Universität Hamburg
Fachbereich Informatik

master's thesis

Predictive Incremental Dependency Parsing Under Time Constraints

Arne Köhn

October 2012

Betreuer: Prof. Dr.-Ing. Wolfgang Menzel
Zweitgutachter: Prof. Dr. Thomas Ludwig

The results of this master's thesis would not have been possible without Niels Beuck's seemingly crazy idea to reimplement cdg in Java. Niels did the majority of that work, which took about half a year. I would like to thank Christine Köhn for proof reading and her very helpful suggestions. My parents provided support by allowing me to not to have to worry about the real world. Thanks to all the people that provided helpful suggestions.

This is not the submitted document but a corrected version, which should reduce the number of typos. The content is unchanged except for the corrected value for "final attachment parallel" in the 16 sec column of Table 5.3.

Contents

1. Introduction

When humans interact with a computer using natural language the computer needs to understand natural language at least at a rudimentary level. The more sophisticated the interaction should be the more the computer needs to understand. Since syntax is a key element in understanding natural language, syntactic representations are used in most natural language processing tasks. Generating such representations from sentences is called *parsing* and is mostly done on whole sentences.

In dialogues, participants often react verbally or non-verbally to sentences which are still being uttered. In order to react to an unfinished sentence, it needs to be processed. This means that a system that parses only complete sentences makes human-computer interaction unnatural since the computer cannot react in the same way a human would. In fact, humans prefer dialogue systems that can already react to unfinished sentences, even factoring out the speedup that comes with incremental interaction (Aist et al. 2007). Humans generate syntactic analyses that already predict syntactic structure for incomplete sentences (Sturt and Lombardo 2005). If machines should interact naturally with humans, they need to do the same.

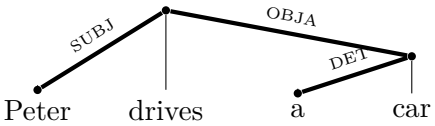
This thesis investigates how predictive incremental parsing can be done under strict time constraints so that the results can be used in an interactive setting.

1.1. Dependency Parsing

There are several formalisms for representing syntactic structure, the one that will be used in this thesis is parsing into a dependency graph. When doing dependency parsing, each word[1] is either assigned to another word, the word's regent, or its regent is set to NIL, meaning "not attached anywhere". Usually cycles are forbidden so that the resulting graph is a forest, i. e. a set of trees[2]. The words whose heads are NIL are the roots of trees. The following is an exemplary analysis:

[1] Non-words such as punctuation need to be processed as well. In this thesis "word" will be used both for words and non-words for readability reasons.

[2] Forbidding cycles leads to directed acyclic graphs. Together with the fact that every word has only one regent it means that every graph is a forest.



A label is assigned to each edge. These labels denote the type of the dependency. "SUBJ" for example denotes that "Peter" is the subject of the verb "drives". Usually dependency parsers are judged only based on the question which head was assigned to a word and which label was assigned to the edge. There is, however, more information a parser can provide, such as information about the word class ("drives" can be a verb as in the sentence above or a noun as in "hard disk drives"), its case and so on.

When looking at an edge, the word that has been attached is called the *dependent* and the word the dependent is attached to the *regent*. We can also say "the regent of 'drives' is NIL", meaning drives has been attached to NIL. A word w can be attached to any other word (or to NIL) and use any possible label from a fixed set of labels. In addition, it can have different readings (e. g. noun or verb reading for "drives"). The set of all these possible combinations is called a *domain*:

$$domain(w) = \{words \setminus \{w\} \cup \{NIL\} \times labels \times readings(w)\}$$

With the notion of domains at hand, dependency parsing can also be described as the task of selecting one edge from each domain so that the resulting graph is a fitting analysis of the sentence.

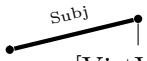
1.2. Incremental Parsing

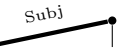
Usually, parsing is only done on whole sentences (in contrast to partial sentences) since texts are assumed to be available as a whole when they get analyzed. However, this is not always the case: In an interactive setting like human-computer interaction or the observation of human-human interaction, sentences may be available to the system while still being produced and the computer could make use of that information. Use cases could be a chat bot or an interactive text translation program. If the computer has access to a sentence while it is being produced, it could make use of intermediate analyses to enable further processing of the yet unfinished sentence.

A parser is incremental if it produces analyses for every sentence prefix. However, if the correct regent of a word is right of it, the regent might not yet be available. An example for this are determiners, which are always attached to their noun. An incremental parser needs to be able to cope with this problem of not-yet-available regents instead of forcibly attaching the word incorrectly.

1.3. Predictive Parsing

If the regent of a word w is not yet available, the parser can make a prediction where w should be attached to. One possibility is to only state that the regent of w lies somewhere in the future without giving any additional information. This approach is called *minimal prediction* (Beuck, Köhn, and Menzel 2011b).

It is possible to predict more: The existence of upcoming words can be predicted and w can then be attached to one of those words. It is of course most of the time not possible to predict exact words. However, abstract pseudo-words that stand for a certain type of words such as nouns or verbs can be predicted. We will call those pseudo words *virtual nodes* and the approach of using virtual nodes *structural prediction* (because the upcoming structure of the sentence is predicted) (ibid.). A virtual node can be included into an analysis to fill blanks that will be filled by real words in later increments. The analysis of "Peter [...]" could then be 

, whereas the analysis of "Peter drives" could either be 

or 

depending on whether we predict that the sentence should continue or not. The latter case shows that virtual nodes can not only be included because of missing regents but also because a word can suggest or even require additional dependents. In the case shown above "drives" is usually used in conjunction with an object so it can be predicted in the analysis.

1.4. Possible Applications for Predictive Incremental Parsing

The scope of applications for predictive incremental parsing lies mostly in areas in which speech or language is produced and a system is required to react to it.

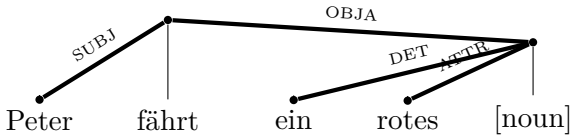
1.4.1. Intermediate Semantic Representations

Predictive intermediate results can be used to generate semantic representations of partial sentences which in turn can be further utilized. A partial syntactic analysis that is predictive has a structure which resembles one of a complete sentence. If no prediction is made, partial analyses tend to disintegrate into disconnected trees. The benefit of structural prediction can be seen when trying to generate a semantic analysis for "Peter fährt ein rotes", which is a prefix of "Peter fährt ein rotes Auto" ("Peter

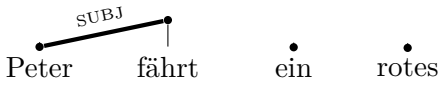
with str. prediction	without str. prediction
Peter[x8,]	Peter[x8,]
SUBJ[x9,x8,]	SUBJ[x9,x8,]
fahren[x9,]	fahren[x9,]
OBJA[x9,x10,]	
∃ x10(P)(Q)	∃ x11(P)(Q)
rot[x10,]	rot[x12,]

Table 1.1.: Semantic analysis of "peter fährt ein rotes" with and without structural prediction

drives a red car"). With structural prediction the following analysis will be obtained[3]:



Without structural prediction, neither "ein" nor "rotes" can be attached anywhere:



When extracting a semantic representation from these analyses, the advantage of structural prediction becomes clear. The semantic representation shown in Table 1.1 has been produced by the LRS (Lexical Resource Semantics) analyzer described in Hahn and Meurers (2011). The semantic formula derived from the analysis without structural prediction fails to capture the identity of the object that is red (x12) and the object whose existence is derived from the determiner "ein" ("a"). It also does not include the information that there is an object that is being driven. The formula derived from the analysis with structural prediction on the other hand is able to infer that there is an object (x10) which is the object of "fahren" (i. e. it is driven by peter) and which is red.

[3]This example has been created by Niels Beuck. The analyses are results of jwcdg running in predictive and non-predictive mode.

1.4.2. Incremental User Interaction

Dialogue systems are usually built using a tight turn-taking approach, which is quite unnatural. Skantze and Schlangen (2009) have created an incremental dialogue system for the domain of dictating numbers and have shown that users prefer it to a non incremental version. To make possible such systems for less focused domains, an broad coverage incremental parser is needed. Incremental interaction can also take place in multi-modal interaction or when using text input, for example with mobile devices: If the processing takes place incrementally, the information already gathered from the user's utterance can be displayed on the screen enabling shorter feedback loops. Partial analyses could also be used to predict which action the user wants to be executed and present a list of possible actions before the request has been finished. This could save time, especially if the input method is slow which is the case on mobile devices, for example.

Having access to partial analyses can even be of use for a system without incremental interaction. If the system runs database queries that can take some time, it can yield faster reactions if it was able to guess the meaning of the whole sentence beforehand.

1.4.3. Predictive Parsing for ICALL Systems

Intelligent Computer-Assisted Language Learning (ICALL) systems can make use of components that recognize grammatical errors in sentences written by the learner. For this constraint based syntactic parsers can be used (Menzel and Schröder 1998a; Boyd 2012). If the parser works incrementally, errors can already be detected while the sentence is being typed. This would make a faster interaction with the learner possible.

If the continuation of a sentence can be predicted, additional help can be given to a learner while she is typing a sentence in a learner program. For example, problems such as case mismatches and the syntactic structure as analyzed by the parser could be displayed as a feedback to the user.

1.5. Aspects of Incrementality

Until now we have looked at incremental parsers as if incrementality would be a binary feature. It is not. Incrementality can be seen from different angles (Beuck, Köhn, and Menzel 2011a):

- Does the system commit to output already made (*monotonic*) or is the system allowed to retract previously given information (*reanalysis*)?

- Is the processing delayed until more input is available so that a more informed decisions can be made? (*lookahead*)
- Is only one result produced or does the system produce multiple (possibly weighted) results? (*decisiveness*)

If monotonic output is guaranteed, the parser is not permitted to change an attachment that has already been made. A non-attached word can, however, be attached and a virtual node can be replaced by a real word since this is merely adding information, not changing. In fact, each virtual word *must* be replaced by a real word at some point in the future since the analysis of a whole sentence should not include virtual nodes. This is problematic if for a predicted virtual node no fitting word turns up. Therefore, prediction without the possibility of reanalysis is risky and should only be done if one is sure that a fitting word will follow. Guaranteeing monotonicity usually leads to a loss of accuracy since decisions that are clearly wrong in retrospect cannot be repaired. On the other hand, reanalysis taken to its extreme form can make the intermediate results worthless if every new analysis can completely retract the information of the old one and the consumer has to throw away all of the results it has produced based on the old analysis.

If lookahead is used, there are already additional words available that are useful for parsing. If a lookahead of one word is used when parsing the sentence "When Fred eats food gets thrown"[4], it is already known that "food" will be succeeded by "gets" when having to decide on the attachment of "food". This could save the program from making the wrong decision (i. e. attaching "food" to "eats"). Therefore lookahead can be used to avoid or reduce the loss of accuracy imposed by monotonicity.

Another approach to mitigate errors is to produce multiple results. A parser can produce multiple analyses and rank them to indicate how likely they are. This way monotonic predictive parsing can be done by producing different possible predictions and later on expand the ones that fit the actual words. This approach is used in the PLTAG parser (Demberg-Winterfors 2010). However, the consumer of the output has to be able to cope with multiple results else it would need to pick one of them and the benefit vanishes. This approach of reduced decisiveness is also used by the TnT part-of-speech tagger that will be used in this thesis.

There is a clear tradeoff to be made between accuracy on the one hand and monotonicity, (lack of) delay and decisiveness on the other hand. To achieve a high accuracy, one has to either compromise on the monotonicity, the decisiveness, or has to accept a delay in decision making. This means that if one is interested in

[4]This is a common example, no clear origin could be found.

accurate and unique results, either lookahead or reanalysis has to be used. Lookahead is, however, unsuited for interactive environments since it makes timely reactions impossible. This leaves reanalysis as the only way to get decisive parsers with a high accuracy for interactive settings.

1.6. Parsers Capable of Incremental Parsing

At the moment, to our knowledge there are two dependency parsers capable of incremental parsing. MaltParser (Nivre et al. 2007) and jwcdg[5], which work in fundamentally different ways. A short comparison of both parsers with respect to the topic of this thesis can be found in Table 1.2.

1.6.1. MaltParser

MaltParser is a shift-reduce parser. It uses an interchangeable parsing algorithm which defines a set of possible parsing actions. A component trained on an annotated corpus then decides which action to conduct based on features of the parser state. This is repeated until the sentence has been completely parsed. MaltParser works monotonically because edges already constructed cannot be changed later on. Since MaltParser is a monotonic parser, it needs lookahead to yield a high accuracy (Beuck, Köhn, and Menzel 2011b). MaltParser will be described in more detail in Chapter 5.

The way MaltParser works makes it impossible to do structural prediction without creating a new parsing algorithm because MaltParser can only incorporate existing words into its analyses. The only information obtainable from MaltParser regarding prediction is whether a word will be attached to an upcoming word or to a word that is already available (minimal prediction). Since using lookahead and the lack of structural prediction is inappropriate for the scenarios described in Section 1.4, this thesis will focus on jwcdg.

1.6.2. jwcdg

jwcdg is a parser that implements the Weighted Constraint Dependency Grammar (WCDG) formalism (Schröder 2002). It utilizes a transformation based algorithm called frobbing (Foth 1999). As such it uses reanalysis which makes the output non-monotonic but makes using lookahead unnecessary. jwcdg has been extended by Beuck forthcoming to be able to produce structural prediction.

[5] `https://nats-www.informatik.uni-hamburg.de/view/CDG/WebHome` (Description), `https://gitorious.org/jwcdg` (Source Code)

	MaltParser	jwcdg
Mode of incremental output	monotonic	non-monotonic
Quality of prediction	minimal prediction	structural prediction
accuracy can be traded of for	lookahead	processing time

Table 1.2.: Characteristics of MaltParser and jwcdg

1.7. Structure of This Thesis

In this thesis we will first take a look at how jwcdg works (Chapter 2). Then jwcdg will be adapted to be truly anytime capable (resulting in jwcdg-0) and evaluated in Chapter 3. After that possibilities for optimizing (jwcdg-1) and parallelizing jwcdg (jwcdg-2) will be investigated and evaluated in Chapter 4. In Chapter 5 MaltParser will be incorporated into jwcdg so that the results of MaltParser can be used to help jwcdg (resulting in jwcdg-3 and jwcdg-4). In Chapter 6 the impact of sentence length will be evaluated for the different versions of jwcdg as well as how well they cope with sentences from language learners.

2. jwcdg

In the WCDG formalism a set of weighted constraints has to be defined which forms the grammar. The grammar is used to determine how good an analysis is. A constraint is a function that maps edges to a penalty and the information if the constraint is violated by the edges:

$$Constraint : \{e_1, e_2 \dots e_n\} \rightarrow [0, 1] \times \{true, false\}$$

jwcdg only uses constraints which operate on up to two edges. A *conflict* is a tuple $(edge1, edge2, constraint, penalty)$ where the edges violate the constraint, yielding the penalty. In the case of a constraint operating on one edge, the conflict contains only one edge.

The *score* of an analysis denotes how good it is. To compute the score of an analysis a, all constraints of the grammar are evaluated on all possible edge combinations, which results in a set of conflicts. The score of a is the product of the penalties of the conflicts of a. Since the penalty of a conflict lies in the range [0..1] and the score of an analysis is the product penalties, it is clear that every score lies in the range [0..1]. The more severe the violation of a constraint should be, the smaller the penalty has to be. If a constraint has a penalty of 0, an analysis violating it will have a score of 0, regardless of other conflicts. Such constraints are called *hard* constraints. The best analysis is defined as the one that has the highest score:

$$best\ analysis = \arg\max_{a} \prod_{conflict \in Conflicts(a)} penalty(c)$$

WCDG itself formulates a definition of the best analysis, but not the way to actually find it. Generating all possible analyses is not an option since the number of analyses grows exponentially with the number of words. Even if edge labels and word readings are ignored, there are n^{n-1} possible analyses for a sentence with n words.

Different approaches have been used to find good analyses in the past, which either work by eliminating edges until one edge per domain is left(Menzel and Schröder 1998b) or by repeatedly transforming an analysis until it is deemed good enough.

2.1. The Frobbing Algorithm

To recap: An analysis consists of exactly one edge from every domain. A domain is the set of possible edges having the same word as a dependent. The domain usually has several edges using the same regent since multiple labels are possible. The possible regents are the other words as well as the special NIL node. If an analysis should have no edge in a domain, it technically has the edge to NIL with the empty label in that domain.

Foth (1999) introduced a transformation based algorithm called *frobbing*. It works by generating an initial analysis and then repeatedly transforming it. In contrast to eliminative approaches, this guarantees that the parser can be stopped at any time and will always yield a solution when stopped because at each step of the algorithm there is a valid analysis at hand (in the sense that for every domain exactly one edge is selected). This property is called the *anytime property*.

Only unary and binary constraints are used in jwcdg. They can be context sensitive or not so in total there are four different groups of constraints. These groups are depicted in Figure 2.1. Context sensitive constraints will be addressed in Section 2.1.3. Until then it is assumed that all constraints are not context sensitive.

Unary constraints operate on a single edge

Binary constraints operate on two edges

Context sensitive constraints incorporate information from additional edges of the analysis

Context insensitive constraints only operate on the given edges

A simplified description of frobbing is depicted in Algorithm 1. Since frobbing works by repeatedly transforming an analysis, first an initial analyses needs to be constructed (line 2). The initial analysis is obtained by using those edges that have the highest score after evaluating all unary constraints that are not context sensitive. Then the analysis is judged (all constraints are evaluated on the analysis) to get the list of conflicts. The most severe conflict that is not marked as unsolvable is then attacked (lines 6 and 7; `attackConflict` will be described in the next section). If `attackConflict` succeeds, it results in an analysis that does neither contain the conflict `c` nor conflicts that have been solved before. If the result is better than the currently best analysis, it is stored as the best (line 9) and frobbing starts with this new best analysis again. Otherwise frobbing tries to resolve the next conflict.

```
   Data: sentence S
   Result: Analysis of S
 1 List removedConflicts ← [ ];
 2 Analysis best ← makeInitialAnalysis(S);
 3 Analysis current ← best;
   // we need to know from which conflict we have started
 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() ;              // resets internal state of attackConflict
   |   |   // store with which conflict we'll start the next iteration
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`

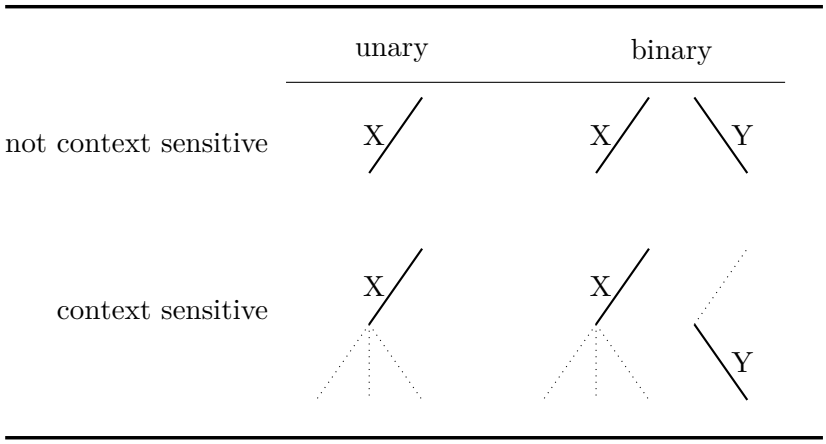


Figure 2.1.: Different types of constraints

If `attackConflict` returns no analysis (line 12), the conflict from which frobbing started (`initalConflict`) is marked as unresolvable since frobbing could not find a path of analyses that lead to a better analysis than the one it started with. Frobbing then starts again with the next solvable conflict. This loop is repeated until no solvable conflict remains.

2.1.1. Attacking Conflicts

The majority of the work that is done by the frobbing algorithm happens inside `attackConflict` (see Algorithm 2). `attackConflict` tries to transform an analysis to another analysis that does not have a given conflict anymore. It does so by generating patches that transform the current analysis to another analysis. A patch is a set of edges $\{e_1, \ldots, e_n\}$ from domains $\{d_1, \ldots, d_n\}$ that should be used instead of the edges that are currently selected in the domains $\{d_1, \ldots, d_n\}$. The patches that are generated (line 2) only change the domains in which edges lie that are part of the conflict since this is the only possible way to resolve a conflict[1]. Because the quality of a patch cannot be computed before it is actually applied to an analysis, all such patches need to be generated. The patches are then judged i. e. they are applied to the analyses and all constraints are evaluated on the resulting analyses. Patches that reintroduce conflicts from `removedConflicts` are discarded (line 6). The best patch from all the possible patches that do not reintroduce an already

[1] This only holds for constraints that are not context sensitive, see Section 2.1.3.

