides predictions for upcoming structure. Since PLTAG uses beams that get expanded, it needs a lookahead of one word to reduce ambiguity (Demberg-Winterfors, 2010, p. 217). Even with this lookahead, some sentences can not be parsed: PLTAG has a coverage of 93.8% on sentences of the Penn Treebank with less than 40 words.

MaltParser is a shift-reduce based parser. It uses a classifier to determine locally optimal actions from a set of possible actions of a parsing algorithm. The classifier is trained using manually annotated sentences. MaltParser can use several different parsing algorithms. In this paper, the *2planar* algorithm described in Gómez-Rodríguez and Nivre (2010) will be used, which is able to produce non-projective analyses.

While MaltParser's processing is fast compared to jwcdg, a lookahead of one word already translates into a delay of one to three seconds, depending on the input speed of the user. Beuck et al. (2011) showed that, at least for German, Malt-

input:	The	fox	jumped	over
tagger:	The/DT	fox/NN	jumped/VBD	
parser:	stack: [The]			
output:	[none]			

Figure 2: Effect of lookahead for MaltParser

Parser suffers from a fairly small decrease in accuracy if only features from the next two words instead of three are used. However, since the PoS tags are needed and a PoS tagger needs lookahead as well to achieve good accuracy, a lookahead of at least three words is needed for the whole tagger-parser pipeline to achieve a high accuracy. The effect of this delay is illustrated in Figure 2. In addition, MaltParser is not capable of producing structural predictions.

4 Parallelizing jwcdg

Among the two phases of frobbing, only the second one can be interrupted. Therefore, initialization needs to be faster than the shortest time limit we would like to impose. Initialization is mostly concerned with evaluating constraints on all edges that come from or point towards the new word. To make this judgement faster, the code has been changed so that it can be done in parallel, using worker threads instead of a sequential constraint evaluation.

Table 1 shows the time needed to construct new edges and judge them while parsing a subset of the NEGRA corpus (Brants et al., 2003).[2] Although the median time is already relatively good in the non-parallelized case, its maximum amounts to almost two seconds. Parallelization brings most benefits for the more complex initializations: the time needed for the 3rd quartile scales nearly linear up to eight cores. More than sixteen cores yield no further improvement.

With the parallelized initialization, jwcdg can spend more time on transforming analyses. In addition, it is able to perform anytime parsing with a lower bound of about 200 ms per word on current hardware.

The heart of the frobbing algorithm is the attackConflict method which, given an analysis a and a conflict c, systematically tries all changes of edges that are part of c. It then returns the best resulting analysis that does not violate the constraints $constr(c)$. Since these transformations all work independently, they have been parallelized in the same manner as the initialization. The result can be seen in Table 2. While the introduction of parallelized code causes a small overhead, using two cores already provides a noticeable benefit. The parallelized code can benefit from up to 32 cores, yet the overhead of managing more cores results in a sub-linear speedup.

These optimizations have a noticeable impact on parsing performance under time pressure: With a time limit of two seconds per word, jwcdg$_{base}$ scores an unlabeled accuracy of 72.54% for the final analyses, while jwcdg$_{parallel}$ scores 76.29%. jwcdg$_{base}$ only reaches this accuracy with a time limit of four seconds. Unless otherwise noted, all evaluations have been carried out on sentences 18602 to 19601 of the NEGRA corpus.

5 MaltParser as a Predictor for jwcdg

When facing a tight time limit, jwcdg has only very little time to improve an analysis by transforming it. Therefore, with tighter time limits a good initial attachment becomes more important and a method which provides frobbing with a good initial analysis could help to achieve drastically better results.

Foth and Menzel (2006) showed that WCDG can be augmented by trainable components to raise the accuracy of WCDG. The output of these predictors was converted into constraints to help WCDG finding a good analysis. One of the components was a shift-reduce parser modeled after Nivre (2003), which was the first description of the MaltParser architecture. Although the shift-reduce parser was relatively simple compared to MaltParser and had a labeled accuracy of only 80.7 percent, it helped to raise the accuracy of WCDG from 87.5 to 89.8 percent. This approach has later been used by Khmylko et al. (2009) to integrate MST-Parser (McDonald et al., 2006) (which does not work incrementally) as an external data source for WCDG. We integrated MaltParser in a similar way.

MaltParser consumes the input from left to right and, if using the 2planar algorithm, constructs edges as soon as possible: An edge can only be created between the word on top of the stack and the current input word. This means that every edge has to be constructed as soon as the second word

[2]All experiments have been carried out on a 48-core machine with four AMD Opteron 6168 processors.

	number of threads used							
	np	1	2	4	8	16	32	48
1st Qu.	43	45	23	13	9	8	8	9
Median	86	91	46	26	17	14	14	15
Mean	161	170	86	47	30	25	24	27
3rd Qu.	186	197	100	56	36	31	31	34
Max.	1940	2049	1029	553	433	200	197	555

Table 1: Timing requirements in ms of the initialization phase for different thread numbers; np = not parallelized (time limit per word = 16 seconds)

	number of threads used							
	np	1	2	4	8	16	32	48
1st Qu.	19	20	14	10	8	7	6	6
Median	62	64	48	36	28	24	22	22
Mean	181	190	141	110	92	81	76	79
3rd Qu.	193	199	153	117	95	84	76	77
Max.	12112	13611	8033	8734	8469	6376	8554	6297

Table 2: Timing requirements in ms of attacking conflicts (Algorithm 1, Line 7) for different thread numbers; np = not parallelized (time limit per word = 16 seconds)

of the edge is the current input word. As soon as that word gets shifted onto the stack, the creation of the edge would no longer be possible.

The parser works monotonically since edges are only added to but never removed from the set of edges. This means that all decisions by the parser are final; if word a is not attached to word b, we can be sure that a will never be attached to b in subsequent analyses. As a corollary, if MaltParser has an accuracy of X percent on whole sentences, the probability that a newly created edge is correct will also be X percent.

As a delay is not acceptable for our application scenario, we will use MaltParser and the TnT tagger (Brants, 2000) without lookahead despite their inferior accuracy in that mode[3].

5.1 An Interface Between MaltParser and jwcdg

A predictor (MaltPredictor) for jwcdg has been implemented that uses a newly written incremental interface for MaltParser so that the regents and labels predicted by MaltParser can be accessed by constraints as soon as they become available. MaltPredictor uses the PoS-tags that are provided by the tagger predictor. Each time a new word w is pushed to jwcdg, MaltPredictor forwards w together with its PoS-tag onto MaltParsers input queue and runs MaltParser's algorithm until a shift operations occurs. With this shift operation, w is consumed from the input queue. If the sentence is marked as being finished, MaltParser is run until it has fully parsed the sentence. MaltPredictor then reads the state of MaltParser and stores for each word the regent it has been assigned to by MaltParser. If Maltparser did not assign a regent to a word, this fact is also stored. Since – as already mentioned – MaltParser works eagerly (i. e. constructs every edge as soon as possible), the regent of such a word must either lie in the future or be the root node.

The three constraints that are used for accessing MaltParser's analyses are depicted as pseudocode in Figure 3. If only these constraints and the tagger constraint (which selects the PoS-tag for every word) are used as a grammar, jwcdg will parse sentences exactly as MaltParser does. This way, jwcdg acts as an incremental interface to MaltParser.