```
   Data: Conflict c, Analysis current, List removedConflicts
   Result: Analysis not violating c
 1 removedConflicts + = c;
 2 patches ← generatePatches(c);
 3 bestPatch ← null;
 4 foreach patch in patches do
 5 |   patch.judge();
 6 |   if detectCircle(patch, current, removedConflicts) then
 7 |   |   continue;            // ignore patches that reintroduce conflicts
 8 |   end
 9 |   if patch > bestPatch then
10 |   |   bestPatch ← patch;
11 |   end
12 end
13 if bestPatch == null then
14 |   return null;
15 end
16 Analysis newA ← bestPatch.apply(current);
17 return newA;
```

Algorithm 2: `attackConflict`

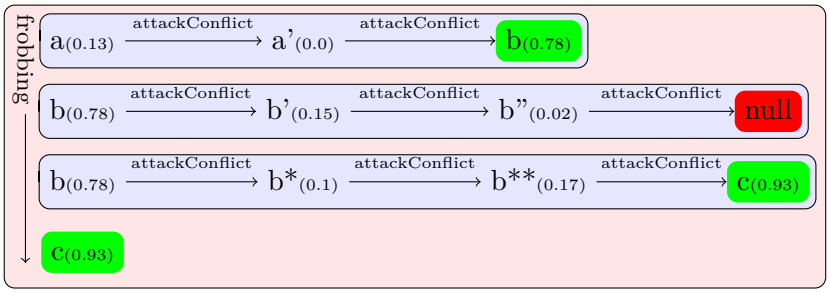


Figure 2.2.: graphical representation of a frobbing run, scores in brackets

removed conflict is then applied to the analysis and the resulting analysis is returned. If there is no such patch, **null** is returned instead to indicate that `c` could not be removed without reintroducing conflicts. Note that the resulting analysis does not need to have a better score than the original one, it is only guaranteed that it does not contain a conflict from `removedConflicts`.

2.1.2. An Exemplary Frobbing Run

All in all the algorithm repeatedly produces a chain of analyses that either result in a better analysis or the insight that the conflict that has been attacked cannot be removed. This is depicted in Figure 2.2, where frobbing transforms an analysis a via b to c via intermediate analyses with a low score. In the last step of the second attempt, no patch can be produced and therefore **null** is returned. Frobbing then tries to solve the next conflict, starting again with b. After c has been produced, no solvable conflict remains and the algorithm terminates.

2.1.3. Context Sensitive Constraints

Foth (2006) and McCrae, Foth, and Menzel (2008) introduced context sensitive constraints into the WCDG formalism. These constraints operate not only on the edges on which they are evaluated but also on additional edges of the analysis. This is mostly done with predicates such as *has(edge, label)*, which "returns TRUE if the [edge] has a child edge with the specified label"[2], which allow for existence qualified formulas. Context sensitive constraints make constructs possible that required relatively complex constructs with additional edges that were not part of the analysis before. They, however, also require special treatment: The result from evaluating

[2] copied from jwcdg source code comment

a context insensitive constraint on a set of edges, say $\{e_1, e_2\}$, can be cached for later use when evaluating other analyses that also contain the edges $\{e_1, e_2\}$. This is impossible for context sensitive constraints, because every change in an analysis could possibly alter the evaluation result of c on $\{e_1, e_2\}$. Therefore, jwcdg only caches the results from constraints that are not context sensitive while context sensitive constraints need to be reevaluated every time. The approach of attacking a conflict of a context sensitive constraint also needs to be adapted from the one described in Section 2.1.1. Since the result of a constraint evaluation does not only depend on the edges the constraint is evaluated on, changing the domains a conflict's edges is no more the only way to resolve the conflict. The following (fictitious) constraint which only checks whether the edge has a child edge that is labelled with "DET" exemplifies this:

X:SYN:"All edges need a DET child": :0.9: has(X,"DET")

If an edge e violates this constraint, the constraint will be violated in every analysis that results from replacing e with another edge from the same domain. The only possible way to produce an analysis that does not violate the constraint is to replace the child node of e that has the label "DET". Because the evaluation result of a context sensitive constraint depends on additional edges, the strategy described in Section 2.1.1 (replacing the edges on which the constraint was evaluated) is not a good strategy for context sensitive constraints. Therefore jwcdg instead creates all patches that change a single edge, regardless of whether it is one of the edges from the conflict or not. The reasoning behind this approach is that most context sensitive constraints encode the assertion of a (non-)existence of a single edge and thus changing the value of a single domain will either add a missing edge or remove an offending edge.

2.2. Predictive Incremental Parsing in jwcdg

The frobbing algorithm itself has no concept of incrementality. It can, however, be used in an incremental algorithm. To parse a sentence incrementally, frobbing is used to create an analysis of the first increment (the prefix of the sentence containing only the first word), then the analysis is extended by attaching the next word to the analysis and frobbing is run again. This is repeated until all words of the sentence are included in the analysis. Beuck (forthcoming) has extended jwcdg to produce structurally predictive analyses when parsing incrementally: A fixed set of virtual nodes is added to the analysis. The frobbing algorithm treats them in the same

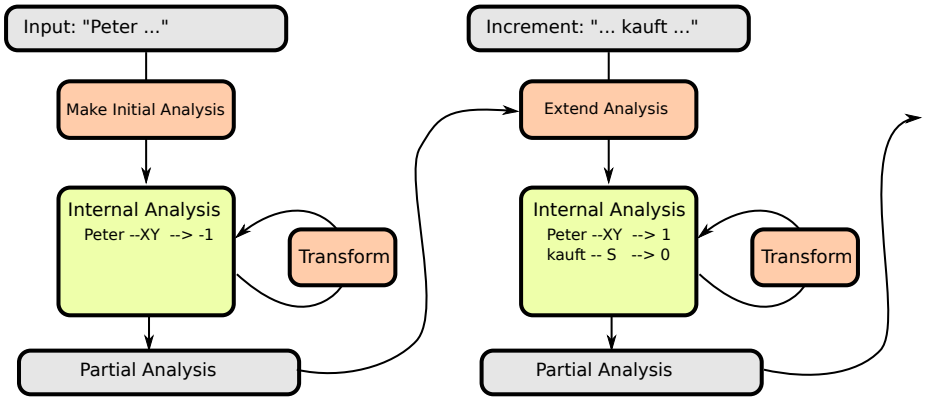


Figure 2.3.: Incremental Parsing with jwcdg, source: Beuck and Köhn 2012

way as all other words of the analysis. Additional constraints are used that penalize edges to and from virtual nodes. If no other word is attached to a virtual node and it is attached to NIL with the empty label (which is the only attachment that is not penalized), the virtual node is assumed not to be a part of the analysis. Since there is a small penalty of including virtual nodes into an analysis, this is only done by frobbing if it solves another conflict. The process of incrementally parsing a sentence is depicted in Figure 2.3.

2.2.1. Choosing a Set of Virtual Nodes

Virtual nodes are under- but not completeley unspecified. A virtual noun for example has the category “noun” but its numerus is unspecified so it can serve as a replacement for both singular and plural nouns. This generalization is necessary because otherwise the set of virtual nodes would need to be considerably larger to cover the same

prediction possibilities. However, the computational complexity grows with the number of provided virtual nodes and because of this a small number of general virtual nodes is used instead. All experiments in this thesis use one virtual verb and two virtual nouns. Most of the structure found in prefixes of German sentences can be predicted with this set of virtual nodes(Beuck forthcoming).

2.2.2. Virtual Node Instantiation

When a new word is added to an analysis that has already been predicted by a virtual node, all domains that touched that virtual node before need to be changed so that the word replaces the virtual node in order to get a correct analysis. Since this replacement could need several steps, a special algorithm to instantiate a virtual node (i. e. replace it with a concrete word) has been implemented in Beuck (ibid.). When extending an analysis with a new word, the possible instantiations along with the resulting analyses' scores are computed. The instantiations are the possible replacements of a virtual node by a lexical reading of the new word for every virtual node that is included in the current analysis.

To prevent the parses from doing an instantiation that is not fitting, a threshold ($score(a) \cdot t; t \in [0, 1]$) is used which makes sure that only instantiation with a high enough score are considered. The best instantiation is used If its score exceeds the threshold. If no such instantiation can be found, the new word is simply added to the analysis by using the edge from its domain which has the least penalty after evaluating the context insensitive unary constraints on them.

3. Anytime-capable jwcdg

Although the frobbing algorithm is an anytime algorithm in principle, the implementation in jwcdg does not allow for interruption. Instead, a time limit can be provided for parsing an increment beforehand. The processing time already used is regularly checked against this limit. The purpose of the time limit is mainly to make sure that the algorithm terminates in acceptable time for example when evaluating the parser. It has not been implemented with interactive systems in mind. This approach has several drawbacks:

- Sometimes more than ten seconds elapse between two checks of the time limit.
- The point in time of the interruption has to be set in advance.

Those two points prevent jwcdg from being used in a setting where quick reactions are needed and the parser does not know in advance when the next word will be available. This chapter deals with the adaptions that have to be made to make the theoretical anytime property usable in practice. The resulting version will then be evaluated with respect to its utilizability in interactive settings.

3.1. An Asynchronous Interface to jwcdg

To construct an interface that can be used asynchronously and which provides measures to quickly interrupt jwcdg, its class structure needed to be adapted. A new class `IncrementalStepFuture` implementing the `Future` interface has been implemented. A call to the incremental frobber now instantly returns an instance of this class instead of computing a result and then returning it. The `Future` interface provides means to encapsulate the computation of a value by serving as a placeholder for it while it is potentially not yet available. It provides methods for checking the state of the computation (`isDone`, `isCancelled`), to cancel the computation (`cancel`) and to get the value, possibly waiting until it is available (`get`). `IncrementalStepFuture` also provides the method `pleaseFinish` to kindly ask the frobbing algorithm to stop processing and register a result in the `IncrementalStepFuture` object. This method

sets a flag that is checked often so that the delay between invoking `pleaseFinish` and the availability of the result is in the order of few milliseconds.

This change transforms the parser from being pull-based (fetching the next word when the previous computation has finished) to being push-based (the next word can be incorporated as soon as it is available).

3.2. Virtual Node Instantiation

The virtual node instantiation, which happens before frobbing starts, does not check for the time limit at all. While it usually only takes milliseconds, sometimes (especially in long sentences) the instantiation can take several seconds in which jwcdg is not interruptible. This has been changed so that jwcdg can also be interrupted while doing the instantiation. If this happens, the instantiation is assumed to have failed and the new word will instead be attached using the best edge as described in Section 2.2.2 and no frobbing takes place.

Because all time that is being spent in (possibly unsuccessful) virtual node instantiation cannot be spent frobbing, excessive running times for it are undesirable. To prevent the virtual node instantiation step from running until the user interrupts, a configurable time limit has been introduced that should be set to a value that is considerably smaller than the expected time until the next interrupt. This way it is guaranteed that there is still time left for frobbing.

Sometimes the virtual node instantiation can even be skipped altogether: If the current analysis violates hard constraints (i. e. the score of the analysis is zero) the algorithm cannot distinguish between a good and a bad instantiation because the score of all analyses that result from instantiating a virtual node will also have a score of zero. This is the case because merely replacing a virtual node with a normal word will not solve a conflict. Since the scores for the resulting analyses will be the same as the score of the original analysis, all replacements would be considered to be good enough, leading to a large number of possible instantiations that need to be considered as being the best. The algorithm has no way of deciding which instantiation to use and therefore picks any of them, which is most likely an unfitting one. All in all this means that if the current analysis violates a hard constraint, the instantiation algorithm will most probably need a long time and then apply a nearly random instantiation. This time is better spent in the frobbing algorithm and therefore the virtual node instantiation algorithm has been changed to skip instantiation altogether if the current analysis violates a hard constraint. Throughout this thesis a time limit of fifty milliseconds will be used for virtual node instantiation.

3.3. Reclaiming the Time Used by the Tagger

jwcdg can incorporate external information sources (also called *predictors*) such as a tagger[1]. In this thesis, the TnT tagger (Brants 2000) will be used as a predictor, as described in Hagenström and Foth (2002). TnT has been proved to be the best tagger for jwcdg, so using another tagger would have degraded the parsing accuracy. TnT tags more than 600,000 tokens per second, making the time needed for tagging negligible in theory. However, the tagger needs to be restarted for every tagging run for technical reasons concerning the process input and output. The problem could not be further analyzed due to the fact that the TnT source code is not available. Starting TnT takes about half a second. Before frobbing takes place, jwcdg spends this amount of time waiting for TnT because the tagger needs to be rerun every time an analysis is extended. To circumvent this problem, an asynchronous worker has been introduced that starts tagger processes in advance so that there is always at least one tagger process that is ready for tagging. This way, the parser does not need to wait for the tagger and can do work half a second more.

3.4. The Baseline Parser

With all this in place, we now have a parser that can be interrupted at any time and delivers a result at most a few milliseconds after the interruption. The anytime property does, however, not cover the time needed for extending the analysis, which has to happen when a new word is incorporated. This variant will be referred to as *jwcdg-0*.

3.4.1. Evaluating Predictive Parsers

Unless noted otherwise, all evaluation in this thesis have been carried out on the sentences 18603 to 19602 of the NEGRA corpus(Brants, Skut, and Uszkoreit 2003). All experiments of this thesis were conducted on a 48 core machine with four AMD Opteron 6168 processors.

Evaluating the results of incremental parsing is not straightforward. In non-incremental parsing the labeled and unlabeled attachment score $\frac{c}{a}$ where c is the number of correct attachments (ignoring the label in the unlabeled case) and a is the number of attachments that had to be made is used. If one would just use this score over all increments to evaluate an incremental parser, the attachment of words

[1] A tagger is a program that assigns word classes to words (it "tags" the words with classes).

that are at the beginning of a sentence would get a bigger weight than words at the end of a sentence since they are a part of more increments. In addition to that, an attachment score requires the number of attachments made to be fixed. This is, however, not the case when the parser may be predicting words. Also, the dynamics over time cannot be captured by a single score. A parser using reanalysis could, for example, wrongly attach every word that is new and reattach it to another word when the next word arrives.

Beuck, Köhn, and Menzel (2011b) have therefore proposed a method to evaluate predictive incremental parsers that tries to account for the requirements outlined above. Instead of providing a single score for evaluation, a set of results is used to capture the dynamics over time by grouping them in slots. Each slot x represents the accuracy of the attachments of the words that are x words to the left of the newest word. The score for the slot 0 is therefore determined using the newest word of each increment, i. e. how good the parser attaches the word it just got. The score for slot 1 is determined using the word that is one word to the left to the most recent word from each increment and so on.

If the parser incorporates virtual nodes into its analyses, every analysis a can be mapped to the corresponding gold standard analysis (*gsa*) with a mapping ϕ. ϕ maps words to their corresponding words in the gsa and virtual nodes to words of the gsa that are not part of a (i. e. they lie in the future). Virtual nodes do not need to be mapped. No two virtual nodes may be mapped to the same word and vice-versa. For each increment the mapping that yields the best attachment score is used for the evaluation. With these mappings at hand, it can be computed how often the word that will come next has been predicted by counting how often the word to the right of the newest word has been mapped to a virtual node. According to the slot numbering this is the slot -1 (the word to the right of the newest word). This can also be done for words further to the right.

A word is counted as *correct* if it is correctly attached to another word (ignoring the edge label for unlabeled evaluation). If the attachment is wrong, it is counted as *wrong*. If a word w is attached to a virtual node vn that is mapped to the regent of w in the gold standard ($regent_{gold}(\phi(w)) = \phi(vn)$), w is counted as *correct structural prediction*. Otherwise it is counted as *wrong structural prediction*. For negative-numbered slots a virtual node vn is counted as *correct structural prediction* if $regent_{gold}(\phi(vn)) = \phi(regent(w))$ (the regent of the word vn is mapped to is the mapped word of the regent of vn). Otherwise it is counted as *wrong structural prediction*. If the evaluation should include the edge labels (which is the case in this thesis unless stated otherwise), the label of the edge needs to be the same as the one in the gold standard to be classified as correct structural prediction, otherwise it will

be classified as wrong structural prediction.

The prediction precision (pred-prec) is used to account for virtual nodes that could not be mapped to words in the analysis. For this purpose, the virtual nodes are grouped into three categories: unmappable (no corresponding word could be found in the gold standard), wrong structural prediction and correct structural prediction. To get a picture of how good the prediction precision is, the number of unmappable virtual nodes needs to be compared against the number of virtual nodes that could be mapped (and are therefore in the categories correct structural prediction and wrong structural prediction).

In addition, the accuracy of the final increments (i. e. the increments containing whole sentences) can be measured using the attachment score, resulting in the *final accuracy*.

The diagrams depicting all this information can be seen in Figure 3.1. To determine how stable the attachments are, their stability with respect to the final result can be used. This information is plotted as a line over the slots. It shows the percentage of attachments at this point that are also in the analysis of the final sentence. The higher the stability, the more attachments will still be in the final analysis and the less changes to already made attachments are to be expected.

3.4.2. Evaluation of the Baseline Parser

When running jwcdg-0 (the baseline version of jwcdg) with a time limit of one second per increment and with a time limit of eight seconds per increment, we get the result depicted in Figure 3.1. One can see that with a time limit of one second the accuracy is extremely low and that the number of correct attachments does not increase much from the initial attachment (slot 0) to the final attachment. The results for the eight second time limit look different: The percentage of correct attachments grows from slot to slot and the difference between the accuracy when a word is first seen (slot 0) and the accuracy for whole sentences is ten percentage points whereas it is only 2.5 with a time limit of one second. This shows that with a tight time limit jwcdg has not enough time to fix errors that have previously been made.

Influence of the Time limit on the Results

The percentage of correct attachments over the processing time available per increment is depicted in Table 3.1. The benefit of additional time decreases as more time is already available. The difference between the accuracy of the first attachment and the accuracy of the final analysis grow with the processing time that is available

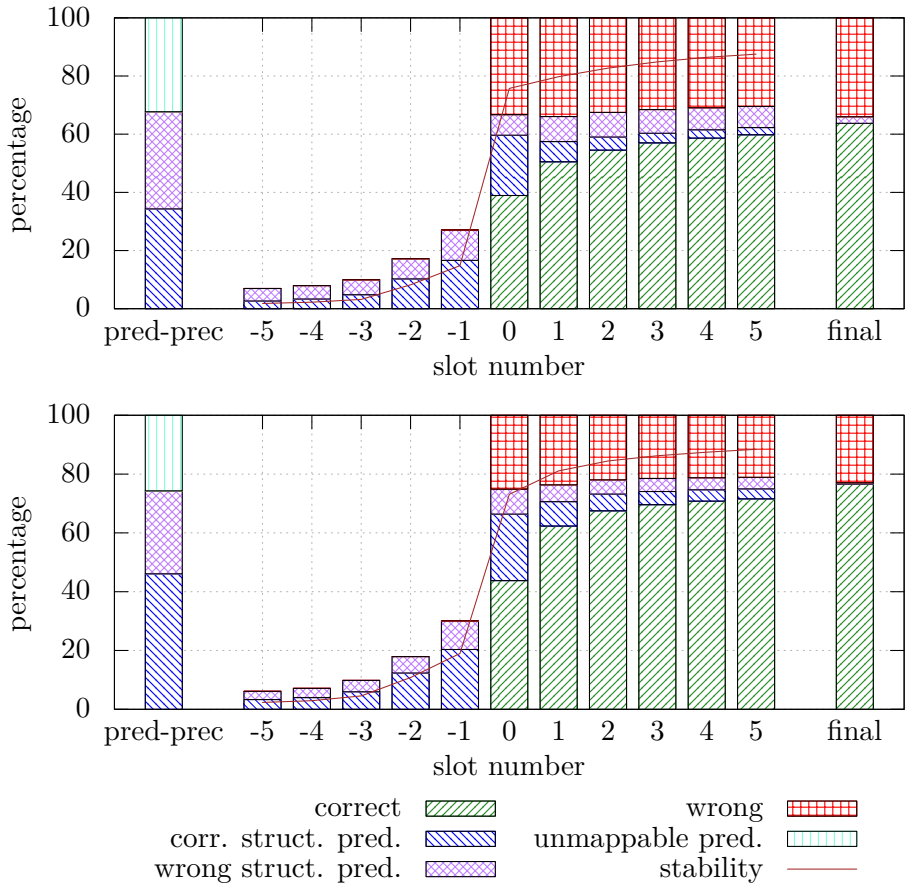


Figure 3.1.: Results for jwcdg-0 with a time limit of one (top) and eight (bottom) seconds

	time limit			
	1 sec	2 sec	4 sec	8 sec
initial attachment	56.01%	61.27%	64.44%	66.39%
final accuracy	58.48%	67.22%	72.58%	76.60%

Table 3.1.: Effect of the time limit to the baseline jwcdg

to jwcdg. This shows that the higher the time limit is the better jwcdg is able to repair wrong first attachments. Seen the other way round it means that a good first attachment is important for the final accuracy if the time limit is short.

Time Needed for Initiation

The time needed for initiating the next run (running predictors, creating all new possible edges etc.) can bee seen in Figure 3.2. Obviously the initiation takes longer if more words are already present. There are only seven increments that are longer than 60 words and they are from only two sentences with length 61 and 66 each. The last column for the increments 61 to 70 has therefore very little support. There are 48 increments with length 51 to 60. We can observe that jwcdg often takes more than one second initiating the next frobbing run even with moderately sized increments. In those cases no time remains at all to actually change the analysis which explains the low difference between the accuracy of the initial attachments and the accuracy of the final analyses. Since the initiation cannot be interrupted, large initiation times lead to a time delay in processing the incoming words. In this thesis the time available for subsequent increments was unchanged but in a real-time scenario the processing of the subsequent would need to be shortened to catch up again.

Summary

To sum it up, we can see that a time limit of at least eight seconds is needed to get a final result with a labeled accuracy of more than 75 percent. The initiation time of big increments takes such a long time that with a time limit of less than two seconds nearly no frobbing can be done at all. The outliers make enforcing a small limit impossible, even for short sentences.

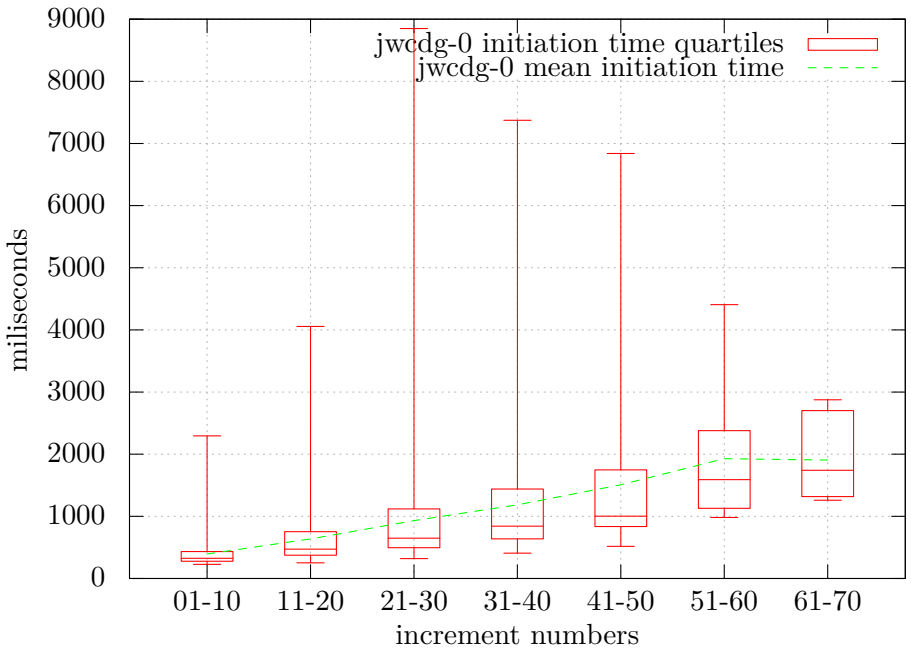


Figure 3.2.: Initiation time over increment number; boxes denote the first, second and third quartile, whiskers denote maximum and minimum)

4. Optimizing the Speed of jwcdg

The last chapter showed that jwcdg-0 is unable to parse with tight time limits and that its results decline considerably when reducing the time limit. In an interactive scenario this is not acceptable. In this chapter we will explore how jwcdg can be optimized to yield acceptable results in less time.

4.1. Constraint Argument Optimization

To pinpoint possible optimizations the parser has been analyzed using a profiler. The profiling results showed that about 20% of the execution time was spent in HashMap.get(). Further inspection showed that nearly all of these calls were made while evaluating constraints when a constraint needed to get the binding of a variable. These bindings where handed down using a hash map: The variable as the key and an edge as the value.

In jwcdg the formula of a constraint is parsed into a tree of objects which each stand for an operation such as *and* or *not*. These objects all have a method called `evaluate` which calls `evaluate` on its child nodes if necessary. An *and* formula, for example, calls `evaluate` on each subformula until one of them returns false. If all subformulas return true, *and* returns true as well. In each call to `evaluate` the variable binding is handed down so subformulas which need to operate on the edge that is bound to a variable can access that binding. For example, the subformula `X@word` will evaluate to the word that is the dependent of the edge X, for which the edge has to be fetched from the variable binding.

4.1.1. Changing the Binding Representation

The sampling has shown that about 97% of all HashMap.get() operations occurred when extracting the value of a variable while evaluating constraints. In addition, about 6% of the entire processing time was used creating variable bindings.

Since over 25% of the processing time was used just dealing with variable bindings, this seemed to be a good opportunity for optimization. For the variable bindings HashMaps have therefore been replaced with Lists. The lookup is now based on the