The first two constraints are only applicable if MaltPredictor has made a prediction for the word in question. The first constraint checks whether

[3]both were trained on sentences 1000 to 18000 of the negra corpus

```
prediction_exists(word)
 -> regent_of(word) =
      predicted_regent(word)

prediction_exists(word)
 -> label_of(word) =
      predicted_label(word)

not prediction_exists(word)
 -> (regent(word) is virtual or
      regent(word) is nonspec or
      regent(word) is NIL or
      word is virtual)
```

Figure 3: Constraints for incorporating MaltParser's results into jwcdg

the regent of a word is the one that has been selected by MaltParser. The second constraint checks that the predicted label matches the label of the edge in the analysis given that a label has been predicted. The third constraint is not as straightforward as the other two: Since we know that MaltParser creates edges as soon as possible and we know that MaltParser has not created an edge with this word as dependent, either the regent lies in the future (i. e. it should be a virtual node in jwcdg's analysis) or the regent is NIL (as MaltParser does not explicitly create edges to NIL while parsing). In the other possible case, the dependent of the current edge is a virtual node. In this case MaltParser cannot possibly predict an attachment. A parameter tuning on sentences 501 to 1000 of the NEGRA corpus has shown that a penalty of 0.9 works best for these constraints.

When the input sentence is going to be extended, the new word is attached using the edge that violates the least unary, non context-sensitive constraints. The MaltParser constraints are unary and not context sensitive and therefore jwcdg will use the edge predicted by MaltParser if no other constraints prevent it from doing so.

5.2 Comparison of MaltParser and jwcdg

To compare MaltParser and jwcdg, the richer prediction of jwcdg has to be transformed into minimal prediction. In this mode, every attachment of a word to a virtual node is considered correct if the word is attached to an upcoming word in the gold standard. The two accuracies that are used for evaluation are *initial attachment* accuracy (how often is the newest word attached correctly?) and the *final accuracy* (how many attachments are correct in the parse trees for the whole sentences?).

Figure 4 shows the accuracy for initial attachment and final accuracy as a function of a given time limit. Since MaltParser does not use an anytime algorithm, its results are the same for all time limits[4]. Note that the labeled initial attachment score is fairly low because MaltParser can not predict labels for edges that have `nonspec` as the regent. When ignoring labels, Maltparser's initial attachment accuracy is higher than its final accuracy since it does not change edges it has created (therefore the accuracy cannot rise) and words can be counted as correct initially but wrong in the final analysis: If the correct regent of a word lies somewhere in the future, the initial decision to not attach it is counted as correct. If it is later attached to a wrong word, it will be counted as wrong in the final accuracy.

As can be seen, jwcdg outperforms MaltParser in most aspects when given enough time. The only exception is the unlabeled attachment score for the initial attachment. Here, however, one has to keep in mind that jwcdg produces more informative predictions with virtual nodes, which is not honored in this evaluation.

5.3 Enhancing jwcdg with MaltParser

jwcdg$_{malt}$ has been derived from jwcdg$_{parallel}$ by adding the MaltParser constraints discussed before to the grammar.

The results (Figure 4) show that jwcdg$_{malt}$ has a considerably higher initial accuracy than jwcdg$_{parallel}$, more than ten percentage points for a time limit of one second. The final accuracy is noticeably better as well. This shows that MaltParser's output helps jwcdg find a good initial analysis which can then be optimized by jwcdg.

6 Evaluation on Additional Corpora

The evaluations discussed so far have been carried out on the NEGRA corpus. NEGRA consists of newspaper texts and thus represents a very specific kind of text. However, most sentences from other text types (e.g. chat) are shorter and have a lower structural complexity. This section tries to measure the impact of these differences between different data sources.

[4]MaltParser parses fast enough to never violate the time limit of one second per word.