```
{X\SYN\Y/SYN} : Dreierzyklus : proj : 0.0 :
  Y^from < X@from
  ->
  parent(Y^id) != X@to;
```

Figure 4.1.: A constraint used in jwcdg

		time limit			
	parser	1 sec	2 sec	4 sec	8 sec
initial attachment	jwcdg-0	56.01%	61.27%	64.44%	66.39%
	jwcdg-1	58.33%	62.80%	65.59%	67.03%
final analysis	jwcdg-0	58.48%	67.22%	72.58%	76.60%
	jwcdg-1	62.05%	69.46%	74.46%	77.95%

Table 4.1.: Effect of the timelimit to jwcdg-1

index of the variable in the signature. When the constraint depicted in Figure 4.1 is parsed, every subformula accessing a variable stores the index of that variable. For example, the subformula `Y^from` stores that it uses the second variable while `X@from` stores that it uses the first variable. This index based lookup is much faster and the creation of a two-element list is faster than creating a HashMap containing two key-value pairs.

4.1.2. Effects of the Optimization

The jwcdg variant resulting from the optimizations described above will be called jwcdg-1. Now that the parser has been optimized, it is time to have a look at the parsing accuracy. Figure 4.2 shows the evaluation of jwcdg-1 with a time limit of one second and a time limit of eight seconds. Clearly the time limit still has a big impact on the accuracy.

Table 4.1 shows the accuracy (correct attachments to virtual nodes and words) of slot 0 and of the final analyses over the time jwcdg is given per increment. The results of jwcdg-0 are shown as well for comparison. With a time limit of one second jwcdg-1 clearly outperforms jwcdg-0. The initiation times are, however, only slightly lower than the ones of jwcdg-0 (see Figure 4.3).

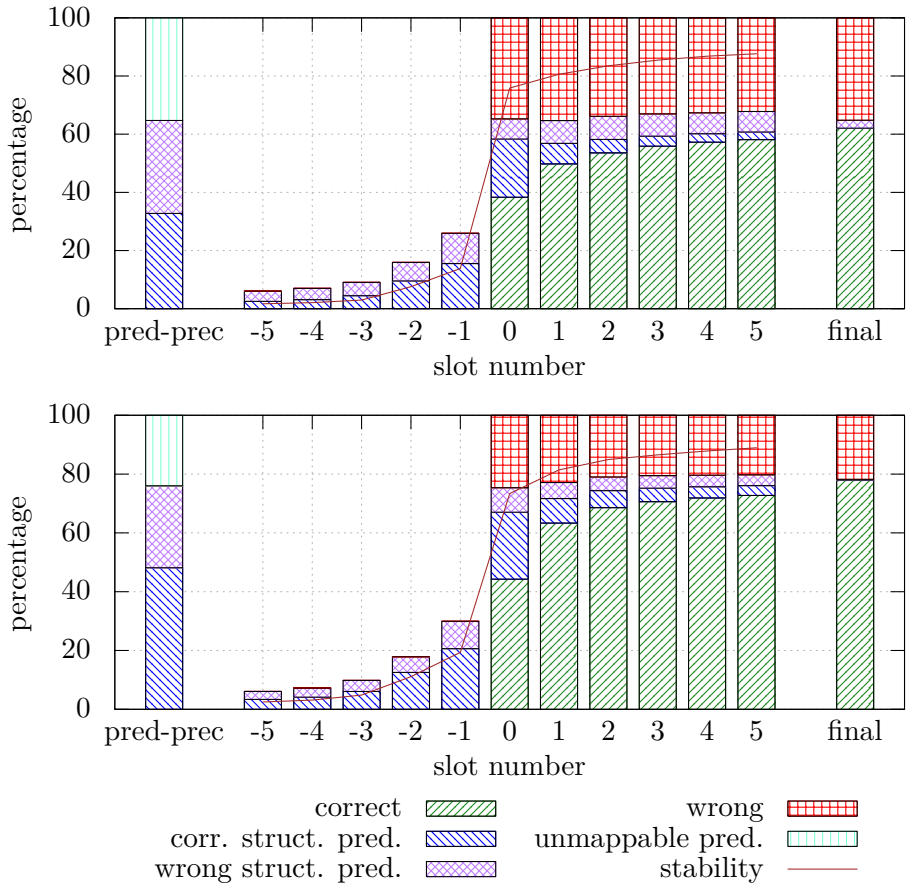


Figure 4.2.: Results for jwcdg-1 with a time limit of one second (above) and eight seconds(below)

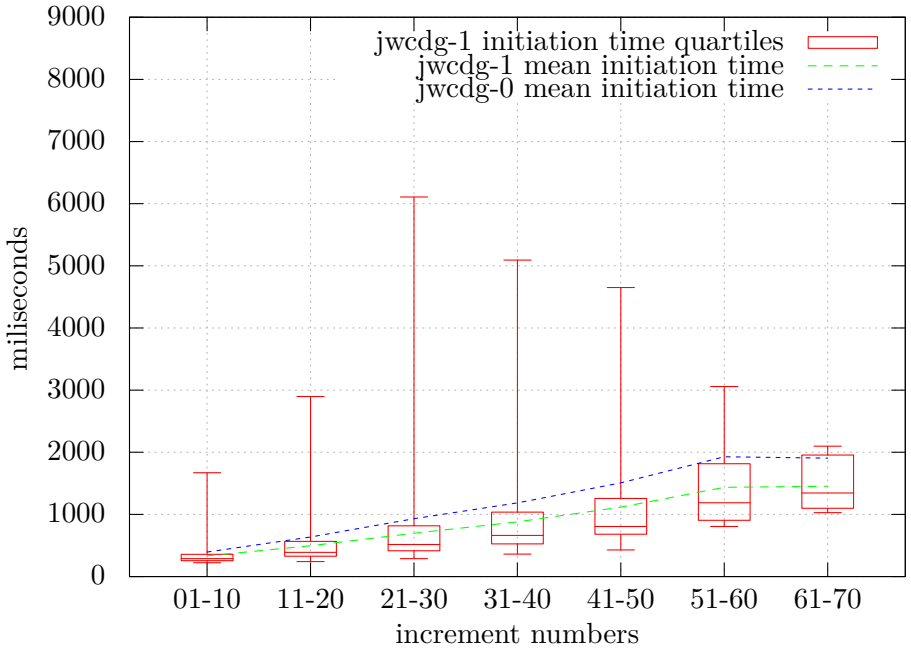


Figure 4.3.: Initiation times after optimization (jwcdg-1)

4.2. Parallelization

To further speed up jwcdg, this section explores possible approaches of parallelizing its methods, leading to the jwcdg variant jwcdg-2. When evaluating an analysis, the binary constraints need to be evaluated on each pair of edges. This is done in the method `evalBinary`, which receives two edges and evaluates the binary constraints on them. `evalUnary` does the same for single edges and unary constraints. In each call to `evalBinary` the method iterates over all applicable constraints and evaluates them on the two given edges. The result is a list of conflicts. Since the evaluation of the constraints is independent and a list of them has to be processed, `evalBinary` seems to be a good candidate for parallelization.

4.2.1. Parallelizing evalBinary

For this experiment `evalBinary` has been rewritten. The code that evaluates a constraint has been extracted into a new worker class and `evalBinary` creates a set of workers that are executed in parallel. The threads which runs these workers are created once on startup so no new threads need to be created when calling `evalBinary`. A `Supplier` encapsulates the list of constraints and provides an interface to get a constraint, making sure that no constraint is handed out twice. The workers then work in a loop in which they fetch a constraint and evaluate it using the given edges. If the constraint is violated the resulting conflict is added to the result list. The workers terminate once the `Supplier` has no constraints to evaluate anymore. No synchronization is needed because all the operations that operate on shared data structures are atomic (fetching constraints and appending to the result list).

An evaluation of the speed of the parallelized `evalBinary` shows that it is unfortunately much slower than jwcdg-1. The parallelization cannot outweigh the overhead it produces. This is not surprising since the invocation time of a single `evalBinary` run is fairly short (about 0.1 milliseconds). Parallelizing `evalBinary` and `evalUnary` has therefore been abandoned.

4.2.2. Parallelizing optimizeDomains

Since `evalBinary` is too small to be parallelized, let us see whether we can find a method that is more high-level and can be parallelized. In fact, we have already seen a method that works on a list of elements: `attackConflict`. It generates a list of patches and then evaluates the analyses that result from applying them. The evaluation (lines 4 to 10 in Algorithm 2) takes place in the method `optimizeDomains`

which gets a list of patches and returns the best of them. This method has been rewritten so that the evaluation of the patches is done in parallel. The structure of this rewrite is similar to the one used in the last section: A set of workers is started which each repeatedly fetch a patch from a patch supplier and then evaluate that patch. After each evaluation the worker checks if another thread has found a new global optimum or the interruption flag has been set. In both cases the worker stops immediately. In contrast to the parallelization of `evalBinary`, synchronized code blocks are needed because multiple variables need to be set when a new best patch has been found.

4.2.3. Parallelizing buildIter

Every time jwcdg starts to parse a new increment, it has to integrate the new word before frobbing can start. The possible edges from the new word to all other words and from all other words to the new word need to be generated. In jwcdg-1, this is done sequentially.

To parallelize the creation of new edges, a `Callable` is created for each old word that will create all possible edges between the old word and the new one when called. All the callables are then handed over to an executor which runs the callables in parallel. This means that the number of threads used is bound by the number of other words which may well be below the number of available processors.

4.2.4. Scalability of optimizeDomains and buildIter

To evaluate whether the parallelization of `optimizeDomains` and `buildIter` really leads to a speedup and how the execution time scales with the number of threads used, jwcdg-2 has been run on the sentences 501 to 600 of the NEGRA corpus. A time limit of 16 seconds has been used.

The results (see Tables 4.2 and 4.3) show that the parallelized methods run indeed faster if at least two threads can be used. As stated in Chapter 3, jwcdg can be interrupted. The results that are reported in this section are only the timings of invocations of `optimizeDomains` that have not been interrupted. The number of invocations of `optimizeDomains` differs across the number of threads used because a time limit has been used. The distribution of the invocation times for the 48 core run can be seen in Figure 4.4.

`buildIter` does not seem to be able to take much advantage of more than sixteen cores. Since the average increment length is about ten words, there are on average only 20 jobs that can be computed in parallel.

	number of threads used							
	np	1	2	4	8	16	32	48
Min.	0	0	0	0	0	0	0	0
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 4.2.: Timing information in ms of `optimizeDomains` with a time limit of 16 seconds for different thread numbers; np = not parallelized

	number of threads used							
	np	1	2	4	8	16	32	48
Min.	8	8	4	3	2	2	2	2
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 4.3.: timing information in ms of `buildIter` with a time limit of 16 seconds for different thread numbers; np = not parallelized

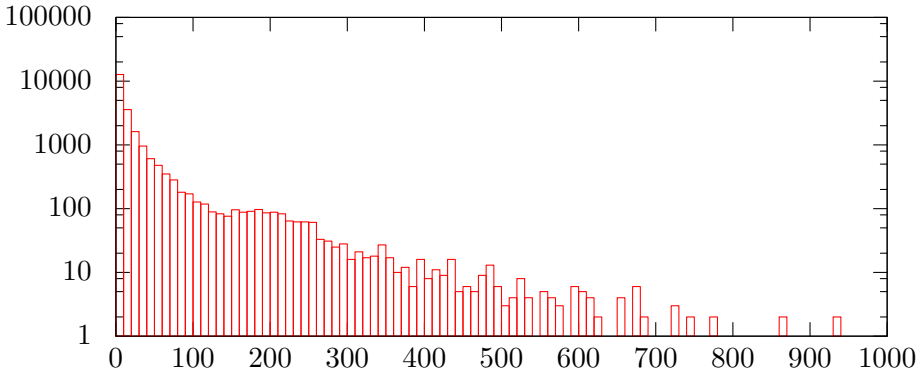


Figure 4.4.: Distribution of invocation times for optimizeDomains using 48 cores

All in all, we can see that although both `optimizeDomains` and `buildIter` do not scale proportionately to the number of threads, they clearly benefit from parallelization. Using two threads is already faster than the unparallelized version. The best result has been achieved using 32 threads so the optimum number of threads seems to lie between 16 and 48 threads.

4.2.5. Evaluation of jwcdg-2

Now that the parser has been further optimized, it is time to have a look at the parsing accuracy. Figure 4.5 shows the evaluation results of jwcdg with a time limit of one second and a time limit of eight seconds. Clearly the time limit still has a big impact on the accuracy.

Table 4.4 shows the accuracy (correct attachments to virtual nodes and words) of the initial attachment and of the final analyses over the time jwcdg is allowed to use per increment. The results for jwcdg-0 and jwcdg-1 are shown as well for comparison. jwcdg-2 has a considerably better accuracy than jwcdg-0 and jwcdg-1. The differences between the initial attachment and the final accuracy are higher as well.

About 29% of the predictions made by jwcdg-2 are classified as wrong structural prediction (meaning that a virtual node could be mapped to a word but is incorrectly attached). This is mostly due to wrong labels (which are not always easy to predict). Figure 4.6 shows results for jwcdg-2 obtained from an unlabeled evaluation. The percentage of attachments that are counted as wrong structural prediction drops to less than 14%.

Figure 4.7 shows the distribution of the time needed by jwcdg-2 for initiation. The differences to jwcdg-1 (cf. Figure 4.3) are obvious: Not only is the mean initiation time much lower, there is also no single initiation that has taken more than one second, even for large increments. This enables the use of jwcdg-2 with enforcable time limit of about one second.

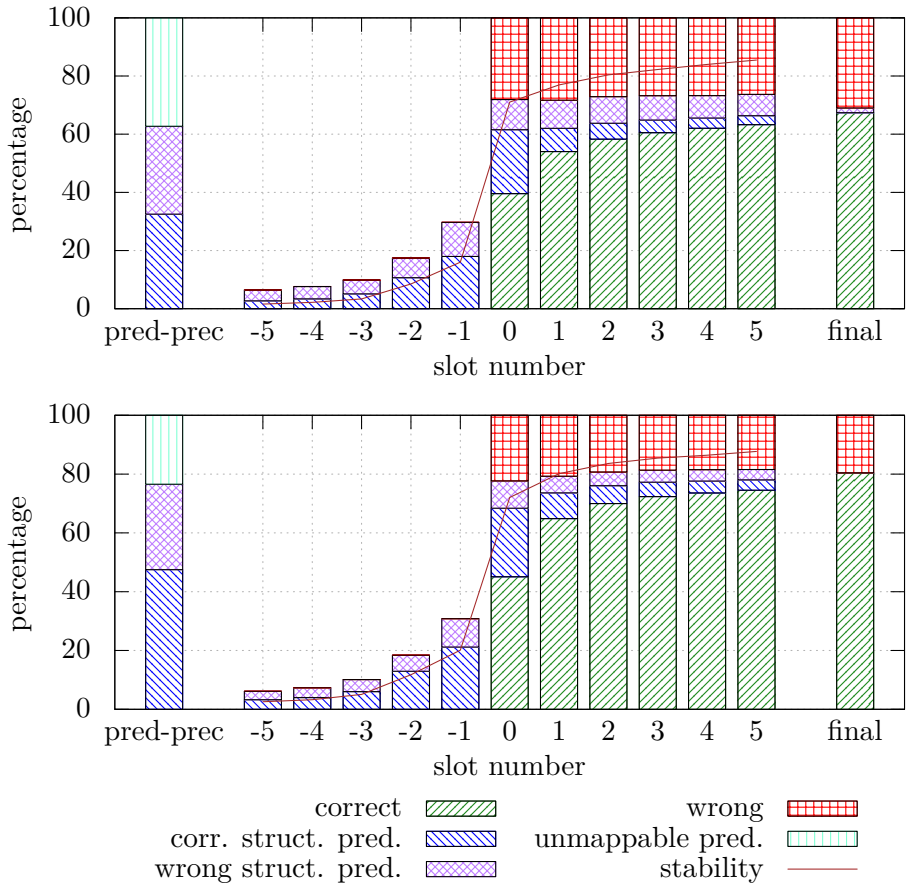


Figure 4.5.: Results for jwcdg-2 with a time limit of one second (above) and eight seconds(below)

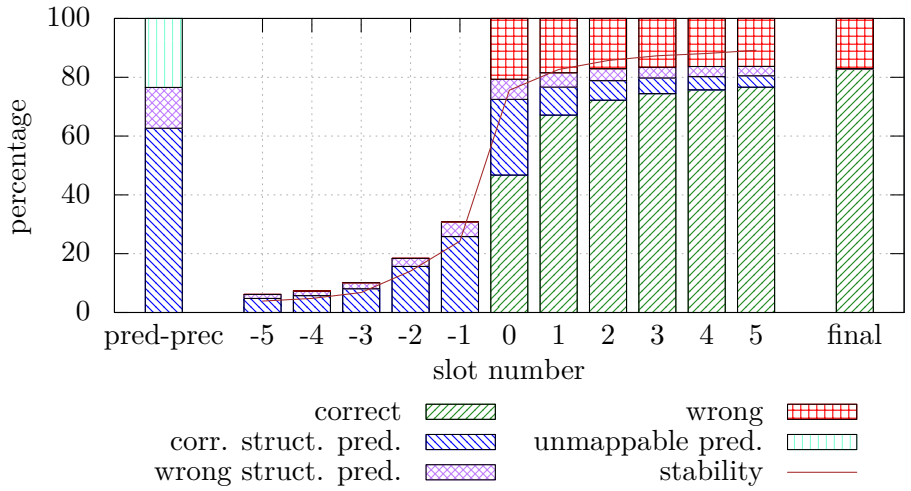


Figure 4.6.: Results for jwcdg-2 with a time limit of eight seconds (unlabeled)

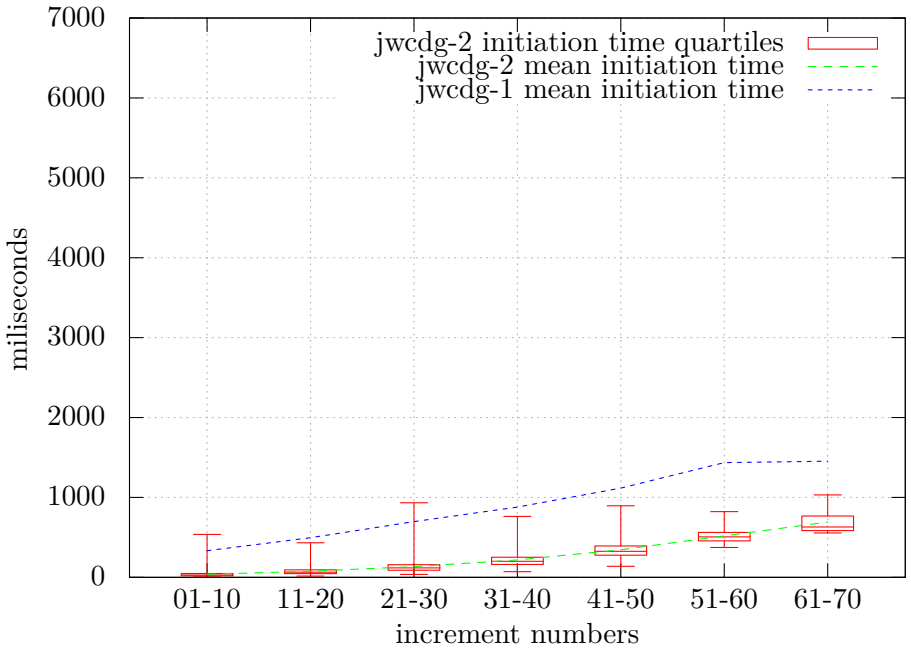


Figure 4.7.: Initiation times of jwcdg-2

		time limit			
		1 sec	2 sec	4 sec	8 sec
initial attachment	jwcdg-0	56.01%	61.27%	64.44%	66.39%
	jwcdg-1	58.33%	62.80%	65.59%	67.03%
	jwcdg-2	61.55%	64.45%	66.96%	68.39%
final accuracy	jwcdg-0	58.48%	67.22%	72.58%	76.60%
	jwcdg-1	62.05%	69.46%	74.46%	77.95%
	jwcdg-2	67.40%	73.22%	77.43%	80.36%

Table 4.4.: Effect of the timelimit on results of jwcdg-2

5. Incorporating MaltParser into jwcdg

When facing a tight time limit, jwcdg has only very little time to improve an analysis by transformation. Therefore, a good initial attachment gets more important with tighter time limits. This means that a method to allow jwcdg to start frobbing with a good analysis could help to achieve drastically better results.

Foth and Menzel (2006) showed that WCDG can be augmented by machine learning based components to raise the accuracy of WCDG. The output of the components was converted to 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. In a similar way, Khmylko, Foth, and Menzel (2009) used the MST parser (McDonald, Lerman, and Pereira 2006) (which does not work incrementally) as an external data source for WCDG.

This chapter will deal with incorporating MaltParser into jwcdg so that jwcdg can use MaltParser's analyses as a possible starting point. First an overview of MaltParser will be given and then we will look at how MaltParser can be incorporated into jwcdg and how it affects the parsing results.

5.1. MaltParser

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. This means that MaltParser is data-driven. MaltParser can use several different parsing algorithms. The simplest one (introduced in Nivre (2003), *nivre*) will be used to explain MaltParser's parsing approach. For evaluation, the algorithm *2planar* described in Gómez-Rodríguez and Nivre (2010) will be used, which is more complex.

5.1.1. Parsing with MaltParser

The state of the parser is a triple $< S, I, A >$ where S is a stack, I the list of remaining input words and A the set of already established dependency edges. The initial state is as follows: S is empty, I contains all words that should be parsed and A is empty , i. e. $< \mathbf{nil}, W, \emptyset >$. The parser now repeatedly decides which of the following four actions should be executed:

Shift Move the next word from the input onto the stack

Left-Arc Add an edge from the word on top of the stack to the next input word. Shift the word from the input onto the stack

Right-Arc Add an edge from the next input word to the word on top of the stack. Shift the word from the input onto the stack

Reduce Pop the word that is on top of the stack

The parser stops when the input and the stack are empty. All remaining words which have no regent are then attached to the NIL node. An example run of MaltParser can be seen in Figure 5.1, where MaltParser parses the sentence "Peter kauft ein Eis" ("Peter buys an ice", Peter buys ice cream).

MaltParser uses a set of features to decide which action to take. They are fed into a decision component which utilizes Support Vector Machines. The features that are used by the decision components can be freely defined. Since MaltParser uses the part-of-speech tag of the words as a feature, it requires the input to contain them already.

5.1.2. Training MaltParser

While training, MaltParser uses an oracle which, given a parse for a sentence, produces a list of actions that would result in that parse. The decision component is then trained on all the state-action pairs that have been generated in this way from a set of annotated sentences. The training algorithm does not need to be adapted. The interested reader can read about training MaltParser in Nivre et al. (2007).

5.1.3. MaltParser and Incrementality

MaltParser consumes the input from left to right and, if using the *2planar* or the *nivre* 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

	state			
state nr.	attachments	stack	input	next action
1			Peter kauft ein Eis	Shift
2		Peter	kauft ein Eis	Left Arc
3	(Peter → kauft)		kauft ein Eis	Shift
4	(Peter → kauft)	kauft	ein Eis	Shift
5	(Peter → kauft)	ein kauft	Eis	Left Arc
6	(Peter → kauft) (ein → Eis)	kauft	Eis	Right Arc
7	(Peter → kauft) (ein → Eis) (Eis → kauft)	Eis kauft		2x Reduce
8	(Peter → kauft) (ein → Eis) (Eis → kauft)			[done]

Figure 5.1.: Parsing “Peter kauft ein Eis” using the *nivre* algorithm

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

Figure 5.2.: Effect of lookahead for MaltParser, source: Beuck and Köhn (2012)

every edge has to be constructed as soon as the second word of the edge is the current input word. As soon as that word gets shifted onto the stack, the creation of the edge would not be possible.

The parser works monotonically since obviously edges are only added to but never removed from the set of edges. MaltParser adheres to decisiveness because it only creates one edge for every word where that word is the dependant. This means that accuracy and lookahead are the only variables left from the ones described in Section 1.5. And indeed MaltParser uses lookahead to gain higher accuracy in the configurations suggested by its authors: the decision component uses features of the three words after the current one.

Beuck, Köhn, and Menzel (2011b) have shown that at least for German MaltParser's performance only slightly degrades when only using features from the next two words instead of three. However, since the POS tags are needed and a POS tagger needs lookahad 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 5.2.

As a delay is not acceptable for our scenario, we will focus on using MaltParser without lookahad despite its degraded accuracy in that mode.

5.2. An Interface between MaltParser and jwcdg

Although MaltParser uses an incremental algorithm, it does not have an incremental interface. Therefore, a new class extending MaltParser was developed for this thesis to obtain an incremental interface. The new `IncrementalMaltParser` provides the methods `addWord`, `doStep` and `finalizeParse` to add a word to the input, parse until a shift action occurs (i.e. the next word will be read) and to finish parsing the remaining input, assuming the sentence to be complete.

In jwcdg, the `IncrementalMaltParser` is used by the `MaltPredictor`. `MaltPredictor` reads the most probable tag for each word (this has been set before by the part-of-speech predictor), submits the words together with their pos-tags to the `IncrementalMaltParser`

```
{X:SYN} : maltanbindung : malt : 0.9 : (fluctuating) :
  notanumber(predict(X@id, malt, regent))
  // something is set by the predictor
  -> isLexemeOf(X^id, predict(X@id, malt, regent));

{X:SYN} : maltlabel : malt : 0.9 : (fluctuating) :
  notanumber(predict(X@id, malt, mlabel))
  // something is set by the predictor
  -> predict(X@id, malt, mlabel) = X.label;

{X:SYN} : 'Malt prädiziert in die Zukunft' : malt : 0.9 : (fluctuating) :
  ~ notanumber(predict(X@id, malt, regent))
    // no prediction from malt -> malt will bind this word to the future
  ->
  virtual(X@id, "misc") | // there are no predictions for virtual nodes
  virtual(X^id,'') |
  nonspec(X^id) |
  X|;
```

Figure 5.3.: Constraints for incorporating MaltParsers results into jwcdg

and then performs as many `doStep` as new words have been added. If the sentence is marked as being complete, `finalizeParse` is called. After this, `MaltPredictor` reads the state of `IncrementalMaltParser` and stores for each word the regent it has been assigned to by MaltParser. If no regent has been assigned to a word, this fact is also stored. Since MaltParser works eagerly (i. e. constructs every edge as soon as possible), the regents of those words must lie in the future. The predicted regent for every word can now be accessed by constraints.

5.2.1. Building Constraints to Access MaltParser's Predictions

The three constraints that are used for accessing MaltParser's analyses are depicted in Figure 5.3. The first constraint checks that if the `MaltPredictor` has set a regent for a word, this word is also the regent in the current analysis. 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 states that if no prediction has been made by MaltParser, either the dependent is virtual (in this case MaltParser

does not know about it) or the regent lies in the future or the regent is NIL. The third constraint is not as straight forward 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). The other possible case is that the dependent of the edge on which the constraint is currently evaluated is a virtual node in which case MaltParser cannot possibly predict an attachment. A weight of 0.9 has proved to work best for the constraints based on an evaluation using the sentences 501 to 1000 of the NEGRA corpus.

When an analysis is being extended, the new word has to be attached somewhere. Since all unary constraints that are not context sensitive are evaluated on an edge when it is created, jwcdg has access to the penalties of each possible new edge. To attach the new word somewhere to the new analysis, jwcdg uses the edge with the least penalty. The MaltParser constraints are unary and not context sensitive and therefore jwcdg will use the edge predicted by MaltParser if no other constraint prohibits the use of this edge.

5.2.2. The Monotonicity Mismatch

Whereas MaltParser is a monotonic parser, the underling TnT tagger is not monotonic. This means that TnT can change the part-of-speech tag for words which MaltParser has already processed. This could lead to suboptimal parsing results because MaltParser uses inconsistent part-of-speech tags. To mitigate this problem, `MaltPredictor` re-parses every increment completely with MaltParser instead of just adding the new word and continuing parsing.

This means that the time needed to run MaltParser does not take constant time anymore but its processing time will be proportional to the number of words in the current increment. This variant of jwcdg will be called jwcdg-3.

5.3. Evaluation of jwcdg with MaltParser

Table 5.1 shows that jwcdg-3 benefits from the prediction made by MaltParser. However, the final score for jwcdg with MaltParser and a time limit of one second is abysmal. Although the final score gets better with a more generous time limit, this result is not satisfying. Figure 5.4 shows the reason for this low score: MaltParser leads jwcdg to better initial attachments, but it also uses most of the time that is

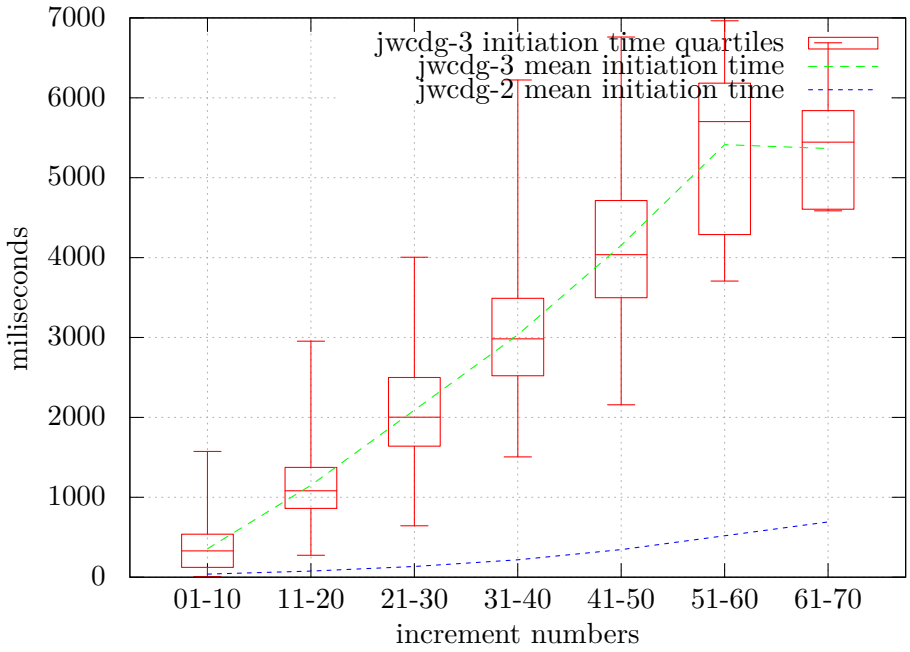


Figure 5.4.: Initiaion time over the increment length for jwcdg-3

available for processing. jwcdg has nearly no time left for transforming the analysis in the frobbing algorithm.

5.4. Disabling Reanalysis for MaltParser

MaltParser needs to be made faster to achieve better results. One possible approach is to drop the reanalysis described in Section 5.2.2. This would possibly reduce the accuracy of MaltParser but increase the time that is available to jwcdg. To check if this change leads to better results, jwcdg has been adapted to store the MaltParser state so that only the newest word needs to be added and only one `doStep` needs to be run (resulting in jwcdg-4). The resulting initiation time distribution can bee seen in Figure 5.5. The new version uses considerably less time for initiation and the mean time is even close to the version without MaltParser.

The reduced initiation time is also reflected in the accuracy, see Table 5.1. jwcdg-4 is the best performing version with respect to the final accuracy with all time limits. The initial attachment accuracy of jwcdg-4 is also the best for tight time limits. Only

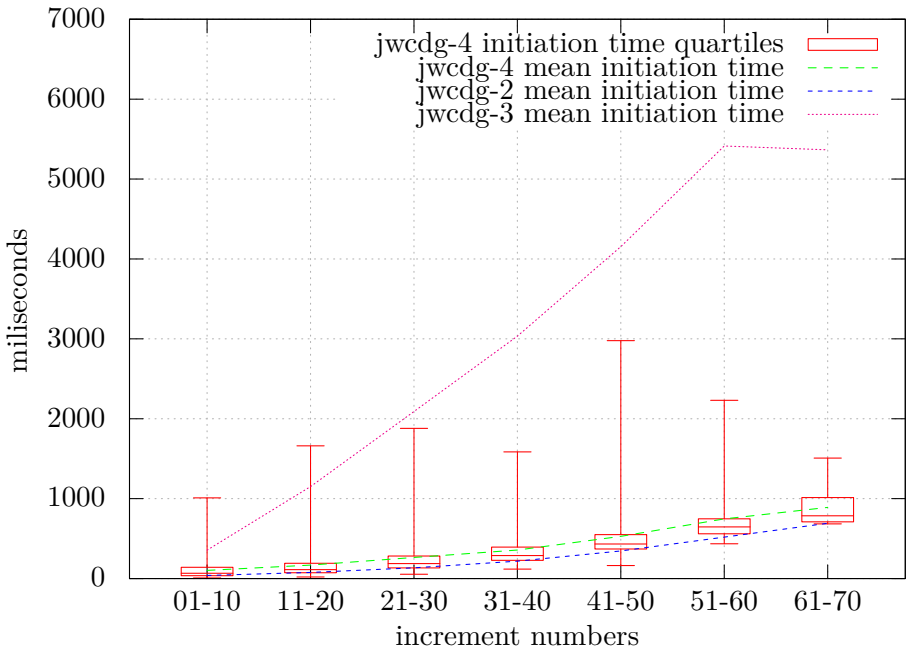


Figure 5.5.: Initiation time over the increment length for jwcdg-4

		time limit				
	parser	1 sec	2 sec	4 sec	8 sec	16 sec
initial attachment	jwcdg-2	61.55%	64.45%	66.96%	68.39%	69.39%
	jwcdg-3	70.83%	72.49%	74.46%	75.95%	76.79%
	jwcdg-4	71.57%	73.05%	74.27%	75.91%	76.39%
final accuracy	jwcdg-2	67.40%	73.22%	77.43%	80.36%	82.70%
	jwcdg-3	58.82%	73.31%	79.48%	84.64%	85.99%
	jwcdg-4	72.76%	78.41%	82.89%	85.09%	86.02%

Table 5.1.: Results for jwcdg with MaltParser without reanalysis (jwcdg-4)

with a time limit of four or more seconds jwcdg-3 can profit from the better accuracy of MaltParser in that mode. Even then the slightly better initial attachment score does not result in better final scores. In contrast to jwcdg-3, jwcdg-4 outperforms jwcdg-2 in every aspect and relatively tight time limits can be enforced.

5.5. Comparing jwcdg and MaltParser

To make sure that the results of jwcdg-4 are not only the results of MaltParser piped through jwcdg, the results of jwcdg-4 and MaltParser need to be compared. Since MaltParser is not able to produce structural prediction, the evaluation in this section will only use minimal prediction meaning that an attachment to the future is considered correct if the correct attachment would be an attachment to any word in the future. Note that this is a less strict evaluation scheme than the one used before and the numbers are therefore not directly comparable.

Since MaltParser does not have an incremental interface for writing the results of incremental steps, jwcdg is used as that interface. For this, jwcdg is started with a minimal grammar consisting only of the MaltParser constraints (Figure 5.3) and a constraint for handling punctuation marks. This way it is guaranteed that jwcdg emits exactly the analysis that MaltParser emits and that MaltParser gets exactly the information about POS-tags that jwcdg can use in its normal mode. This is important because jwcdg compares the results of the POS tagger and adjusts it if jwcdg's lexicon disagrees with the tagger about the possible tags for a word.

As can be seen in Table 5.2, MaltParser outperforms jwcdg-4 with a time limit of one second regarding the final score. With a time limit of two or more seconds,

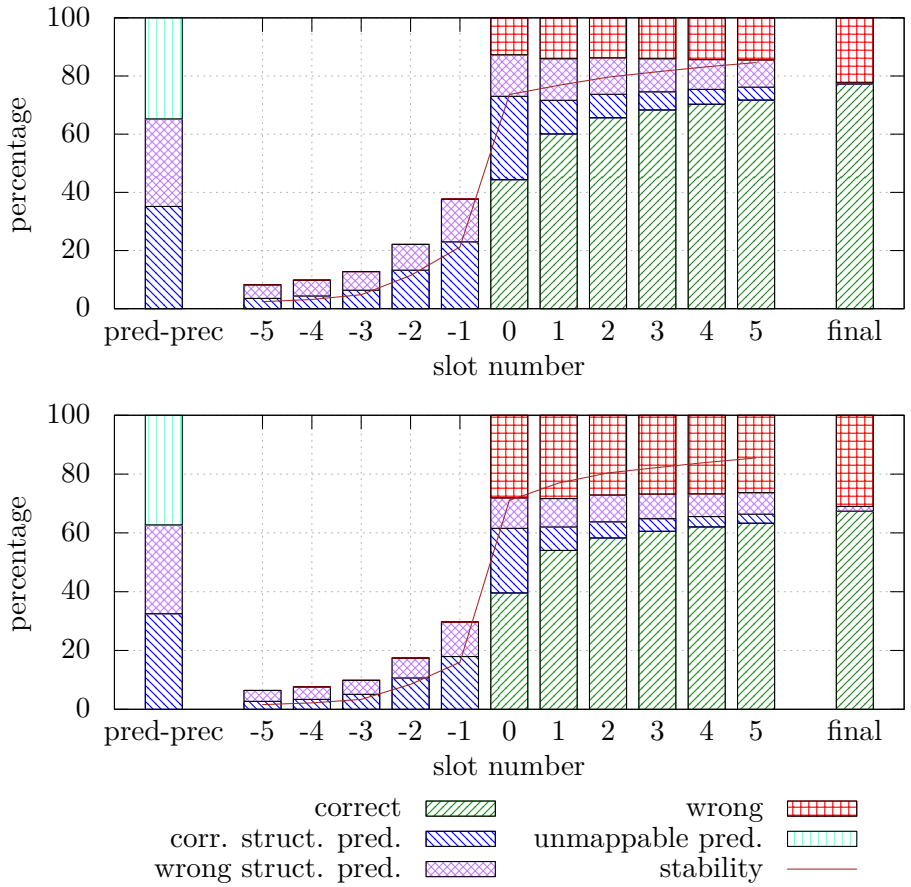


Figure 5.6.: Results for jwcdg-4 (above) and jwcdg-2 (below) with a time limit of one second

		time limit (seconds)				
	parser	1	2	4	8	16
initial attachment	jwcdg-2	63.09%	66.17%	68.56%	69.99%	71.09%
	jwcdg-4	74.34%	75.65%	77.06%	78.39%	78.90%
	MaltParser	51.51%	51.51%	51.51%	51.51%	51.51%
final accuracy	jwcdg-2	67.40%	73.22%	77.43%	80.36%	82.70%
	jwcdg-4	72.76%	78.41%	82.89%	85.09%	86.02%
	MaltParser	77.93%	77.93%	77.93%	77.93%	77.93%

Table 5.2.: Comparison of jwcdg and MaltParser using minimal prediction (labeled)

		time limit				
	parser	1 sec	2 sec	4 sec	8 sec	16 sec
initial attachment	jwcdg-2	69.04%	71.89%	73.98%	75.26%	76.21%
	jwcdg-4	81.19%	82.18%	83.18%	84.25%	84.74%
	MaltParser	84.64%	84.64%	84.64%	84.64%	84.64%
final accuracy	jwcdg-2	70.84%	76.29%	80.42%	82.86%	85.29%
	jwcdg-4	75.98%	81.37%	85.57%	87.63%	88.41%
	MaltParser	80.97%	80.97%	80.97%	80.97%	80.97%

Table 5.3.: Comparison of jwcdg and MaltParser using minimal prediction (unlabeled)

jwcdg-4 outperforms MaltParser in that category. The initial attachment accuracy of MaltParser is fairly low because it does not assign a label to attachments into the future. Table 5.3 shows that when omitting the labels from the analysis, MaltParser has a high accuracy for the initial attachment that can only be matched by jwcdg-4 with generous time limits. It has, however, to be noted that this evaluation discards all the additional information jwcdg-4 predicts.

6. Evaluation on Sentences with Different Characteristics

All the evaluations in the previous chapters have been carried out on the NEGRA corpus. NEGRA consists of newspaper texts and thus represents a very specific kind of text. However, in most of the scenarios outlined in Section 1.4 the type of the text has other characteristics: The sentences are shorter and usually have a lower structural complexity than sentences from newspapers. This chapter tries to account for this differences by adjusting the corpora that are used for evaluation.

6.1. Evalation on Subsets of NEGRA

To evaluate how the sentence length influences the parsing results, jwcdg has been evaluated on subsets of the NEGRA corpus. Sentences with a length of less than three have been excluded, which are mostly fragments such as "Sulzbach ."[1]. In addition, sentences longer than x (for $x \in \{10, 20, 30\}$) have been excluded, resulting in the subsets, NEGRA-3-10, NEGRA-3-20 and NEGRA-3-30.

The results shown in Table 6.1 show that jwcdg's accuracy increases when only evaluated on short sentences. This increase could be due to the smaller syntactic complexity of short sentences and the shorter initiation time needed for short increments (cf. Figure 5.5) leaving more time for frobbing. On NEGRA-3-10, jwcdg-4 even produces competitive results with a time limit of only one second.

6.2. Evaluation on the creg-109 Corpus

Section 1.4.3 mentioned the use case of Intelligent Computer-Assisted Language Learning (ICALL) systems for predictive parsing. To get an impression of how well jwcdg works in this domain (and the effects of time limits in this domain), an evaluation has been conducted on the creg-109 corpus, a set of 109 sentences of the corpus described in Meurers, Ott, and Ziai (2010). The creg-109 corpus "consists of

[1] NEGRA sentence number 18663

		time limit				
	parser	1 sec	2 sec	4 sec	8 sec	16 sec
initial attachment 3-10	jwcdg-2	74.82%	75.18%	76.06%	75.70%	75.94%
	jwcdg-4	78.52%	79.28%	79.81%	79.75%	79.99%
final attachment 3-10	jwcdg-2	88.26%	88.91%	89.32%	88.56%	88.97%
	jwcdg-4	90.08%	90.73%	90.73%	90.90%	90.73%
initial attachment 3-20	jwcdg-2	69.50%	71.39%	73.80%	74.30%	74.83%
	jwcdg-4	75.28%	77.44%	78.32%	79.40%	78.85%
final attachment 3-20	jwcdg-2	79.96%	84.76%	87.07%	88.34%	88.96%
	jwcdg-4	83.38%	88.57%	90.16%	90.43%	89.65%
initial attachment 3-30	jwcdg-2	64.80%	67.72%	69.97%	70.82%	71.67%
	jwcdg-4	73.01%	74.57%	75.74%	77.03%	77.42%
final attachment 3-30	jwcdg-2	72.30%	77.62%	81.35%	83.52%	85.40%
	jwcdg-4	77.21%	82.54%	85.78%	87.06%	87.65%

Table 6.1.: Evaluation results on NEGRA-3-X

answers to German reading comprehension questions written by American college students learning German" (Meurers, Ott, and Ziai 2010).

Table 6.2 shows the accuracy of the different parser versions on this corpus. The results have a different pattern than the ones obtained from using the NEGRA corpus: jwcdg-2 has nearly the same initial attachment accuracy as jwcdg-4 and even slightly outperforms it in the final score. Not only the accuracy for the initial attachments ar nearly identical for jwcdg-2 and jwcdg-4, their performance is nearly identical, as can be seen in Figure 6.1. In addition to that, the result does not improve much anymore 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 needed to guide jwcdg.

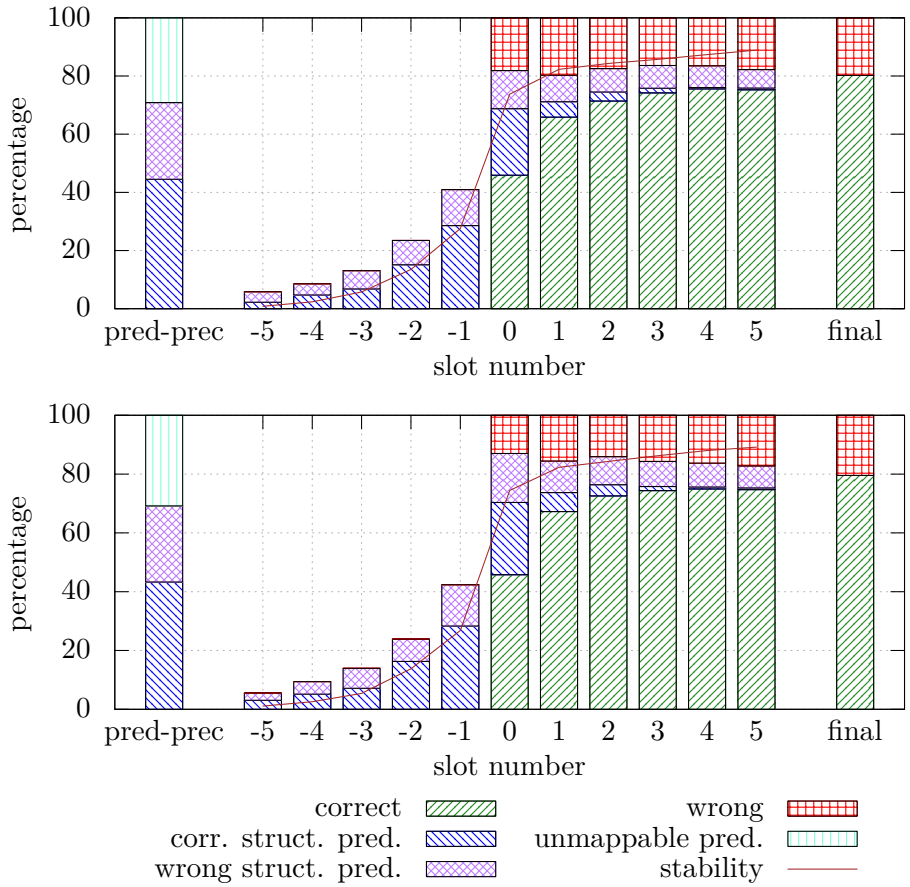


Figure 6.1.: Results for jwcdg-2 (above) and jwcdg-4 (below) with a time limit of 16 seconds on the CREG-109 corpus

		time limit				
	parser	1 sec	2 sec	4 sec	8 sec	16 sec
initial attachment	jwcdg-0	64.11	65.67	67.56	68.33	68.78
	jwcdg-2	65.56	68.11	69.00	69.11	68.78
	jwcdg-4	67.22	70.00	70.22	70.78	70.33
final attachment	jwcdg-0	71.89	76.67	78.11	79.22	80.22
	jwcdg-2	78.11	79.56	80.11	80.00	80.22
	jwcdg-4	77.44	79.22	79.67	79.44	79.56

Table 6.2.: Evaluation results on the CREG-109 corpus

7. Conclusion and Future Work

jwcdg has been extended to be capable of predictive interactive parsing under relatively tight time constraints. The frobbing algorithm benefits from parallelization and can make use of up to 32 cores. Time limits of about one second are enforceable, at least when not using MaltParser as a predictor.

Using MaltParser as a predictor has shown to be useful, using reanalysis for MaltParser has not, at least not for the time limits that have been used in this experiment. The use of MaltParser enables jwcdg to produce a better initial attachment which means that less time has to be used correcting attachment errors. MaltParser is more helpful when parsing long and complex sentences; when parsing shorter sentences from NEGRA-3-10 or CREG-109, the benefit is less noticeable.

While the proposed changes to jwcdg already provide a noticeable speedup and the use of MaltParser also helps, there are still possible optimizations that have not been dealt with in this thesis. Two of them are outlined in the following sections.

7.1. Compiling Constraint Formulas

jwcdg parses constraints into object trees where each object represents a sub-formula which has references to all its direct sub-formulas and a method to evaluate this sub-formula. A formula is evaluated by calling the `evaluate` method on the formula which calls its sub-formulas `evaluate` method and combines the result (e. g. by conjunctive combination in case of an *and* formula). This approach has the advantage of encapsulating the knowledge of how constraint constructs are evaluated but it also leads to inefficiencies because for each sub-formula a method needs to be invoked.

Hagenström (2001) has implemented a component for the cdg parser (the predecessor of jwcdg, written in C) which compiles the grammar of cdg via C code to machine code. This code can then used as a library instead of interpreding the constraints. The compilation did not lead to a speedup and this functionality has been abandoned.

To test whether compiling constraints can lead to a speed up in jwcdg, a short experiment has been done by manually translating a single constraint to native java

code. Evaluating this constraint using the compiled version turned out to be about 1.8 times faster than using the interpreted version. This shows that a speed up is indeed achievable by compiling constraints.