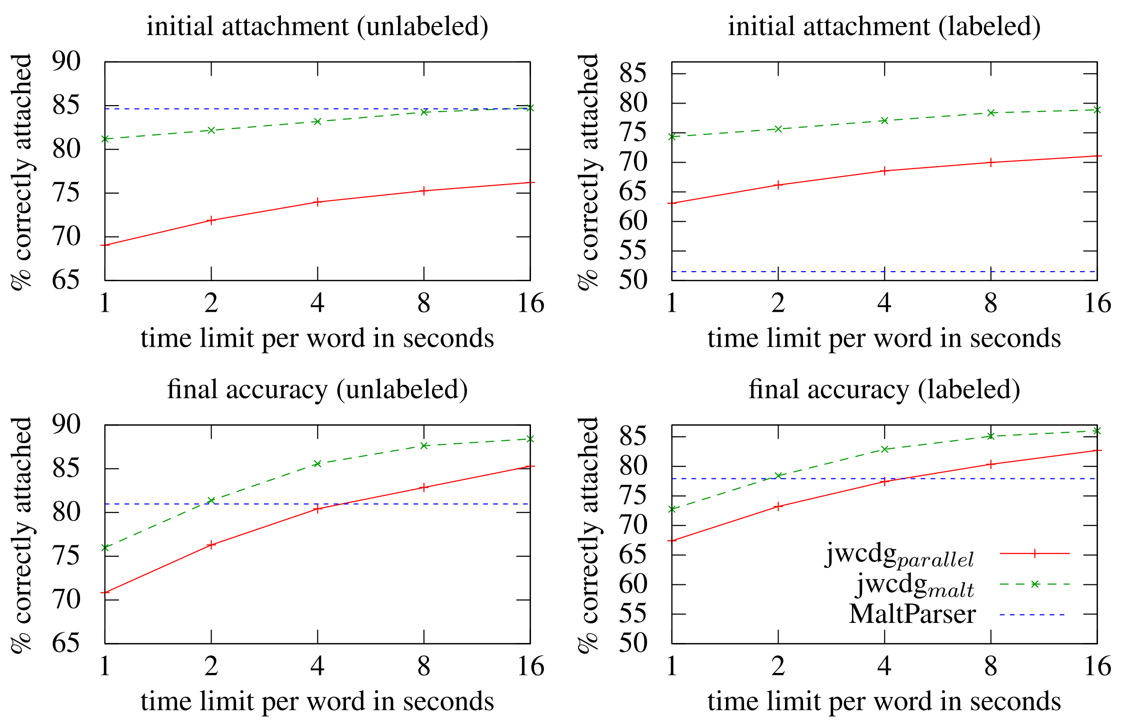


Figure 4: Comparison of jwcdg and MaltParser using minimal prediction

6.1 Evaluation on Subsets of NEGRA

To evaluate how sentence length influences the parsing results, jwcdg has been evaluated on a subset of the NEGRA corpus. Sentences with a length of less than three have been excluded. In addition, sentences longer than twenty have been excluded. The resulting subset NEGRA-3-20 contains 57.8% of the sentences of the original NEGRA test set.

The results shown in Figure 5 confirm the assumption that jwcdg's accuracy increases when being evaluated only on short sentences. This difference could be due to the syntactic simplicity of short sentences and the shorter initialization time needed for short increments leaving more time for the transformation. In addition, jwcdg$_{malt}$ already approaches its best result with a time limit of four seconds.

6.2 Evaluation on the creg-109 Corpus

Since interactive Computer-Assisted Language Learning could be an interesting application domain for an incremental parser, an additional evaluation has been conducted on the creg-109 corpus, a set of 109 sentences of the corpus described in Meurers et al. (2010). The creg-109 corpus "consists of answers to German reading comprehension questions written by American college students learning German" (Meurers et al., 2010).

Figure 6 shows the accuracy of the different parser versions on this corpus. The results have a different pattern than the ones for the NEGRA corpus: jwcdg$_{parallel}$ has nearly the same initial attachment accuracy as jwcdg$_{malt}$ and even slightly outperforms it in the final score. In addition to that, the result does not improve much with a time limit of more than two seconds. Both phenomena could be due to the lesser syntactic complexity of the sentences so that the MaltParser's analyses are not so beneficial for guiding jwcdg.

7 Conclusion

We have shown that it is possible to gain parse-as-you-type speed with good accuracy using a combination of different incremental approaches to dependency parsing. Our solution based on a parallelized version of jwcdg benefits from using up to 32 cores. In contrast to other approaches, which have to make algorithmic refinements, jwcdg can take advantage from the advances in processing speed due to its anytime property. MaltParser turned out to be a good predictor that helps jwcdg to produce good analyses earlier.