The implementation would, however, need to be easily understandable and ideally allow mixing of language constructs for which the compilation to java code is defined and constructs for which no such translation is defined. Otherwise all extensions to the constraint language would either need to provide a translator or the constraint compilation would need to be disabled all together. The compiling step will be easier to do than in the C version because modern Java virtual machines support the compilation of java code at run time without having to deal with low level components.

7.2. Restricting the Generateable Analysis Structures

The dependency structures that can be generated by jwcdg are not restricted. Properties such as non-cyclicity are only enforced by constraints in the grammar. This means that, for example, analyses with a cycle are being generated, judged and then discarded because they violate a hard constraint. Restricting which analyses can be constructed could drastically reduce the size of the search space, speeding up the algorithm. Gómez-Rodríguez and Nivre (2010) showed that most (i. e. >98%) analyses that can be found in corpora of different languages are 2-planar. Kuhlmann and Nivre (2006) have shown that nearly all analyses of the Prague Dependency Treebank and the Danish Dependency Treebank are well-nested while only 76.85% (PDT) and 84.95% (DDT) are projective. Restricting the search space to either well-nested or 2-planar analyses could speed up the parsing process and therefore lead to better results.

A. Software used for the Experiments

All evaluation has been done on a 64bit GNU/Linux system using The OpenJDK Runtime Environment version 7 (64bit).

The different versions of the parser can be found on `https://gitorious.org/jwcdg/msc-koehn`. The branches are as follows:

old_varbindings The baseline version (jwcdg-0)

master The version containing the constraint argument optimization (Section 4.1, jwcdg-1)

parallel_too_slow jwcdg with parallelized `evalBinary` (Section 4.2.1)

parallel jwcdg with parallelized `optimizeDomains` and `buildIter` (Section 4.2.2, jwcdg-2)

buildIterParallel The version for timing measurements of `optimizeDomains` in Section 4.2.5

jwcdg-3 and jwcdg-4 are the same as jwcdg-2 but have MaltParser enabled as a predictor. The lexicon needed for jwcdg is not contained in the git repository and can be downloaded from `https://nats-www.informatik.uni-hamburg.de/pub/CDG/DownloadPage/german-lexicon.tar.bz2`. For the timing measurements of `optimizeDomains` and `buildIter`, the variable `performTimingMeasurements` in the respective source files needs to be set to true. Evaluation of the results has been done using `cdgevaluator_timecourse.py` from the the parseeval toolbox, which can be found on `https://gitorious.org/poseval/parseeval`.

B. Raw Experiment Results

These are the raw evaluation results that have been used in this thesis. The columns are prediction precision (pp), the different slots (by number) and the final scores (f). The columns are:

1. the number of attachments that have been counted in that slot
2. the number of virtual nodes in that slot
3. the number of correct attachment (unlabeled)
4. the number of correct attachment (labeled)
5. the stability (unlabeled)
6. the stability (labeled)
7. Correct attachments to the future for minimal evaluation
8. the number of correct attachments involving virtual nodes (unlabeled)
9. the number of correct attachments involving virtual nodes (labeled)
10. (pp only) count of predictions made overall
11. (pp only) number of correct predictions overall (unlabeled)
12. (pp only) number of correct predictions overall (labeled)
13. (pp only) number of predictions that could not be mapped to a word

Evaluation NEGRA

jwcdg-0

16sec

pp	12567	0	8135	6384	0	0	12567	8135	6384	12567	8135	6384	2724
-5	8403	7890	407	301	333	237	513	407	301	0	0	0	0
-4	9002	8349	526	382	446	313	653	526	382	0	0	0	0
-3	9617	8663	764	605	667	509	954	764	605	0	0	0	0
-2	10249	8349	1627	1370	1477	1242	1900	1627	1370	0	0	0	0
-1	10919	7579	2819	2361	2690	2253	3340	2819	2361	0	0	0	0
0	11623	0	8387	7961	9052	8667	3581	2891	2670	0	0	0	0
1	10919	0	8325	8010	9266	9015	1442	976	899	0	0	0	0
2	10249	0	8039	7787	9010	8813	1014	644	589	0	0	0	0
3	9617	0	7618	7373	8572	8398	806	479	435	0	0	0	0
4	9002	0	7169	6955	8106	7953	653	377	341	0	0	0	0
5	8403	0	6741	6546	7644	7511	548	315	293	0	0	0	0
f	11623	0	9618	9325	11623	11623	20	0	0	0	0	0	0

8sec

pp	18634	0	11310	8590	0	0	18634	11310	8590	18634	11310	8590	4789
-5	12029	11293	577	399	439	280	736	577	399	0	0	0	0
-4	12889	11971	728	513	576	377	918	728	513	0	0	0	0
-3	13778	12426	1082	821	874	627	1352	1082	821	0	0	0	0
-2	14691	12058	2211	1813	1953	1587	2633	2211	1813	0	0	0	0
-1	15644	10941	3906	3184	3632	2969	4703	3906	3184	0	0	0	0
0	16642	0	11708	11049	12736	12171	5172	4112	3762	0	0	0	0
1	15644	0	11548	11049	13073	12686	2189	1411	1294	0	0	0	0
2	14691	0	11158	10753	12700	12408	1547	917	834	0	0	0	0
3	13778	0	10579	10205	12128	11883	1232	678	617	0	0	0	0
4	12889	0	9962	9622	11489	11271	1024	549	495	0	0	0	0
5	12029	0	9322	9018	10829	10638	883	444	408	0	0	0	0
f	16642	0	13233	12747	16642	16642	76	0	0	0	0	0	0

4sec

pp	19583	0	11243	8104	0	0	19583	11243	8104	19583	11243	8104	5720
-5	12029	11298	572	370	395	238	731	572	370	0	0	0	0
-4	12889	11948	738	486	542	335	941	738	486	0	0	0	0
-3	13778	12415	1077	781	808	566	1363	1077	781	0	0	0	0
-2	14691	12074	2180	1723	1847	1457	2617	2180	1723	0	0	0	0
-1	15644	11005	3786	3032	3428	2760	4639	3786	3032	0	0	0	0
0	16642	0	11444	10724	12908	12335	5007	4022	3671	0	0	0	0
1	15644	0	11031	10471	13084	12690	2201	1347	1216	0	0	0	0
2	14691	0	10594	10130	12677	12379	1646	859	767	0	0	0	0
3	13778	0	10064	9640	12123	11872	1326	628	550	0	0	0	0
4	12889	0	9494	9102	11475	11272	1107	502	433	0	0	0	0
5	12029	0	8899	8557	10801	10627	950	408	365	0	0	0	0
f	16642	0	12586	12079	16642	16642	159	0	0	0	0	0	0

2sec

pp	20134	0	10923	7306	0	0	20134	10923	7306	20134	10923	7306	6429
-5	12029	11293	570	327	369	202	736	570	327	0	0	0	0
-4	12889	11958	722	430	498	286	931	722	430	0	0	0	0
-3	13778	12472	1018	687	733	473	1306	1018	687	0	0	0	0
-2	14691	12180	2072	1584	1692	1286	2511	2072	1584	0	0	0	0
-1	15644	11238	3525	2741	3138	2447	4406	3525	2741	0	0	0	0
0	16642	0	10931	10197	12997	12466	4757	3807	3488	0	0	0	0
1	15644	0	10293	9684	13016	12668	2274	1293	1158	0	0	0	0
2	14691	0	9841	9330	12605	12330	1718	793	696	0	0	0	0
3	13778	0	9368	8911	12039	11830	1404	576	500	0	0	0	0
4	12889	0	8830	8388	11392	11217	1191	450	383	0	0	0	0
5	12029	0	8311	7912	10747	10611	1044	366	312	0	0	0	0
f	16642	0	11740	11187	16642	16642	311	0	0	0	0	0	0

1sec

pp	20350	0	10363	6104	0	0	20350	10363	6104	20350	10363	6104	7408
-5	12029	11271	590	287	424	189	758	590	287	0	0	0	0
-4	12889	11961	719	379	508	252	928	719	379	0	0	0	0
-3	13778	12536	963	582	675	381	1242	963	582	0	0	0	0
-2	14691	12453	1837	1291	1456	1016	2238	1837	1291	0	0	0	0
-1	15644	11903	2983	2206	2597	1940	3741	2983	2206	0	0	0	0
0	16642	0	10108	9321	13164	12752	4282	3428	3171	0	0	0	0
1	15644	0	9021	8327	12818	12561	2386	1168	1032	0	0	0	0
2	14691	0	8571	7959	12388	12199	1916	727	622	0	0	0	0
3	13778	0	8168	7605	11841	11697	1627	525	435	0	0	0	0
4	12889	0	7735	7223	11225	11123	1413	417	349	0	0	0	0
5	12029	0	7279	6800	10583	10490	1257	341	280	0	0	0	0
f	16642	0	10382	9732	16642	16642	609	0	0	0	0	0	0

jwcdg-1

32sec

pp	17502	0	11488	9082	0	0	17502	11488	9082	17502	11488	9082	3617
-5	12029	11295	577	427	485	342	734	577	427	0	0	0	0
-4	12889	11972	737	540	643	453	917	737	540	0	0	0	0
-3	13778	12416	1099	850	964	727	1362	1099	850	0	0	0	0
-2	14691	12018	2293	1931	2105	1762	2673	2293	1931	0	0	0	0
-1	15644	10920	4016	3353	3843	3206	4724	4016	3353	0	0	0	0
0	16642	0	12126	11492	12784	12205	5246	4220	3862	0	0	0	0
1	15644	0	12150	11688	13166	12785	2091	1431	1313	0	0	0	0
2	14691	0	11736	11355	12840	12535	1480	949	875	0	0	0	0
3	13778	0	11114	10759	12233	11973	1162	725	665	0	0	0	0
4	12889	0	10461	10133	11595	11367	935	566	514	0	0	0	0
5	12029	0	9793	9491	10909	10728	791	467	430	0	0	0	0
f	16642	0	14008	13578	16642	16642	14	0	0	0	0	0	0

16sec

pp	17797	0	11510	9010	0	0	17797	11510	9010	17797	11510	9010	3883
-5	12029	11300	577	417	476	330	729	577	417	0	0	0	0
-4	12889	11973	738	537	620	434	916	738	537	0	0	0	0
-3	13778	12431	1083	843	934	707	1347	1083	843	0	0	0	0
-2	14691	12020	2284	1909	2080	1724	2671	2284	1909	0	0	0	0
-1	15644	10912	3996	3328	3800	3168	4732	3996	3328	0	0	0	0
0	16642	0	12010	11373	12809	12217	5256	4205	3850	0	0	0	0
1	15644	0	11971	11501	13192	12784	2128	1430	1308	0	0	0	0
2	14691	0	11585	11209	12860	12541	1505	946	871	0	0	0	0
3	13778	0	10975	10616	12248	11978	1196	722	662	0	0	0	0
4	12889	0	10316	9990	11590	11349	970	565	514	0	0	0	0
5	12029	0	9666	9360	10894	10700	808	460	425	0	0	0	0
f	16642	0	13802	13354	16642	16642	20	0	0	0	0	0	0

8sec

pp	18113	0	11255	8723	0	0	18113	11255	8723	18113	11255	8723	4343
-5	12029	11300	566	406	443	300	729	566	406	0	0	0	0
-4	12889	11959	740	529	603	406	930	740	529	0	0	0	0
-3	13778	12425	1083	835	907	667	1353	1083	835	0	0	0	0
-2	14691	12068	2221	1845	1998	1634	2623	2221	1845	0	0	0	0
-1	15644	10960	3924	3228	3682	3029	4684	3924	3228	0	0	0	0
0	16642	0	11786	11155	12767	12213	5177	4131	3789	0	0	0	0
1	15644	0	11700	11213	13116	12737	2158	1431	1304	0	0	0	0
2	14691	0	11322	10929	12762	12480	1528	934	851	0	0	0	0
3	13778	0	10726	10362	12168	11923	1219	696	631	0	0	0	0
4	12889	0	10090	9756	11542	11324	998	544	489	0	0	0	0
5	12029	0	9458	9152	10879	10698	845	439	403	0	0	0	0
f	16642	0	13440	12973	16642	16642	43	0	0	0	0	0	0

4sec

pp	19216	0	11171	8270	0	0	19216	11171	8270	19216	11171	8270	5365
-5	12029	11301	557	371	406	252	728	557	371	0	0	0	0
-4	12889	11958	721	488	555	350	931	721	488	0	0	0	0
-3	13778	12414	1076	798	844	595	1364	1076	798	0	0	0	0
-2	14691	12045	2205	1774	1920	1536	2646	2205	1774	0	0	0	0
-1	15644	10974	3828	3097	3517	2860	4670	3828	3097	0	0	0	0
0	16642	0	11572	10916	12802	12254	5080	4069	3723	0	0	0	0
1	15644	0	11271	10759	13041	12676	2200	1371	1252	0	0	0	0
2	14691	0	10876	10468	12689	12407	1590	891	809	0	0	0	0
3	13778	0	10327	9935	12093	11861	1274	659	589	0	0	0	0
4	12889	0	9725	9367	11469	11256	1054	523	465	0	0	0	0
5	12029	0	9120	8802	10809	10630	910	424	390	0	0	0	0
f	16642	0	12877	12391	16642	16642	126	0	0	0	0	0	0

2sec

pp	19528	0	10751	7623	0	0	19528	10751	7623	19528	10751	7623	6064
-5	12029	11318	547	334	369	220	711	547	334	0	0	0	0
-4	12889	11978	709	453	501	306	911	709	453	0	0	0	0
-3	13778	12458	1033	734	764	520	1320	1033	734	0	0	0	0
-2	14691	12133	2111	1660	1766	1379	2558	2111	1660	0	0	0	0
-1	15644	11152	3622	2864	3250	2585	4492	3622	2864	0	0	0	0
0	16642	0	11157	10451	12989	12448	4827	3888	3566	0	0	0	0
1	15644	0	10613	10038	13091	12733	2222	1313	1175	0	0	0	0
2	14691	0	10157	9683	12643	12366	1707	824	727	0	0	0	0
3	13778	0	9646	9216	12078	11856	1382	601	521	0	0	0	0
4	12889	0	9090	8698	11452	11276	1158	467	402	0	0	0	0
5	12029	0	8536	8186	10782	10641	1009	387	333	0	0	0	0
f	16642	0	12073	11560	16642	16642	251	0	0	0	0	0	0

1sec

pp	20142	0	10298	6600	0	0	20142	10298	6600	20142	10298	6600	7106
-5	12029	11301	559	304	383	203	728	559	304	0	0	0	0
-4	12889	11988	696	403	487	269	901	696	403	0	0	0	0
-3	13778	12531	958	615	674	409	1247	958	615	0	0	0	0
-2	14691	12353	1901	1401	1511	1113	2338	1901	1401	0	0	0	0
-1	15644	11590	3196	2429	2813	2160	4054	3196	2429	0	0	0	0
0	16642	0	10475	9707	13109	12626	4485	3610	3330	0	0	0	0
1	15644	0	9537	8890	12916	12610	2331	1227	1099	0	0	0	0
2	14691	0	9106	8549	12483	12271	1844	769	676	0	0	0	0
3	13778	0	8666	8171	11922	11774	1528	551	472	0	0	0	0
4	12889	0	8222	7756	11302	11191	1301	441	376	0	0	0	0
5	12029	0	7739	7307	10640	10544	1161	366	312	0	0	0	0
f	16642	0	10924	10327	16642	16642	457	0	0	0	0	0	0

jwcdg-2

1sec

pp	22538	0	10953	7325	0	0	22538	10953	7325	22538	10953	7325	8402
-5	12029	11258	575	326	364	192	771	575	326	0	0	0	0
-4	12889	11908	730	434	495	281	981	730	434	0	0	0	0
-3	13778	12422	1052	702	734	463	1356	1052	702	0	0	0	0
-2	14691	12131	2059	1565	1653	1248	2560	2059	1565	0	0	0	0
-1	15644	10998	3650	2810	3196	2501	4646	3650	2810	0	0	0	0
0	16642	0	11038	10243	12389	11821	5380	4058	3656	0	0	0	0
1	15644	0	10314	9703	12427	12038	2760	1406	1249	0	0	0	0
2	14691	0	9890	9366	12116	11811	2146	931	802	0	0	0	0
3	13778	0	9430	8933	11606	11330	1749	696	592	0	0	0	0
4	12889	0	8880	8450	11045	10824	1450	539	455	0	0	0	0
5	12029	0	8368	7982	10469	10283	1250	432	369	0	0	0	0
f	16642	0	11789	11216	16642	16642	267	0	0	0	0	0	0

2sec

pp	20706	0	11175	7963	0	0	20706	11175	7963	20706	11175	7963	6602
-5	12029	11280	564	357	388	233	749	564	357	0	0	0	0
-4	12889	11954	710	469	523	331	935	710	469	0	0	0	0
-3	13778	12405	1063	763	803	558	1373	1063	763	0	0	0	0
-2	14691	12043	2201	1716	1870	1450	2648	2201	1716	0	0	0	0
-1	15644	10887	3844	3069	3473	2780	4757	3844	3069	0	0	0	0
0	16642	0	11489	10725	12392	11791	5442	4160	3745	0	0	0	0
1	15644	0	11095	10533	12570	12170	2534	1427	1297	0	0	0	0
2	14691	0	10663	10188	12264	11946	1863	933	836	0	0	0	0
3	13778	0	10152	9732	11771	11504	1491	700	621	0	0	0	0
4	12889	0	9593	9210	11183	10950	1232	555	485	0	0	0	0
5	12029	0	9014	8672	10584	10391	1030	436	389	0	0	0	0
f	16642	0	12696	12185	16642	16642	87	0	0	0	0	0	0

4sec

pp	19482	0	11305	8381	0	0	19482	11305	8381	19482	11305	8381	5503
-5	12029	11297	559	375	435	279	732	559	375	0	0	0	0
-4	12889	11942	737	496	593	387	947	737	496	0	0	0	0
-3	13778	12416	1079	802	870	640	1362	1079	802	0	0	0	0
-2	14691	12006	2253	1825	2003	1632	2685	2253	1825	0	0	0	0
-1	15644	10866	3935	3205	3681	3028	4778	3935	3205	0	0	0	0
0	16642	0	11843	11144	12499	11893	5435	4249	3853	0	0	0	0
1	15644	0	11640	11122	12841	12419	2372	1460	1329	0	0	0	0
2	14691	0	11229	10788	12498	12168	1702	956	859	0	0	0	0
3	13778	0	10620	10232	11940	11643	1343	699	631	0	0	0	0
4	12889	0	9983	9629	11316	11062	1110	547	487	0	0	0	0
5	12029	0	9373	9050	10696	10482	918	437	397	0	0	0	0
f	16642	0	13383	12886	16642	16642	61	0	0	0	0	0	0

8sec

pp	18411	0	11536	8748	0	0	18411	11536	8748	18411	11536	8748	4320
-5	12029	11290	576	393	467	308	739	576	393	0	0	0	0
-4	12889	11946	740	514	617	421	943	740	514	0	0	0	0
-3	13778	12385	1114	831	937	697	1393	1114	831	0	0	0	0
-2	14691	11984	2302	1900	2072	1725	2707	2302	1900	0	0	0	0
-1	15644	10824	4040	3313	3797	3145	4820	4040	3313	0	0	0	0
0	16642	0	12061	11381	12578	11988	5423	4284	3877	0	0	0	0
1	15644	0	11989	11514	12929	12537	2254	1487	1366	0	0	0	0
2	14691	0	11585	11172	12590	12285	1576	980	891	0	0	0	0
3	13778	0	10990	10641	12030	11773	1237	735	673	0	0	0	0
4	12889	0	10340	10005	11354	11139	1024	580	522	0	0	0	0
5	12029	0	9683	9388	10725	10549	849	466	425	0	0	0	0
f	16642	0	13790	13373	16642	16642	12	0	0	0	0	0	0

16sec

pp	17884	0	11639	8904	0	0	17884	11639	8904	17884	11639	8904	3766
-5	12029	11284	585	403	506	350	745	585	403	0	0	0	0
-4	12889	11955	737	531	645	456	934	737	531	0	0	0	0
-3	13778	12392	1111	841	980	739	1386	1111	841	0	0	0	0
-2	14691	11996	2301	1912	2145	1781	2695	2301	1912	0	0	0	0
-1	15644	10820	4074	3362	3909	3244	4824	4074	3362	0	0	0	0
0	16642	0	12212	11548	12654	12113	5404	4285	3907	0	0	0	0
1	15644	0	12260	11773	13104	12745	2180	1480	1358	0	0	0	0
2	14691	0	11846	11451	12740	12445	1569	997	910	0	0	0	0
3	13778	0	11216	10870	12169	11914	1212	742	678	0	0	0	0
4	12889	0	10544	10225	11498	11284	969	579	528	0	0	0	0
5	12029	0	9878	9577	10840	10659	807	475	436	0	0	0	0
f	16642	0	14194	13763	16642	16642	0	0	0	0	0	0	0

jwcdg-3

1sect

pp	27061	0	10990	6570	0	0	27061	10990	6570	27061	10990	6570	11317
-5	12029	11097	621	312	381	193	932	621	312	0	0	0	0
-4	12889	11732	776	414	497	274	1157	776	414	0	0	0	0
-3	13778	12231	1042	609	711	437	1547	1042	609	0	0	0	0
-2	14691	12034	1918	1307	1474	1062	2657	1918	1307	0	0	0	0
-1	15644	10882	3308	2370	2725	2093	4762	3308	2370	0	0	0	0
0	16642	0	12731	11788	11672	11475	6776	5122	4587	0	0	0	0
1	15644	0	10370	9696	10987	10883	4972	1975	1737	0	0	0	0
2	14691	0	9471	8931	10333	10268	4467	1285	1103	0	0	0	0
3	13778	0	8901	8400	9789	9743	4022	977	824	0	0	0	0
4	12889	0	8291	7827	9228	9194	3664	760	619	0	0	0	0
5	12029	0	7718	7297	8699	8672	3324	602	487	0	0	0	0
f	16642	0	0	9789	16642	16642	3881	0	0	0	0	0	0

2sec

pp	25994	0	11987	8226	0	0	25994	11987	8226	25994	11987	8226	9667
-5	12029	11116	632	380	413	243	913	632	380	0	0	0	0
-4	12889	11729	816	513	558	353	1160	816	513	0	0	0	0
-3	13778	12174	1119	762	834	575	1604	1119	762	0	0	0	0
-2	14691	11761	2226	1702	1869	1452	2930	2226	1702	0	0	0	0
-1	15644	10265	4052	3172	3594	2897	5379	4052	3172	0	0	0	0
0	16642	0	12885	12063	12439	12047	6988	5215	4687	0	0	0	0
1	15644	0	11561	10974	12077	11852	4163	2042	1825	0	0	0	0
2	14691	0	10845	10390	11536	11378	3401	1346	1193	0	0	0	0
3	13778	0	10237	9831	10964	10833	2908	1016	894	0	0	0	0
4	12889	0	9597	9216	10380	10272	2513	793	681	0	0	0	0
5	12029	0	8992	8657	9817	9737	2206	641	554	0	0	0	0
f	16642	0	12200	11763	16642	16642	1952	0	0	0	0	0	0

4sec

pp	24383	0	12863	9459	0	0	24383	12863	9459	24383	12863	9459	7589
-5	12029	11095	666	434	518	325	934	666	434	0	0	0	0
-4	12889	11724	837	588	672	460	1165	837	588	0	0	0	0
-3	13778	12137	1209	896	1008	733	1641	1209	896	0	0	0	0
-2	14691	11620	2426	1930	2158	1727	3071	2426	1930	0	0	0	0
-1	15644	10016	4426	3541	4115	3341	5628	4426	3541	0	0	0	0
0	16642	0	13133	12391	13053	12525	7043	5283	4782	0	0	0	0
1	15644	0	12416	11923	12905	12609	3629	2096	1923	0	0	0	0
2	14691	0	11779	11379	12380	12154	2695	1372	1251	0	0	0	0
3	13778	0	11093	10755	11786	11610	2176	1024	934	0	0	0	0
4	12889	0	10408	10101	11158	11009	1828	796	715	0	0	0	0
5	12029	0	9749	9467	10529	10399	1556	639	575	0	0	0	0
f	16642	0	13663	13227	16642	16642	496	0	0	0	0	0	0

8sec

pp	22515	0	13224	9965	0	0	22515	13224	9965	22515	13224	9965	5829
-5	12029	11124	675	469	567	382	905	675	469	0	0	0	0
-4	12889	11737	857	601	738	509	1152	857	601	0	0	0	0
-3	13778	12133	1263	937	1097	810	1645	1263	937	0	0	0	0
-2	14691	11620	2514	2027	2307	1863	3071	2514	2027	0	0	0	0
-1	15644	10002	4593	3745	4374	3588	5642	4593	3745	0	0	0	0
0	16642	0	13335	12640	13414	12858	7040	5336	4847	0	0	0	0
1	15644	0	12965	12525	13464	13162	3296	2139	1976	0	0	0	0
2	14691	0	12330	11981	12873	12651	2307	1383	1270	0	0	0	0
3	13778	0	11619	11300	12218	12024	1802	1050	958	0	0	0	0
4	12889	0	10896	10602	11525	11358	1495	817	741	0	0	0	0
5	12029	0	10172	9912	10820	10676	1264	661	604	0	0	0	0
f	16642	0	14460	14086	16642	16642	52	0	0	0	0	0	0

16sec

pp	21800	0	13247	10108	0	0	21800	13247	10108	21800	13247	10108	5241
-5	12029	11137	672	476	579	397	892	672	476	0	0	0	0
-4	12889	11766	856	626	753	545	1123	856	626	0	0	0	0
-3	13778	12158	1250	961	1113	847	1620	1250	961	0	0	0	0
-2	14691	11618	2534	2065	2337	1914	3073	2534	2065	0	0	0	0
-1	15644	10019	4622	3790	4429	3663	5625	4622	3790	0	0	0	0
0	16642	0	13467	12780	13580	13020	7002	5387	4912	0	0	0	0
1	15644	0	13101	12666	13584	13298	3202	2133	1981	0	0	0	0
2	14691	0	12481	12124	13002	12780	2213	1396	1289	0	0	0	0
3	13778	0	11725	11410	12317	12121	1735	1044	965	0	0	0	0
4	12889	0	10999	10708	11602	11431	1400	813	746	0	0	0	0
5	12029	0	10297	10031	10897	10752	1167	651	599	0	0	0	0
f	16642	0	14699	14310	16642	16642	0	0	0	0	0	0	0

jwcdg-4

1sec

pp	27551	0	12196	8126	0	0	27551	12196	8126	27551	12196	8126	10506
-5	12029	11076	629	364	434	236	953	629	364	0	0	0	0
-4	12889	11668	829	484	589	333	1221	829	484	0	0	0	0
-3	13778	12102	1155	753	872	562	1676	1155	753	0	0	0	0
-2	14691	11638	2281	1705	1899	1437	3053	2281	1705	0	0	0	0
-1	15644	10065	4078	3122	3640	2853	5579	4078	3122	0	0	0	0
0	16642	0	12744	11910	12383	11858	7179	5256	4701	0	0	0	0
1	15644	0	11351	10754	11841	11570	4443	2093	1860	0	0	0	0
2	14691	0	10778	10291	11433	11214	3485	1379	1204	0	0	0	0
3	13778	0	10163	9731	10951	10765	2893	1019	880	0	0	0	0
4	12889	0	9559	9181	10450	10301	2421	783	675	0	0	0	0
5	12029	0	8946	8601	9903	9767	2103	617	529	0	0	0	0
f	16642	0	12644	12109	16642	16642	639	0	0	0	0	0	0

2sec

pp	25758	0	12610	8927	0	0	25758	12610	8927	25758	12610	8927	8970
-5	12029	11127	622	389	484	289	902	622	389	0	0	0	0
-4	12889	11716	828	528	662	419	1173	828	528	0	0	0	0
-3	13778	12134	1194	845	983	685	1644	1194	845	0	0	0	0
-2	14691	11631	2391	1844	2082	1615	3060	2391	1844	0	0	0	0
-1	15644	9984	4352	3445	4004	3212	5660	4352	3445	0	0	0	0
0	16642	0	12912	12157	12805	12247	7225	5289	4771	0	0	0	0
1	15644	0	12061	11547	12576	12249	3913	2071	1892	0	0	0	0
2	14691	0	11427	11017	12109	11863	2917	1362	1233	0	0	0	0
3	13778	0	10816	10450	11575	11363	2337	1033	935	0	0	0	0
4	12889	0	10148	9812	10956	10788	1951	791	713	0	0	0	0
5	12029	0	9543	9223	10386	10231	1657	645	583	0	0	0	0
f	16642	0	13541	13049	16642	16642	239	0	0	0	0	0	0

4sec

pp	23892	0	12953	9545	0	0	23892	12953	9545	23892	12953	9545	7117
-5	12029	11125	641	428	538	341	904	641	428	0	0	0	0
-4	12889	11712	856	588	721	479	1177	856	588	0	0	0	0
-3	13778	12105	1250	920	1069	786	1673	1250	920	0	0	0	0
-2	14691	11633	2464	1953	2237	1789	3058	2464	1953	0	0	0	0
-1	15644	10014	4475	3623	4235	3458	5630	4475	3623	0	0	0	0
0	16642	0	13083	12360	13153	12558	7123	5273	4775	0	0	0	0
1	15644	0	12620	12143	13118	12767	3503	2096	1927	0	0	0	0
2	14691	0	12012	11628	12593	12315	2529	1371	1253	0	0	0	0
3	13778	0	11358	11012	12029	11779	1982	1033	938	0	0	0	0
4	12889	0	10644	10334	11346	11153	1651	801	723	0	0	0	0
5	12029	0	9979	9683	10704	10533	1372	649	586	0	0	0	0
f	16642	0	14240	13795	16642	16642	64	0	0	0	0	0	0

8sec

pp	22745	0	13147	9787	0	0	22745	13147	9787	22745	13147	9787	6032
-5	12029	11116	668	438	577	378	913	668	438	0	0	0	0
-4	12889	11734	858	589	748	513	1155	858	589	0	0	0	0
-3	13778	12131	1245	933	1104	829	1647	1245	933	0	0	0	0
-2	14691	11609	2516	2017	2341	1883	3082	2516	2017	0	0	0	0
-1	15644	9974	4585	3736	4416	3613	5670	4585	3736	0	0	0	0
0	16642	0	13328	12633	13443	12880	7104	5386	4893	0	0	0	0
1	15644	0	12937	12469	13374	13073	3388	2162	1993	0	0	0	0
2	14691	0	12287	11933	12816	12590	2370	1382	1268	0	0	0	0
3	13778	0	11596	11272	12166	11958	1842	1050	961	0	0	0	0
4	12889	0	10878	10583	11476	11301	1512	808	740	0	0	0	0
5	12029	0	10183	9907	10806	10652	1250	652	598	0	0	0	0
f	16642	0	14584	14160	16642	16642	7	0	0	0	0	0	0

16sec

pp	21892	0	13183	10141	0	0	21892	13183	10141	21892	13183	10141	5379
-5	12029	11139	673	476	583	407	890	673	476	0	0	0	0
-4	12889	11757	853	630	761	550	1132	853	630	0	0	0	0
-3	13778	12143	1259	966	1124	861	1635	1259	966	0	0	0	0
-2	14691	11621	2530	2079	2365	1942	3070	2530	2079	0	0	0	0
-1	15644	10018	4620	3797	4456	3684	5626	4620	3797	0	0	0	0
0	16642	0	13406	12712	13544	12976	7075	5392	4904	0	0	0	0
1	15644	0	13066	12621	13548	13258	3278	2172	2015	0	0	0	0
2	14691	0	12414	12074	12946	12730	2282	1399	1293	0	0	0	0
3	13778	0	11694	11392	12270	12084	1754	1053	978	0	0	0	0
4	12889	0	10962	10692	11561	11389	1420	812	756	0	0	0	0
5	12029	0	10271	10015	10869	10715	1183	654	609	0	0	0	0
f	16642	0	14713	14316	16642	16642	9	0	0	0	0	0	0

Evaluation NEGRA-3-10

jwcdg-0

16sec

pp	1020	0	789	603	0	0	1020	789	603	1020	789	603	169
-5	352	326	23	13	22	12	26	23	13	0	0	0	0
-4	488	434	47	29	45	27	54	47	29	0	0	0	0
-3	640	545	83	63	77	57	95	83	63	0	0	0	0
-2	809	586	207	156	196	146	223	207	156	0	0	0	0
-1	978	547	407	330	393	321	431	407	330	0	0	0	0
0	1147	0	932	892	965	926	370	281	263	0	0	0	0
1	978	0	828	788	874	849	123	64	60	0	0	0	0
2	809	0	695	672	742	727	81	36	34	0	0	0	0
3	640	0	574	552	595	583	44	16	16	0	0	0	0
4	488	0	443	429	460	454	20	4	4	0	0	0	0
5	352	0	325	310	339	336	8	0	0	0	0	0	0
f	1147	0	1069	1037	1147	1147	0	0	0	0	0	0	0

8sec

pp	1514	0	1145	840	0	0	1514	1145	840	1514	1145	840	290
-5	506	468	35	20	34	19	38	35	20	0	0	0	0
-4	708	633	68	43	64	37	75	68	43	0	0	0	0
-3	939	805	120	85	108	74	134	120	85	0	0	0	0
-2	1194	870	304	220	283	199	324	304	220	0	0	0	0
-1	1449	822	592	458	568	440	627	592	458	0	0	0	0
0	1704	0	1358	1296	1405	1351	559	398	371	0	0	0	0
1	1449	0	1202	1148	1272	1240	195	86	81	0	0	0	0
2	1194	0	1009	973	1078	1054	120	43	40	0	0	0	0
3	939	0	828	796	866	847	67	17	17	0	0	0	0
4	708	0	634	609	663	653	33	4	4	0	0	0	0
5	506	0	469	443	482	475	15	0	0	0	0	0	0
f	1704	0	1587	1529	1704	1704	0	0	0	0	0	0	0

4sec

pp	1513	0	1140	841	0	0	1513	1140	841	1513	1140	841	293
-5	506	469	34	20	33	19	37	34	20	0	0	0	0
-4	708	633	68	43	64	37	75	68	43	0	0	0	0
-3	939	808	117	85	105	74	131	117	85	0	0	0	0
-2	1194	869	306	220	281	194	325	306	220	0	0	0	0
-1	1449	823	589	459	564	438	626	589	459	0	0	0	0
0	1704	0	1351	1290	1401	1345	557	397	370	0	0	0	0
1	1449	0	1196	1141	1271	1236	195	85	79	0	0	0	0
2	1194	0	1008	973	1083	1057	118	43	40	0	0	0	0
3	939	0	821	792	865	847	67	17	17	0	0	0	0
4	708	0	630	607	664	655	32	5	5	0	0	0	0
5	506	0	465	441	483	478	14	0	0	0	0	0	0
f	1704	0	1573	1514	1704	1704	0	0	0	0	0	0	0

2sec

pp	1563	0	1143	828	0	0	1563	1143	828	1563	1143	828	334
-5	506	468	35	20	33	19	38	35	20	0	0	0	0
-4	708	631	69	43	64	37	77	69	43	0	0	0	0
-3	939	808	118	84	107	73	131	118	84	0	0	0	0
-2	1194	864	310	218	286	194	330	310	218	0	0	0	0
-1	1449	823	584	449	557	428	626	584	449	0	0	0	0
0	1704	0	1337	1277	1397	1338	554	397	370	0	0	0	0
1	1449	0	1176	1123	1258	1222	198	84	79	0	0	0	0
2	1194	0	991	956	1065	1037	124	41	38	0	0	0	0
3	939	0	802	773	845	827	80	17	17	0	0	0	0
4	708	0	616	592	651	643	44	5	5	0	0	0	0
5	506	0	453	429	468	464	26	0	0	0	0	0	0
f	1704	0	1573	1515	1704	1704	0	0	0	0	0	0	0

1sec

pp	1590	0	1110	794	0	0	1590	1110	794	1590	1110	794	382
-5	506	466	37	20	34	17	40	37	20	0	0	0	0
-4	708	633	67	40	60	31	75	67	40	0	0	0	0
-3	939	814	111	77	94	59	125	111	77	0	0	0	0
-2	1194	870	300	212	268	181	324	300	212	0	0	0	0
-1	1449	834	566	431	535	408	615	566	431	0	0	0	0
0	1704	0	1302	1241	1369	1307	558	396	370	0	0	0	0
1	1449	0	1120	1068	1219	1183	222	83	77	0	0	0	0
2	1194	0	944	900	1044	1015	139	42	38	0	0	0	0
3	939	0	767	736	834	814	88	16	15	0	0	0	0
4	708	0	594	568	639	631	46	5	5	0	0	0	0
5	506	0	433	409	463	459	29	0	0	0	0	0	0
f	1704	0	1524	1463	1704	1704	0	0	0	0	0	0	0

jwcdg-2

1sec

pp	1580	0	1137	820	0	0	1580	1137	820	1580	1137	820	356
-5	506	467	36	20	33	18	39	36	20	0	0	0	0
-4	708	632	69	43	62	37	76	69	43	0	0	0	0
-3	939	806	121	83	107	70	133	121	83	0	0	0	0
-2	1194	872	302	216	275	189	322	302	216	0	0	0	0
-1	1449	826	578	445	549	424	623	578	445	0	0	0	0
0	1704	0	1342	1275	1387	1334	570	405	373	0	0	0	0
1	1449	0	1174	1116	1250	1212	207	84	77	0	0	0	0
2	1194	0	999	952	1070	1038	126	44	38	0	0	0	0
3	939	0	810	774	852	833	76	17	16	0	0	0	0
4	708	0	626	605	659	651	35	5	4	0	0	0	0
5	506	0	463	444	479	475	17	0	0	0	0	0	0
f	1704	0	1568	1504	1704	1704	0	0	0	0	0	0	0

2sec

pp	1545	0	1146	824	0	0	1545	1146	824	1545	1146	824	318
-5	506	466	37	17	36	16	40	37	17	0	0	0	0
-4	708	631	70	41	66	35	77	70	41	0	0	0	0
-3	939	806	119	83	108	72	133	119	83	0	0	0	0
-2	1194	872	303	216	279	193	322	303	216	0	0	0	0
-1	1449	821	590	454	565	432	628	590	454	0	0	0	0
0	1704	0	1352	1281	1406	1351	562	398	367	0	0	0	0
1	1449	0	1190	1131	1262	1227	203	83	77	0	0	0	0
2	1194	0	1001	958	1071	1041	126	43	40	0	0	0	0
3	939	0	819	787	862	842	70	17	17	0	0	0	0
4	708	0	634	609	661	650	34	4	4	0	0	0	0
5	506	0	469	444	481	474	17	0	0	0	0	0	0
f	1704	0	1579	1515	1704	1704	0	0	0	0	0	0	0

4sec

pp	1524	0	1149	821	0	0	1524	1149	821	1524	1149	821	296
-5	506	466	37	18	36	17	40	37	18	0	0	0	0
-4	708	631	71	40	67	34	77	71	40	0	0	0	0
-3	939	805	120	83	109	72	134	120	83	0	0	0	0
-2	1194	873	302	214	279	191	321	302	214	0	0	0	0
-1	1449	820	592	455	568	437	629	592	455	0	0	0	0
0	1704	0	1363	1296	1411	1358	560	402	370	0	0	0	0
1	1449	0	1202	1146	1275	1244	194	85	80	0	0	0	0
2	1194	0	1012	970	1079	1052	123	43	40	0	0	0	0
3	939	0	827	793	867	848	68	17	17	0	0	0	0
4	708	0	633	610	663	655	34	4	4	0	0	0	0
5	506	0	468	445	480	476	19	0	0	0	0	0	0
f	1704	0	1583	1522	1704	1704	0	0	0	0	0	0	0

8sec

pp	1528	0	1157	836	0	0	1528	1157	836	1528	1157	836	292
-5	506	466	37	17	36	16	40	37	17	0	0	0	0
-4	708	630	72	43	68	37	78	72	43	0	0	0	0
-3	939	803	122	83	111	72	136	122	83	0	0	0	0
-2	1194	871	304	221	279	196	323	304	221	0	0	0	0
-1	1449	817	595	460	571	441	632	595	460	0	0	0	0
0	1704	0	1358	1290	1403	1350	564	401	368	0	0	0	0
1	1449	0	1200	1141	1271	1237	199	86	81	0	0	0	0
2	1194	0	1013	965	1083	1053	120	43	40	0	0	0	0
3	939	0	825	790	866	845	66	17	17	0	0	0	0
4	708	0	635	611	663	654	33	4	4	0	0	0	0
5	506	0	466	446	478	473	20	0	0	0	0	0	0
f	1704	0	1575	1509	1704	1704	0	0	0	0	0	0	0

16sec

pp	1539	0	1149	836	0	0	1539	1149	836	1539	1149	836	300
-5	506	466	37	20	36	19	40	37	20	0	0	0	0
-4	708	629	71	44	67	38	79	71	44	0	0	0	0
-3	939	804	121	82	110	71	135	121	82	0	0	0	0
-2	1194	872	301	216	277	191	322	301	216	0	0	0	0
-1	1449	816	589	461	565	440	633	589	461	0	0	0	0
0	1704	0	1361	1294	1411	1356	566	403	374	0	0	0	0
1	1449	0	1202	1150	1273	1245	202	87	82	0	0	0	0
2	1194	0	1013	971	1080	1056	125	44	41	0	0	0	0
3	939	0	825	792	864	848	69	17	17	0	0	0	0
4	708	0	637	617	666	658	31	4	4	0	0	0	0
5	506	0	468	447	481	474	17	0	0	0	0	0	0
f	1704	0	1580	1516	1704	1704	0	0	0	0	0	0	0

jwcdg-3

1sec

pp	1746	0	1210	858	0	0	1746	1210	858	1746	1210	858	420
-5	506	467	36	17	35	15	39	36	17	0	0	0	0
-4	708	620	79	42	73	35	88	79	42	0	0	0	0
-3	939	799	125	83	112	71	140	125	83	0	0	0	0
-2	1194	855	311	221	287	198	339	311	221	0	0	0	0
-1	1449	758	630	485	603	466	691	630	485	0	0	0	0
0	1704	0	1415	1338	1441	1378	665	476	437	0	0	0	0
1	1449	0	1214	1161	1269	1241	267	119	111	0	0	0	0
2	1194	0	1018	981	1080	1060	151	58	54	0	0	0	0
3	939	0	818	792	862	851	90	25	25	0	0	0	0
4	708	0	628	609	662	657	41	7	7	0	0	0	0
5	506	0	465	448	483	478	19	0	0	0	0	0	0
f	1704	0	1584	1535	1704	1704	0	0	0	0	0	0	0

2sec

pp	1682	0	1208	868	0	0	1682	1208	868	1682	1208	868	366
-5	506	467	36	18	36	16	39	36	18	0	0	0	0
-4	708	622	78	47	75	40	86	78	47	0	0	0	0
-3	939	797	127	88	114	75	142	127	88	0	0	0	0
-2	1194	858	309	219	286	195	336	309	219	0	0	0	0
-1	1449	765	629	484	604	463	684	629	484	0	0	0	0
0	1704	0	1430	1351	1469	1408	654	475	433	0	0	0	0
1	1449	0	1238	1186	1300	1275	242	118	110	0	0	0	0
2	1194	0	1033	996	1102	1083	139	58	53	0	0	0	0
3	939	0	837	807	878	868	79	26	26	0	0	0	0
4	708	0	641	618	671	664	36	7	6	0	0	0	0
5	506	0	471	453	488	481	19	1	1	0	0	0	0
f	1704	0	1597	1546	1704	1704	0	0	0	0	0	0	0

4sec

pp	1643	0	1201	862	0	0	1643	1201	862	1643	1201	862	333
-5	506	468	35	18	35	16	38	35	18	0	0	0	0
-4	708	623	77	48	74	41	85	77	48	0	0	0	0
-3	939	798	125	89	112	77	141	125	89	0	0	0	0
-2	1194	858	308	219	287	195	336	308	219	0	0	0	0
-1	1449	766	629	480	605	460	683	629	480	0	0	0	0
0	1704	0	1431	1360	1468	1409	645	470	433	0	0	0	0
1	1449	0	1243	1192	1304	1278	240	118	109	0	0	0	0
2	1194	0	1034	1000	1104	1085	138	58	55	0	0	0	0
3	939	0	839	811	882	871	77	27	27	0	0	0	0
4	708	0	643	625	675	668	32	7	7	0	0	0	0
5	506	0	471	454	488	481	18	1	1	0	0	0	0
f	1704	0	1596	1546	1704	1704	0	0	0	0	0	0	0

8sec

pp	1636	0	1204	874	0	0	1636	1204	874	1636	1204	874	328
-5	506	467	36	20	35	18	39	36	20	0	0	0	0
-4	708	625	74	49	70	42	83	74	49	0	0	0	0
-3	939	800	124	86	112	74	139	124	86	0	0	0	0
-2	1194	858	310	221	288	196	336	310	221	0	0	0	0
-1	1449	765	633	485	606	464	684	633	485	0	0	0	0
0	1704	0	1427	1359	1464	1406	645	468	433	0	0	0	0