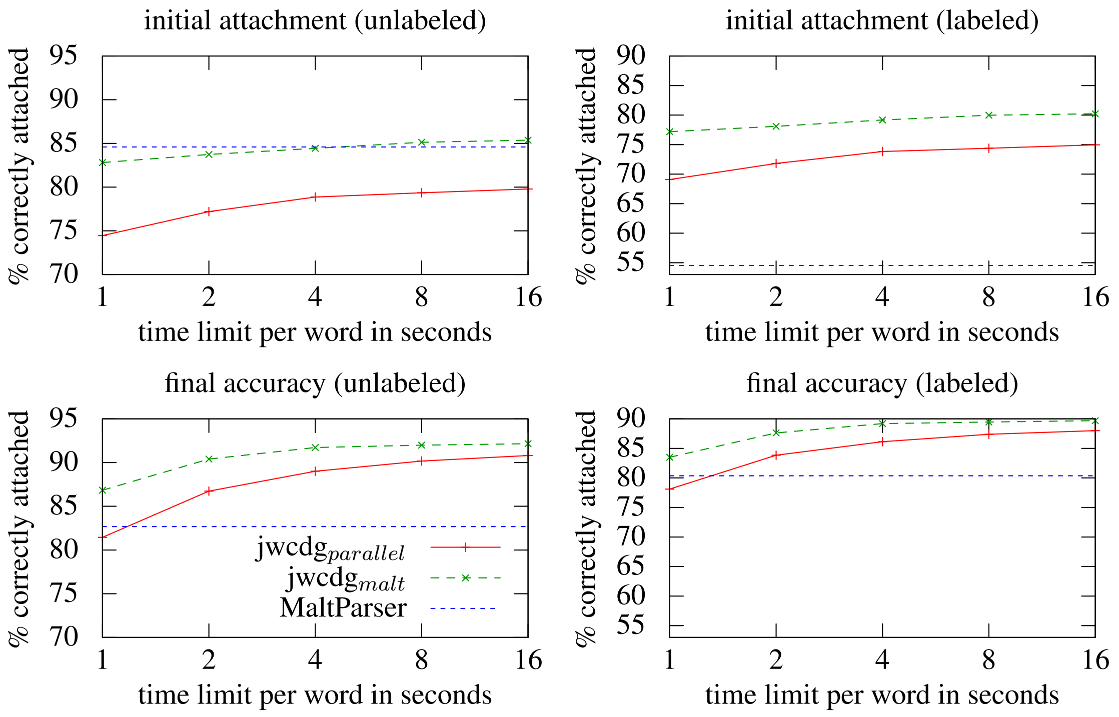


Figure 5: Comparison on NEGRA-3-20 using minimal prediction

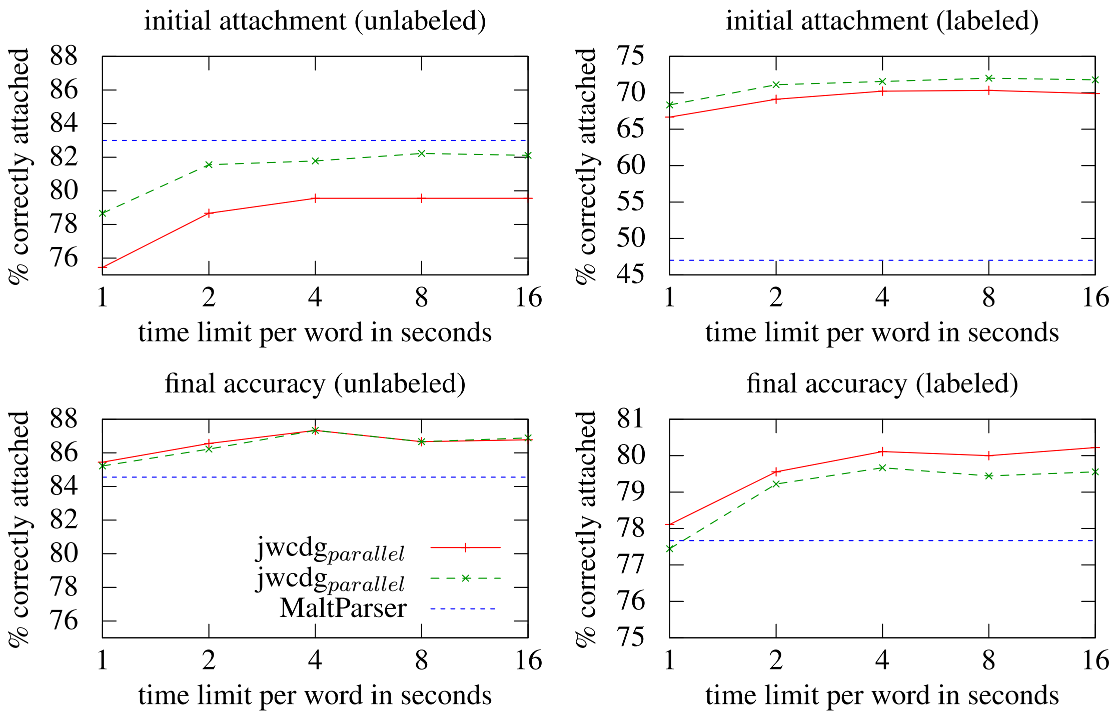


Figure 6: Comparison on creg-109 using minimal prediction

References

Gregory Aist, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. 2007. Incremental dialogue system faster than and preferred to its nonincremental counterpart. In *Proceedings of the 29th Annual Meeting of the Cognitive Science Society*, pages 761–766.

A.S. Arif and W. Stuerzlinger. 2009. Analysis of text entry performance metrics. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, pages 100–105.

Niels Beuck, Arne Köhn, and Wolfgang Menzel. 2011. Incremental parsing and the evaluation of partial dependency analyses. In *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011.

Thorsten Brants, Wojciech Skut, and Hans Uszkoreit. 2003. Syntactic annotation of a german newspaper corpus. In Anne Abeillé and Nancy Ide, editors, *Treebanks*, volume 20 of *Text, Speech and Language Technology*, pages 73–87. Springer Netherlands.

Thorsten Brants. 2000. TnT: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Daum. 2004. Dynamic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 67–73, Stroudsburg, PA, USA. Association for Computational Linguistics.

Vera Demberg-Winterfors. 2010. *A Broad-Coverage Model of Prediction in Human Sentence Processing*. Ph.D. thesis, University of Edinburgh.