1	1449	0	1245	1195	1306	1282	238	118	111	0	0	0	0
2	1194	0	1032	998	1101	1083	140	57	54	0	0	0	0
3	939	0	840	813	880	870	79	27	27	0	0	0	0
4	708	0	646	626	674	668	34	7	7	0	0	0	0
5	506	0	470	453	487	481	20	1	1	0	0	0	0
f	1704	0	1599	1549	1704	1704	0	0	0	0	0	0	0

16sec

pp	1628	0	1196	868	0	0	1628	1196	868	1628	1196	868	323
-5	506	468	35	19	35	17	38	35	19	0	0	0	0
-4	708	624	74	45	71	38	84	74	45	0	0	0	0
-3	939	798	124	84	111	72	141	124	84	0	0	0	0
-2	1194	862	303	221	282	197	332	303	221	0	0	0	0
-1	1449	768	631	485	607	464	681	631	485	0	0	0	0
0	1704	0	1434	1363	1470	1416	643	471	436	0	0	0	0
1	1449	0	1246	1197	1308	1287	238	118	111	0	0	0	0
2	1194	0	1033	998	1101	1084	139	57	54	0	0	0	0
3	939	0	839	811	879	869	79	27	27	0	0	0	0
4	708	0	645	626	675	669	32	7	7	0	0	0	0
5	506	0	471	454	488	482	18	1	1	0	0	0	0
f	1704	0	1599	1546	1704	1704	0	0	0	0	0	0	0

Evaluation NEGRA-3-20

jwcdg-0

16sec

pp	4729	0	3443	2721	0	0	4729	3443	2721	4729	3443	2721	774
-5	2832	2624	175	125	162	120	208	175	125	0	0	0	0
-4	3203	2920	245	170	229	162	283	245	170	0	0	0	0
-3	3590	3165	345	267	330	257	425	345	267	0	0	0	0
-2	3994	3121	773	642	736	618	873	773	642	0	0	0	0
-1	4398	2839	1384	1166	1371	1168	1559	1384	1166	0	0	0	0
0	4802	0	3740	3569	3911	3752	1526	1237	1159	0	0	0	0
1	4398	0	3591	3464	3842	3745	541	353	330	0	0	0	0
2	3994	0	3354	3261	3613	3540	363	220	205	0	0	0	0
3	3590	0	3063	2966	3308	3240	255	147	134	0	0	0	0
4	3203	0	2753	2673	2980	2923	186	101	91	0	0	0	0
5	2832	0	2455	2384	2664	2617	127	74	71	0	0	0	0
f	4802	0	4315	4193	4802	4802	0	0	0	0	0	0	0

8sec

pp	6736	0	4699	3595	0	0	6736	4699	3595	6736	4699	3595	1319
-5	3870	3587	234	157	209	138	283	234	157	0	0	0	0
-4	4394	4012	326	226	291	194	382	326	226	0	0	0	0
-3	4947	4368	481	361	431	316	579	481	361	0	0	0	0
-2	5524	4342	1042	841	970	777	1182	1042	841	0	0	0	0
-1	6101	3946	1900	1558	1828	1509	2155	1900	1558	0	0	0	0
0	6678	0	5088	4843	5305	5096	2165	1704	1588	0	0	0	0
1	6101	0	4889	4688	5238	5089	805	504	469	0	0	0	0
2	5524	0	4537	4397	4905	4800	530	301	282	0	0	0	0
3	4947	0	4119	3986	4466	4381	380	194	179	0	0	0	0
4	4394	0	3704	3586	4032	3959	281	144	130	0	0	0	0
5	3870	0	3290	3186	3578	3516	211	106	101	0	0	0	0
f	6678	0	5900	5703	6678	6678	1	0	0	0	0	0	0

4sec

-5	3870	3583	238	155	203	128	287	238	155	0	0	0	0
-4	4394	3999	334	221	280	174	395	334	221	0	0	0	0
-3	4947	4361	483	350	406	284	586	483	350	0	0	0	0
-2	5524	4331	1047	823	945	737	1193	1047	823	0	0	0	0
-1	6101	3948	1868	1522	1767	1438	2153	1868	1522	0	0	0	0
0	6678	0	5011	4753	5294	5066	2142	1691	1577	0	0	0	0
1	6101	0	4748	4542	5206	5037	812	488	451	0	0	0	0
2	5524	0	4426	4274	4888	4768	548	297	277	0	0	0	0
3	4947	0	4027	3887	4456	4352	393	187	168	0	0	0	0
4	4394	0	3612	3488	4001	3921	295	133	118	0	0	0	0
5	3870	0	3205	3098	3557	3495	233	98	92	0	0	0	0
f	6678	0	5735	5542	6678	6678	5	0	0	0	0	0	0

2sec

pp	7165	0	4614	3321	0	0	7165	4614	3321	7165	4614	3321	1723
-5	3870	3585	230	140	178	106	285	230	140	0	0	0	0
-4	4394	4004	325	203	253	154	390	325	203	0	0	0	0
-3	4947	4358	477	323	385	254	589	477	323	0	0	0	0
-2	5524	4343	1029	784	898	673	1181	1029	784	0	0	0	0
-1	6101	3955	1832	1447	1702	1343	2146	1832	1447	0	0	0	0
0	6678	0	4844	4578	5257	5015	2124	1652	1540	0	0	0	0
1	6101	0	4537	4299	5165	4993	856	483	448	0	0	0	0
2	5524	0	4202	4023	4828	4700	584	280	259	0	0	0	0
3	4947	0	3819	3672	4382	4282	441	179	164	0	0	0	0
4	4394	0	3430	3286	3941	3863	343	134	118	0	0	0	0
5	3870	0	3063	2940	3510	3453	279	98	89	0	0	0	0
f	6678	0	5470	5231	6678	6678	38	0	0	0	0	0	0

1sec

pp	7435	0	4366	2955	0	0	7435	4366	2955	7435	4366	2955	2209
-5	3870	3574	234	128	182	85	296	234	128	0	0	0	0
-4	4394	4011	310	182	229	122	383	310	182	0	0	0	0
-3	4947	4387	449	288	333	198	560	449	288	0	0	0	0
-2	5524	4410	960	707	806	581	1114	960	707	0	0	0	0
-1	6101	4151	1636	1274	1482	1152	1950	1636	1274	0	0	0	0
0	6678	0	4590	4302	5260	5029	2021	1566	1461	0	0	0	0
1	6101	0	4103	3860	5039	4891	921	458	414	0	0	0	0
2	5524	0	3797	3595	4725	4617	662	267	238	0	0	0	0
3	4947	0	3469	3299	4314	4238	518	178	155	0	0	0	0
4	4394	0	3131	2983	3889	3838	402	125	111	0	0	0	0
5	3870	0	2789	2659	3469	3427	334	92	85	0	0	0	0
f	6678	0	5040	4785	6678	6678	55	0	0	0	0	0	0

parallel

1sec

pp	7520	0	4657	3341	0	0	7520	4657	3341	7520	4657	3341	2018
-5	3870	3580	237	142	183	111	290	237	142	0	0	0	0
-4	4394	3997	324	201	264	165	397	324	201	0	0	0	0
-3	4947	4353	490	333	393	265	594	490	333	0	0	0	0
-2	5524	4337	1023	784	887	679	1187	1023	784	0	0	0	0
-1	6101	3917	1860	1473	1727	1387	2184	1860	1473	0	0	0	0
0	6678	0	4873	4591	5140	4909	2251	1696	1565	0	0	0	0
1	6101	0	4559	4325	5050	4882	924	494	452	0	0	0	0
2	5524	0	4224	4036	4734	4602	654	304	270	0	0	0	0
3	4947	0	3858	3689	4340	4225	477	196	171	0	0	0	0
4	4394	0	3465	3326	3934	3845	339	134	116	0	0	0	0
5	3870	0	3084	2965	3503	3433	271	96	87	0	0	0	0
f	6678	0	5511	5282	6678	6678	7	0	0	0	0	0	0

2sec

pp	7003	0	4689	3473	0	0	7003	4689	3473	7003	4689	3473	1552
-5	3870	3589	232	152	197	127	281	232	152	0	0	0	0
-4	4394	4010	316	214	276	184	384	316	214	0	0	0	0
-3	4947	4358	478	343	422	301	589	478	343	0	0	0	0
-2	5524	4337	1046	810	949	738	1187	1046	810	0	0	0	0
-1	6101	3925	1895	1533	1801	1470	2176	1895	1533	0	0	0	0
0	6678	0	5001	4716	5226	5001	2205	1697	1563	0	0	0	0
1	6101	0	4766	4537	5147	4984	845	486	444	0	0	0	0
2	5524	0	4424	4246	4826	4701	568	301	272	0	0	0	0
3	4947	0	4024	3877	4425	4317	397	194	171	0	0	0	0
4	4394	0	3617	3484	3980	3889	291	137	119	0	0	0	0
5	3870	0	3220	3100	3546	3472	219	96	86	0	0	0	0
f	6678	0	5792	5599	6678	6678	0	0	0	0	0	0	0

4sec

pp	6695	0	4688	3542	0	0	6695	4688	3542	6695	4688	3542	1275
-5	3870	3584	233	153	217	140	286	233	153	0	0	0	0
-4	4394	4002	334	216	312	199	392	334	216	0	0	0	0
-3	4947	4372	474	350	432	323	575	474	350	0	0	0	0
-2	5524	4337	1047	833	981	787	1187	1047	833	0	0	0	0
-1	6101	3940	1899	1557	1835	1527	2161	1899	1557	0	0	0	0
0	6678	0	5130	4875	5291	5085	2181	1725	1603	0	0	0	0
1	6101	0	4925	4724	5232	5098	793	499	463	0	0	0	0
2	5524	0	4577	4411	4897	4794	518	309	280	0	0	0	0
3	4947	0	4129	3984	4444	4354	374	201	181	0	0	0	0
4	4394	0	3710	3584	4005	3932	267	146	125	0	0	0	0
5	3870	0	3304	3189	3581	3522	189	100	90	0	0	0	0
f	6678	0	5944	5752	6678	6678	1	0	0	0	0	0	0

8sec

pp	6573	0	4741	3602	0	0	6573	4741	3602	6573	4741	3602	1121
-5	3870	3584	238	156	223	147	286	238	156	0	0	0	0
-4	4394	4003	336	226	315	211	391	336	226	0	0	0	0
-3	4947	4361	481	353	451	333	586	481	353	0	0	0	0
-2	5524	4337	1055	850	997	809	1187	1055	850	0	0	0	0
-1	6101	3927	1916	1572	1867	1555	2174	1916	1572	0	0	0	0
0	6678	0	5162	4908	5333	5126	2173	1724	1600	0	0	0	0
1	6101	0	4976	4777	5292	5145	780	497	460	0	0	0	0
2	5524	0	4638	4476	4959	4851	510	309	283	0	0	0	0
3	4947	0	4209	4069	4514	4423	357	204	185	0	0	0	0
4	4394	0	3781	3658	4051	3978	261	148	130	0	0	0	0
5	3870	0	3352	3239	3591	3534	190	99	90	0	0	0	0
f	6678	0	6022	5836	6678	6678	0	0	0	0	0	0	0

16sec

pp	6583	0	4742	3600	0	0	6583	4742	3600	6583	4742	3600	1105
-5	3870	3583	238	153	223	149	287	238	153	0	0	0	0
-4	4394	4007	329	222	312	208	387	329	222	0	0	0	0
-3	4947	4362	483	356	457	336	585	483	356	0	0	0	0
-2	5524	4326	1053	847	1003	805	1198	1053	847	0	0	0	0
-1	6101	3917	1920	1582	1875	1562	2184	1920	1582	0	0	0	0
0	6678	0	5193	4943	5365	5167	2179	1735	1619	0	0	0	0
1	6101	0	5008	4814	5294	5167	784	504	469	0	0	0	0
2	5524	0	4652	4499	4967	4870	522	311	287	0	0	0	0
3	4947	0	4223	4077	4526	4435	363	202	180	0	0	0	0
4	4394	0	3794	3678	4068	4000	259	145	129	0	0	0	0
5	3870	0	3369	3264	3604	3550	189	103	96	0	0	0	0
f	6678	0	6064	5877	6678	6678	0	0	0	0	0	0	0

jwcdg-4

1sec

pp	8877	0	5101	3636	0	0	8877	5101	3636	8877	5101	3636	2657
-5	3870	3540	253	150	206	115	330	253	150	0	0	0	0
-4	4394	3935	364	220	298	171	459	364	220	0	0	0	0
-3	4947	4264	539	368	446	300	683	539	368	0	0	0	0
-2	5500	4181	1102	848	983	754	1319	1102	848	0	0	0	0
-1	6053	3604	2028	1605	1903	1522	2449	2028	1605	0	0	0	0
0	6606	0	5236	4973	5278	5058	2775	2032	1882	0	0	0	0
1	6053	0	4799	4581	5019	4889	1359	699	648	0	0	0	0
2	5500	0	4430	4264	4689	4586	932	427	393	0	0	0	0
3	4947	0	4023	3885	4301	4218	690	283	262	0	0	0	0
4	4394	0	3616	3492	3879	3816	508	188	172	0	0	0	0
5	3870	0	3211	3102	3459	3404	397	133	123	0	0	0	0
f	6606	0	5727	5508	6606	6606	0	0	0	0	0	0	0

2sec

pp	8336	0	5132	3750	0	0	8336	5132	3750	8336	5132	3750	2178
-5	3870	3540	250	150	228	136	330	250	150	0	0	0	0
-4	4394	3943	360	226	329	205	451	360	226	0	0	0	0
-3	4947	4282	530	378	470	334	665	530	378	0	0	0	0
-2	5524	4222	1112	858	1023	790	1302	1112	858	0	0	0	0
-1	6101	3632	2095	1679	1995	1621	2469	2095	1679	0	0	0	0
0	6678	0	5384	5116	5477	5260	2767	2067	1909	0	0	0	0
1	6101	0	5076	4882	5299	5175	1201	709	664	0	0	0	0
2	5524	0	4636	4493	4892	4797	812	435	406	0	0	0	0
3	4947	0	4192	4054	4437	4354	580	295	267	0	0	0	0
4	4394	0	3750	3625	3966	3900	436	203	185	0	0	0	0
5	3870	0	3328	3219	3532	3476	339	147	137	0	0	0	0
f	6678	0	6037	5851	6678	6678	0	0	0	0	0	0	0

4sec

pp	7904	0	5178	3842	0	0	7904	5178	3842	7904	5178	3842	1753
-5	3870	3546	254	164	238	149	324	254	164	0	0	0	0
-4	4394	3937	370	248	345	228	457	370	248	0	0	0	0
-3	4947	4280	534	389	482	356	667	534	389	0	0	0	0
-2	5524	4223	1122	879	1052	821	1301	1122	879	0	0	0	0
-1	6101	3661	2093	1696	2023	1655	2440	2093	1696	0	0	0	0
0	6678	0	5427	5174	5543	5324	2721	2042	1898	0	0	0	0
1	6101	0	5161	4969	5386	5262	1137	708	656	0	0	0	0
2	5524	0	4722	4585	4964	4873	748	426	400	0	0	0	0
3	4947	0	4265	4137	4506	4426	530	283	261	0	0	0	0
4	4394	0	3818	3707	4021	3964	392	199	181	0	0	0	0
5	3870	0	3383	3281	3573	3521	295	141	131	0	0	0	0
f	6678	0	6125	5956	6678	6678	0	0	0	0	0	0	0

8sec

pp	7633	0	5196	3895	0	0	7633	5196	3895	7633	5196	3895	1515
-5	3870	3548	258	169	239	157	322	258	169	0	0	0	0
-4	4394	3951	363	250	337	228	443	363	250	0	0	0	0
-3	4947	4286	536	402	494	370	661	536	402	0	0	0	0
-2	5524	4219	1138	900	1078	845	1305	1138	900	0	0	0	0
-1	6101	3649	2117	1720	2051	1679	2452	2117	1720	0	0	0	0
0	6678	0	5492	5245	5604	5395	2702	2075	1934	0	0	0	0
1	6101	0	5214	5029	5430	5317	1108	716	671	0	0	0	0
2	5524	0	4760	4632	5004	4927	725	433	408	0	0	0	0
3	4947	0	4305	4182	4529	4456	508	293	271	0	0	0	0
4	4394	0	3863	3754	4057	4002	376	209	190	0	0	0	0
5	3870	0	3417	3325	3600	3553	272	145	136	0	0	0	0
f	6678	0	6143	5974	6678	6678	0	0	0	0	0	0	0

16sec

pp	7461	0	5170	3949	0	0	7461	5170	3949	7461	5170	3949	1409
-5	3870	3547	264	180	248	167	323	264	180	0	0	0	0
-4	4394	3958	360	259	342	240	436	360	259	0	0	0	0
-3	4947	4295	533	403	497	373	652	533	403	0	0	0	0
-2	5500	4209	1118	904	1061	853	1291	1118	904	0	0	0	0
-1	6053	3656	2089	1716	2036	1686	2397	2089	1716	0	0	0	0
0	6606	0	5458	5209	5571	5367	2675	2066	1924	0	0	0	0
1	6053	0	5213	5035	5429	5324	1088	726	680	0	0	0	0
2	5500	0	4757	4635	5003	4934	713	433	410	0	0	0	0
3	4947	0	4323	4206	4550	4493	497	290	273	0	0	0	0
4	4394	0	3862	3761	4067	4019	351	200	188	0	0	0	0
5	3870	0	3425	3335	3614	3578	257	142	135	0	0	0	0
f	6606	0	6082	5922	6606	6606	0	0	0	0	0	0	0

Evaluation NEGRA-3-30

jwcdg-0

16sec

pp	9001	0	6237	4976	0	0	9001	6237	4976	9001	6237	4976	1687
-5	5940	5551	314	234	265	197	389	314	234	0	0	0	0
-4	6466	5950	428	314	370	268	516	428	314	0	0	0	0
-3	7008	6257	614	488	550	429	751	614	488	0	0	0	0
-2	7567	6056	1311	1111	1210	1027	1511	1311	1111	0	0	0	0
-1	8126	5485	2278	1926	2200	1868	2641	2278	1926	0	0	0	0
0	8685	0	6491	6165	6905	6610	2745	2249	2077	0	0	0	0
1	8126	0	6413	6188	7004	6823	1050	720	670	0	0	0	0
2	7567	0	6160	5983	6762	6618	722	468	431	0	0	0	0
3	7008	0	5767	5594	6349	6222	553	336	308	0	0	0	0
4	6466	0	5347	5196	5917	5802	433	258	236	0	0	0	0
5	5940	0	4957	4822	5485	5387	343	208	198	0	0	0	0
f	8685	0	7444	7224	8685	8685	5	0	0	0	0	0	0

8sec

pp	13394	0	8746	6743	0	0	13394	8746	6743	13394	8746	6743	3017
-5	8405	7838	458	315	366	235	567	458	315	0	0	0	0
-4	9157	8434	592	422	486	323	723	592	422	0	0	0	0
-3	9938	8885	860	660	714	519	1053	860	660	0	0	0	0
-2	10743	8648	1794	1478	1612	1308	2095	1794	1478	0	0	0	0
-1	11548	7814	3169	2602	2979	2451	3734	3169	2602	0	0	0	0
0	12353	0	9028	8534	9579	9149	3982	3184	2919	0	0	0	0
1	11548	0	8874	8498	9741	9443	1618	1051	969	0	0	0	0
2	10743	0	8514	8222	9375	9155	1125	683	624	0	0	0	0
3	9938	0	7962	7690	8809	8624	873	493	449	0	0	0	0
4	9157	0	7392	7147	8221	8056	702	391	351	0	0	0	0
5	8405	0	6818	6606	7613	7475	584	313	289	0	0	0	0
f	12353	0	10256	9879	12353	12353	25	0	0	0	0	0	0

4sec

pp	13785	0	8583	6391	0	0	13785	8583	6391	13785	8583	6391	3442
-5	8405	7842	445	291	330	206	563	445	291	0	0	0	0
-4	9157	8420	591	394	458	290	737	591	394	0	0	0	0
-3	9938	8872	855	628	676	479	1066	855	628	0	0	0	0
-2	10743	8636	1782	1412	1554	1221	2107	1782	1412	0	0	0	0
-1	11548	7834	3099	2512	2864	2321	3714	3099	2512	0	0	0	0
0	12353	0	8898	8370	9670	9231	3904	3156	2889	0	0	0	0
1	11548	0	8560	8157	9733	9418	1629	1011	920	0	0	0	0
2	10743	0	8189	7865	9352	9112	1182	650	585	0	0	0	0
3	9938	0	7683	7392	8820	8614	923	467	411	0	0	0	0
4	9157	0	7147	6878	8214	8049	750	366	319	0	0	0	0
5	8405	0	6602	6368	7608	7469	631	294	267	0	0	0	0
f	12353	0	9884	9509	12353	12353	59	0	0	0	0	0	0

2sec

pp	14247	0	8311	5893	0	0	14247	8311	5893	14247	8311	5893	4010
-5	8405	7848	429	259	291	169	557	429	259	0	0	0	0
-4	9157	8434	563	353	404	244	723	563	353	0	0	0	0
-3	9938	8896	813	564	610	407	1042	813	564	0	0	0	0
-2	10743	8700	1700	1318	1429	1097	2043	1700	1318	0	0	0	0
-1	11548	7942	2937	2319	2668	2107	3606	2937	2319	0	0	0	0
0	12353	0	8563	8028	9686	9242	3794	3057	2807	0	0	0	0
1	11548	0	8078	7643	9662	9361	1705	992	899	0	0	0	0
2	10743	0	7685	7337	9270	9031	1249	606	542	0	0	0	0
3	9938	0	7222	6916	8738	8553	988	430	381	0	0	0	0
4	9157	0	6724	6423	8137	7985	816	331	286	0	0	0	0
5	8405	0	6247	5980	7561	7444	695	266	232	0	0	0	0
f	12353	0	9322	8907	12353	12353	141	0	0	0	0	0	0

1sec

pp	14353	0	7923	4927	0	0	14353	7923	4927	14353	7923	4927	4631
-5	8405	7822	455	223	330	150	583	455	223	0	0	0	0
-4	9157	8432	565	305	400	205	725	565	305	0	0	0	0
-3	9938	8951	777	478	549	318	987	777	478	0	0	0	0
-2	10743	8900	1537	1100	1243	883	1843	1537	1100	0	0	0	0
-1	11548	8422	2541	1919	2243	1709	3126	2541	1919	0	0	0	0
0	12353	0	7936	7367	9715	9357	3434	2767	2558	0	0	0	0
1	11548	0	7119	6635	9437	9213	1798	907	802	0	0	0	0
2	10743	0	6750	6340	9051	8886	1393	556	482	0	0	0	0
3	9938	0	6368	5996	8562	8435	1138	393	330	0	0	0	0
4	9157	0	5967	5633	8005	7914	953	307	261	0	0	0	0
5	8405	0	5542	5233	7438	7357	820	246	207	0	0	0	0
f	12353	0	8342	7872	12353	12353	281	0	0	0	0	0	0

jwcdg-2

1sec

pp	15480	0	8576	5915	0	0	15480	8576	5915	15480	8576	5915	4862
-5	8405	7816	457	261	305	168	589	457	261	0	0	0	0
-4	9157	8387	595	359	418	245	770	595	359	0	0	0	0
-3	9938	8862	864	588	620	406	1076	864	588	0	0	0	0
-2	10743	8675	1711	1308	1411	1077	2068	1711	1308	0	0	0	0
-1	11548	7806	3037	2376	2714	2168	3742	3037	2376	0	0	0	0
0	12353	0	8564	8005	9235	8783	4168	3165	2866	0	0	0	0
1	11548	0	8081	7647	9253	8945	1969	1048	938	0	0	0	0
2	10743	0	7705	7337	8939	8693	1492	687	595	0	0	0	0
3	9938	0	7259	6913	8456	8240	1170	496	426	0	0	0	0
4	9157	0	6747	6454	7930	7753	933	378	325	0	0	0	0
5	8405	0	6275	6017	7394	7245	784	300	262	0	0	0	0
f	12353	105	9351	8931	12353	12353	105	0	0	0	0	0	0

2sec

pp	14415	0	8698	6360	0	0	14415	8698	6360	14415	8698	6360	3880
-5	8405	7832	447	285	330	199	573	447	285	0	0	0	0
-4	9157	8426	574	384	446	287	731	574	384	0	0	0	0
-3	9938	8861	857	620	680	481	1077	857	620	0	0	0	0
-2	10743	8625	1795	1400	1566	1224	2118	1795	1400	0	0	0	0
-1	11548	7765	3149	2535	2887	2342	3783	3149	2535	0	0	0	0
0	12353	0	8926	8365	9356	8899	4146	3236	2925	0	0	0	0
1	11548	0	8631	8228	9449	9140	1785	1053	963	0	0	0	0