Kilian A. Foth and Wolfgang Menzel. 2006. Hybrid parsing: Using probabilistic models as predictors for a symbolic parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 321–328, Sydney, Australia. Association for Computational Linguistics.

Kilian A. Foth, Wolfgang Menzel, and Ingo Schröder. 2000. A transformation-based parsing technique with anytime properties. In *4th Int. Workshop on Parsing Technologies, IWPT-2000*, pages 89 – 100, Trento, Italy.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden. Association for Computational Linguistics.

Lidia Khmylko, Kilian A. Foth, and Wolfgang Menzel. 2009. Co-parsing with competitive models. In *Proceedings of the International Conference RANLP-2009*, pages 173–179, Borovets, Bulgaria. Association for Computational Linguistics.

Patrick McCrae, Kilian A. Foth, and Wolfgang Menzel. 2008. Modelling global phenomena with extended local constraints. In Jørgen Villadsen and Henning Christiansen, editors, *Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008, Hamburg, Germany)*, pages 48–60. Roskilde University.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 216–220, Stroudsburg, PA, USA. Association for Computational Linguistics.

Detmar Meurers, Niels Ott, and Ramon Ziai. 2010. Compiling a task-based corpus for the analysis of learner language in context. In *Pre-Proceedings of Linguistic Evidence 2010*, pages 214–217, Tübingen.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160.