2	10743	0	8251	7920	9121	8877	1269	691	625	0	0	0	0
3	9938	0	7744	7457	8632	8418	974	501	450	0	0	0	0
4	9157	0	7210	6951	8062	7877	781	388	344	0	0	0	0
5	8405	0	6671	6439	7481	7327	643	299	269	0	0	0	0
f	12353	0	9964	9589	12353	12353	24	0	0	0	0	0	0

4sec

pp	13515	0	8711	6603	0	0	13515	8711	6603	13515	8711	6603	3111
-5	8405	7841	443	298	365	239	564	443	298	0	0	0	0
-4	9157	8417	599	402	513	337	740	599	402	0	0	0	0
-3	9938	8879	861	644	731	547	1059	861	644	0	0	0	0
-2	10743	8633	1808	1467	1661	1356	2110	1808	1467	0	0	0	0
-1	11548	7787	3182	2613	3025	2514	3761	3182	2613	0	0	0	0
0	12353	0	9152	8644	9518	9089	4080	3259	2973	0	0	0	0
1	11548	0	8981	8614	9687	9393	1665	1067	981	0	0	0	0
2	10743	0	8605	8295	9333	9103	1153	696	631	0	0	0	0
3	9938	0	8014	7746	8770	8564	893	498	454	0	0	0	0
4	9157	0	7430	7186	8183	8008	712	390	350	0	0	0	0
5	8405	0	6888	6663	7609	7462	571	304	279	0	0	0	0
f	12353	0	10421	10049	12353	12353	23	0	0	0	0	0	0

8sec

pp	13003	0	8839	6780	0	0	13003	8839	6780	13003	8839	6780	2569
-5	8405	7843	453	307	386	255	562	453	307	0	0	0	0
-4	9157	8422	601	417	520	360	735	601	417	0	0	0	0
-3	9938	8866	886	662	767	578	1072	886	662	0	0	0	0
-2	10743	8619	1839	1512	1687	1396	2124	1839	1512	0	0	0	0
-1	11548	7776	3235	2664	3078	2572	3772	3235	2664	0	0	0	0
0	12353	0	9259	8749	9578	9153	4048	3264	2973	0	0	0	0
1	11548	0	9162	8797	9747	9476	1601	1077	995	0	0	0	0
2	10743	0	8790	8482	9404	9191	1112	716	655	0	0	0	0
3	9938	0	8225	7957	8861	8678	852	526	483	0	0	0	0
4	9157	0	7615	7366	8236	8080	687	405	368	0	0	0	0
5	8405	0	7037	6814	7631	7507	553	315	290	0	0	0	0
f	12353	0	10644	10317	12353	12353	1	0	0	0	0	0	0

16sec

pp	12789	0	8857	6825	0	0	12789	8857	6825	12789	8857	6825	2339
-5	8405	7838	459	315	417	285	567	459	315	0	0	0	0
-4	9157	8425	598	429	548	385	732	598	429	0	0	0	0
-3	9938	8875	873	657	807	604	1063	873	657	0	0	0	0
-2	10743	8628	1828	1513	1738	1436	2115	1828	1513	0	0	0	0
-1	11548	7775	3244	2692	3146	2622	3773	3244	2692	0	0	0	0
0	12353	0	9355	8853	9664	9260	4055	3284	3010	0	0	0	0
1	11548	0	9286	8923	9856	9608	1561	1074	994	0	0	0	0
2	10743	0	8887	8595	9485	9283	1106	722	657	0	0	0	0
3	9938	0	8303	8041	8930	8754	836	524	475	0	0	0	0
4	9157	0	7689	7456	8301	8162	651	400	362	0	0	0	0
5	8405	0	7110	6890	7696	7579	528	319	293	0	0	0	0
f	12353	0	10875	10549	12353	12353	0	0	0	0	0	0	0

jwcdg-3

1sec

pp	18876	0	9483	6637	0	0	18876	9483	6637	18876	9483	6637	6557
-5	8405	7717	485	297	355	204	688	485	297	0	0	0	0
-4	9157	8257	657	401	495	291	900	657	401	0	0	0	0
-3	9938	8672	933	631	733	479	1266	933	631	0	0	0	0
-2	10743	8369	1863	1421	1601	1225	2374	1863	1421	0	0	0	0
-1	11548	7197	3372	2653	3084	2451	4351	3372	2653	0	0	0	0
0	12353	0	9599	9019	9525	9097	5307	3923	3539	0	0	0	0
1	11548	0	8695	8273	9095	8871	2989	1487	1335	0	0	0	0
2	10743	0	8259	7918	8733	8552	2211	979	865	0	0	0	0
3	9938	0	7710	7411	8247	8096	1753	708	621	0	0	0	0
4	9157	0	7163	6904	7740	7617	1403	528	465	0	0	0	0
5	8405	0	6626	6400	7221	7113	1161	416	370	0	0	0	0
f	12353	0	9955	9538	12353	12353	57	0	0	0	0	0	0

2sec

pp	17368	0	9640	7110	0	0	17368	9640	7110	17368	9640	7110	5275
-5	8405	7745	481	312	399	247	660	481	312	0	0	0	0
-4	9157	8288	655	435	557	363	869	655	435	0	0	0	0
-3	9938	8715	936	685	801	572	1223	936	685	0	0	0	0
-2	10743	8390	1925	1513	1724	1352	2353	1925	1513	0	0	0	0
-1	11548	7175	3533	2843	3309	2681	4373	3533	2843	0	0	0	0
0	12353	0	9765	9212	9821	9373	5313	3968	3590	0	0	0	0
1	11548	0	9254	8885	9635	9384	2627	1504	1388	0	0	0	0
2	10743	0	8706	8423	9178	8993	1883	974	896	0	0	0	0
3	9938	0	8143	7884	8626	8468	1437	725	660	0	0	0	0
4	9157	0	7548	7314	8027	7899	1158	550	502	0	0	0	0
5	8405	0	7003	6782	7477	7361	955	443	409	0	0	0	0
f	12353	0	10564	10196	12353	12353	4	0	0	0	0	0	0

4sec

pp	16368	0	9813	7398	0	0	16368	9813	7398	16368	9813	7398	4284
-5	8405	7739	498	337	433	285	666	498	337	0	0	0	0
-4	9157	8278	674	470	594	411	879	674	470	0	0	0	0
-3	9938	8694	971	723	851	636	1244	971	723	0	0	0	0
-2	10743	8388	1958	1558	1810	1451	2355	1958	1558	0	0	0	0
-1	11548	7226	3571	2917	3419	2817	4322	3571	2917	0	0	0	0
0	12353	0	9883	9356	9985	9539	5224	3936	3584	0	0	0	0
1	11548	0	9567	9207	9946	9687	2393	1501	1384	0	0	0	0
2	10743	0	9029	8751	9441	9237	1686	979	900	0	0	0	0
3	9938	0	8403	8151	8860	8680	1283	715	651	0	0	0	0
4	9157	0	7768	7545	8222	8087	1039	550	497	0	0	0	0
5	8405	0	7184	6972	7629	7510	836	440	401	0	0	0	0
f	12353	0	10923	10597	12353	12353	17	0	0	0	0	0	0

8sec

pp	15689	0	9856	7468	0	0	15689	9856	7468	15689	9856	7468	3632
-5	8405	7738	508	338	457	307	667	508	338	0	0	0	0
-4	9157	8293	671	468	605	420	864	671	468	0	0	0	0
-3	9938	8716	960	725	872	661	1222	960	725	0	0	0	0
-2	10743	8384	1973	1590	1859	1499	2359	1973	1590	0	0	0	0
-1	11548	7206	3611	2957	3492	2881	4342	3611	2957	0	0	0	0
0	12353	0	10033	9515	10162	9741	5200	4009	3655	0	0	0	0
1	11548	0	9716	9371	10085	9868	2345	1535	1423	0	0	0	0
2	10743	0	9140	8882	9561	9399	1631	990	912	0	0	0	0
3	9938	0	8491	8259	8931	8784	1236	726	667	0	0	0	0
4	9157	0	7865	7654	8292	8171	995	559	511	0	0	0	0
5	8405	0	7257	7065	7675	7565	794	438	404	0	0	0	0
f	12353	0	11070	10755	12353	12353	0	0	0	0	0	0	0

16sec

pp	15125	0	9888	7665	0	0	15125	9888	7665	15125	9888	7665	3211
-5	8405	7754	509	362	461	328	651	509	362	0	0	0	0
-4	9157	8311	673	495	620	453	846	673	495	0	0	0	0
-3	9938	8726	971	746	888	682	1212	971	746	0	0	0	0
-2	10743	8397	1976	1620	1866	1533	2346	1976	1620	0	0	0	0
-1	11548	7243	3643	3007	3529	2935	4305	3643	3007	0	0	0	0
0	12353	0	10071	9564	10225	9806	5185	4008	3667	0	0	0	0
1	11548	0	9786	9457	10183	9977	2287	1541	1437	0	0	0	0
2	10743	0	9204	8956	9631	9477	1578	993	924	0	0	0	0
3	9938	0	8535	8318	8986	8861	1183	720	678	0	0	0	0
4	9157	0	7892	7695	8336	8219	932	549	515	0	0	0	0
5	8405	0	7302	7123	7720	7619	750	440	414	0	0	0	0
f	12353	0	11122	10828	12353	12353	6	0	0	0	0	0	0

Minimal Evaluation on NEGRA

jwcdg-2

1sec

0	16642	0	11489	10499	13523	12633	5380	4509	3912	0	0	0	0
1	15644	0	10526	9836	13111	12549	2760	1618	1382	0	0	0	0
2	14691	0	10021	9446	12670	12239	2146	1062	882	0	0	0	0
3	13778	0	9524	8994	12074	11709	1749	790	653	0	0	0	0
4	12889	0	8963	8506	11463	11166	1450	622	511	0	0	0	0
5	12029	0	8442	8031	10837	10592	1250	506	418	0	0	0	0
f	16642	0	11789	11216	16642	16642	267	0	0	0	0	0	0

2sec

0	16642	0	11964	11012	13291	12413	5442	4635	4032	0	0	0	0
1	15644	0	11306	10669	13044	12509	2534	1638	1433	0	0	0	0
2	14691	0	10799	10272	12599	12193	1863	1069	920	0	0	0	0
3	13778	0	10258	9796	12054	11705	1491	806	685	0	0	0	0
4	12889	0	9681	9268	11439	11136	1232	643	543	0	0	0	0
5	12029	0	9103	8729	10812	10561	1030	525	446	0	0	0	0
f	16642	0	12696	12185	16642	16642	87	0	0	0	0	0	0

4sec

0	16642	0	12311	11410	13299	12440	5435	4717	4119	0	0	0	0
1	15644	0	11869	11268	13255	12722	2372	1689	1475	0	0	0	0
2	14691	0	11386	10889	12806	12385	1702	1113	960	0	0	0	0
3	13778	0	10736	10310	12185	11821	1343	815	709	0	0	0	0
4	12889	0	10068	9691	11523	11215	1110	632	549	0	0	0	0
5	12029	0	9452	9106	10862	10602	918	516	453	0	0	0	0
f	16642	0	13383	12886	16642	16642	61	0	0	0	0	0	0

8sec

0	16642	0	12524	11647	13313	12465	5423	4747	4143	0	0	0	0
1	15644	0	12188	11640	13308	12797	2254	1686	1492	0	0	0	0
2	14691	0	11731	11272	12864	12482	1576	1126	991	0	0	0	0
3	13778	0	11101	10715	12248	11933	1237	846	747	0	0	0	0
4	12889	0	10425	10066	11536	11273	1024	665	583	0	0	0	0
5	12029	0	9763	9447	10871	10658	849	546	484	0	0	0	0
f	16642	0	13790	13373	16642	16642	12	0	0	0	0	0	0

16sec

0	16642	0	12683	11831	13362	12576	5404	4756	4190	0	0	0	0
1	15644	0	12465	11904	13442	12981	2180	1685	1489	0	0	0	0
2	14691	0	12000	11553	12999	12626	1569	1151	1012	0	0	0	0
3	13778	0	11329	10948	12362	12051	1212	855	756	0	0	0	0
4	12889	0	10634	10293	11661	11402	969	669	596	0	0	0	0
5	12029	0	9959	9638	10972	10754	807	556	497	0	0	0	0
f	16642	0	14194	13763	16642	16642	0	0	0	0	0	0	0

MaltParser without Reanalysis

0	16642	0	14085	8574	15579	10430	6957	6137	903	0	0	0	0
1	15644	0	13048	10563	14844	12829	3094	2427	343	0	0	0	0
2	14691	0	12297	10561	13994	12713	2141	1590	204	0	0	0	0
3	13778	0	11501	10099	13138	12201	1673	1185	135	0	0	0	0
4	12889	0	10741	9559	12308	11568	1380	942	95	0	0	0	0
5	12029	0	9996	8984	11495	10906	1175	773	77	0	0	0	0
f	16642	0	13475	12862	16642	16642	0	0	0	0	0	0	0

MaltParser with Reanalysis

0	16642	0	14085	8573	15579	10429	6957	6137	902	0	0	0	0
1	15644	0	13048	10596	14844	12826	3094	2427	343	0	0	0	0
2	14691	0	12297	10573	13994	12711	2141	1590	204	0	0	0	0
3	13778	0	11501	10112	13138	12200	1673	1185	135	0	0	0	0
4	12889	0	10741	9573	12308	11567	1380	942	95	0	0	0	0
5	12029	0	9996	8991	11495	10905	1175	773	77	0	0	0	0
f	16642	0	13475	12969	16642	16642	0	0	0	0	0	0	0

jwcdg-4

1sec

0	16642	0	13511	12371	14092	13189	7179	6023	5162	0	0	0	0
1	15644	0	11693	10971	13039	12557	4443	2435	2077	0	0	0	0
2	14691	0	11028	10439	12421	12061	3485	1629	1352	0	0	0	0
3	13778	0	10367	9856	11817	11529	2893	1223	1005	0	0	0	0
4	12889	0	9732	9284	11212	10977	2421	956	778	0	0	0	0
5	12029	0	9095	8694	10579	10375	2103	766	622	0	0	0	0
f	16642	0	12644	12109	16642	16642	639	0	0	0	0	0	0

2sec

0	16642	0	13677	12590	14127	13169	7225	6054	5204	0	0	0	0
1	15644	0	12424	11795	13353	12833	3913	2434	2140	0	0	0	0
2	14691	0	11681	11191	12705	12327	2917	1616	1407	0	0	0	0
3	13778	0	11001	10571	12058	11748	2337	1218	1056	0	0	0	0
4	12889	0	10308	9927	11383	11129	1951	951	828	0	0	0	0
5	12029	0	9679	9325	10759	10536	1657	781	685	0	0	0	0
f	16642	0	13541	13049	16642	16642	239	0	0	0	0	0	0

4sec

0	16642	0	13843	12825	14246	13312	7123	6033	5240	0	0	0	0
1	15644	0	12954	12364	13676	13190	3503	2430	2148	0	0	0	0
2	14691	0	12254	11788	12996	12612	2529	1613	1413	0	0	0	0
3	13778	0	11533	11128	12340	12015	1982	1208	1054	0	0	0	0
4	12889	0	10788	10431	11617	11359	1651	945	820	0	0	0	0
5	12029	0	10096	9764	10935	10714	1372	766	667	0	0	0	0
f	16642	0	14240	13795	16642	16642	64	0	0	0	0	0	0

8sec

0	16642	0	14021	13046	14402	13522	7104	6079	5306	0	0	0	0
1	15644	0	13228	12671	13852	13421	3388	2453	2195	0	0	0	0
2	14691	0	12506	12089	13151	12837	2370	1601	1424	0	0	0	0
3	13778	0	11748	11377	12407	12134	1842	1202	1066	0	0	0	0
4	12889	0	11021	10684	11682	11446	1512	951	841	0	0	0	0
5	12029	0	10300	9994	10986	10777	1250	769	685	0	0	0	0
f	16642	0	14584	14160	16642	16642	7	0	0	0	0	0	0

16sec

0	16642	0	14103	13130	14494	13611	7075	6089	5322	0	0	0	0
1	15644	0	13349	12816	13996	13594	3278	2455	2210	0	0	0	0
2	14691	0	12617	12218	13264	12964	2282	1602	1437	0	0	0	0
3	13778	0	11839	11496	12509	12261	1754	1198	1082	0	0	0	0
4	12889	0	11088	10786	11763	11541	1420	938	850	0	0	0	0
5	12029	0	10380	10100	11037	10843	1183	763	694	0	0	0	0
f	16642	0	14713	14316	16642	16642	9	0	0	0	0	0	0

jwcdg-3

1sec

0	16642	0	13488	12269	15833	15405	6776	5879	5068	0	0	0	0
1	15644	0	10804	9988	14931	14724	4972	2409	2029	0	0	0	0
2	14691	0	9818	9151	14183	14061	4467	1632	1323	0	0	0	0
3	13778	0	9162	8563	13460	13380	4022	1238	987	0	0	0	0
4	12889	0	8513	7973	12665	12608	3664	982	765	0	0	0	0
5	12029	0	7915	7428	11884	11837	3324	799	618	0	0	0	0
f	16642	0	10254	9789	16642	16642	3881	0	0	0	0	0	0

2sec

0	16642	0	13657	12540	14992	14281	6988	5987	5164	0	0	0	0
1	15644	0	11931	11226	14237	13882	4163	2412	2077	0	0	0	0
2	14691	0	11121	10565	13590	13347	3401	1622	1368	0	0	0	0
3	13778	0	10451	9970	12885	12686	2908	1230	1033	0	0	0	0
4	12889	0	9770	9335	12155	11988	2513	966	800	0	0	0	0
5	12029	0	9140	8759	11445	11325	2206	789	656	0	0	0	0
f	16642	0	12200	11763	16642	16642	1952	0	0	0	0	0	0

4sec

0	16642	0	13884	12843	14470	13589	7043	6034	5234	0	0	0	0
1	15644	0	12742	12146	13845	13388	3629	2422	2146	0	0	0	0
2	14691	0	12014	11541	13172	12835	2695	1607	1413	0	0	0	0
3	13778	0	11277	10876	12467	12211	2176	1208	1055	0	0	0	0
4	12889	0	10563	10213	11768	11553	1828	951	827	0	0	0	0
5	12029	0	9881	9561	11073	10893	1556	771	669	0	0	0	0
f	16642	0	13663	13227	16642	16642	496	0	0	0	0	0	0

Minimal Evaluation NEGRA-3-10

jwcdg-2

1sec

0	1704	0	1366	1284	1428	1353	570	429	382	0	0	0	0
1	1449	0	1177	1118	1262	1216	207	87	79	0	0	0	0
2	1194	0	999	952	1073	1039	126	44	38	0	0	0	0
3	939	0	810	774	853	833	76	17	16	0	0	0	0
4	708	0	626	605	660	651	35	5	4	0	0	0	0
5	506	0	463	444	479	475	17	0	0	0	0	0	0
f	1704	0	1568	1504	1704	1704	0	0	0	0	0	0	0

2sec

0	1704	0	1381	1293	1449	1370	562	427	379	0	0	0	0
1	1449	0	1192	1133	1272	1231	203	85	79	0	0	0	0
2	1194	0	1001	958	1074	1043	126	43	40	0	0	0	0
3	939	0	819	787	863	842	70	17	17	0	0	0	0
4	708	0	635	610	663	651	34	5	5	0	0	0	0
5	506	0	469	444	481	474	17	0	0	0	0	0	0
f	1704	0	1579	1515	1704	1704	0	0	0	0	0	0	0

4sec

0	1704	0	1387	1305	1449	1376	560	426	379	0	0	0	0
1	1449	0	1204	1148	1285	1248	194	87	82	0	0	0	0
2	1194	0	1012	970	1082	1054	123	43	40	0	0	0	0
3	939	0	827	793	868	848	68	17	17	0	0	0	0
4	708	0	634	611	665	656	34	5	5	0	0	0	0
5	506	0	468	445	480	476	19	0	0	0	0	0	0
f	1704	0	1583	1522	1704	1704	0	0	0	0	0	0	0

8sec

0	1704	0	1384	1301	1443	1370	564	427	379	0	0	0	0
1	1449	0	1202	1143	1281	1241	199	88	83	0	0	0	0
2	1194	0	1013	965	1085	1054	120	43	40	0	0	0	0
3	939	0	825	790	867	845	66	17	17	0	0	0	0
4	708	0	636	612	665	655	33	5	5	0	0	0	0
5	506	0	466	446	478	473	20	0	0	0	0	0	0
f	1704	0	1575	1509	1704	1704	0	0	0	0	0	0	0

16sec

0	1704	0	1387	1307	1453	1378	566	429	387	0	0	0	0
1	1449	0	1205	1152	1285	1249	202	90	84	0	0	0	0
2	1194	0	1013	971	1084	1058	125	44	41	0	0	0	0
3	939	0	825	792	866	848	69	17	17	0	0	0	0
4	708	0	638	618	668	659	31	5	5	0	0	0	0
5	506	0	468	447	481	474	17	0	0	0	0	0	0
f	1704	0	1580	1516	1704	1704	0	0	0	0	0	0	0

MaltParser without Reanalysis

0	1704	0	1450	1000	1530	1110	671	508	94	0	0	0	0
1	1449	0	1202	1061	1321	1220	248	116	24	0	0	0	0
2	1194	0	993	896	1106	1055	145	58	11	0	0	0	0
3	939	0	791	728	876	852	90	27	7	0	0	0	0
4	708	0	606	564	671	666	44	9	4	0	0	0	0
5	506	0	441	418	485	484	21	1	0	0	0	0	0
f	1704	0	1498	1377	1704	1704	0	0	0	0	0	0	0

MaltParser with Reanalysis

0	1704	0	1450	1000	1530	1110	671	508	94	0	0	0	0
1	1449	0	1202	1067	1321	1219	248	116	24	0	0	0	0
2	1194	0	993	907	1106	1054	145	58	11	0	0	0	0
3	939	0	791	741	876	852	90	27	7	0	0	0	0
4	708	0	606	578	671	666	44	9	4	0	0	0	0
5	506	0	441	424	485	484	21	1	0	0	0	0	0
f	1704	0	1498	1438	1704	1704	0	0	0	0	0	0	0

jwcdg-4

1sec

0	1704	0	1447	1353	1489	1404	665	508	452	0	0	0	0
1	1449	0	1217	1163	1283	1247	267	122	113	0	0	0	0
2	1194	0	1018	981	1083	1063	151	58	54	0	0	0	0
3	939	0	819	793	864	853	90	26	26	0	0	0	0
4	708	0	629	610	664	659	41	8	8	0	0	0	0
5	506	0	465	448	483	478	19	0	0	0	0	0	0
f	1704	0	1584	1535	1704	1704	0	0	0	0	0	0	0

2sec

0	1704	0	1461	1364	1514	1431	654	506	446	0	0	0	0
1	1449	0	1242	1188	1312	1280	242	122	112	0	0	0	0
2	1194	0	1033	996	1104	1085	139	58	53	0	0	0	0
3	939	0	837	807	879	869	79	26	26	0	0	0	0
4	708	0	642	619	671	664	36	8	7	0	0	0	0
5	506	0	471	453	489	482	19	1	1	0	0	0	0
f	1704	0	1597	1546	1704	1704	0	0	0	0	0	0	0

4sec

0	1704	0	1459	1371	1511	1431	645	498	444	0	0	0	0
1	1449	0	1246	1194	1315	1283	240	121	111	0	0	0	0
2	1194	0	1034	1000	1106	1087	138	58	55	0	0	0	0
3	939	0	839	811	883	872	77	27	27	0	0	0	0
4	708	0	644	626	675	668	32	8	8	0	0	0	0
5	506	0	471	454	488	481	18	1	1	0	0	0	0
f	1704	0	1596	1546	1704	1704	0	0	0	0	0	0	0

8sec

0	1704	0	1384	1301	1443	1370	564	427	379	0	0	0	0
1	1449	0	1202	1143	1281	1241	199	88	83	0	0	0	0
2	1194	0	1013	965	1085	1054	120	43	40	0	0	0	0
3	939	0	825	790	867	845	66	17	17	0	0	0	0
4	708	0	636	612	665	655	33	5	5	0	0	0	0
5	506	0	466	446	478	473	20	0	0	0	0	0	0
f	1704	0	1575	1509	1704	1704	0	0	0	0	0	0	0

16sec

0	1704	0	1387	1307	1453	1378	566	429	387	0	0	0	0
1	1449	0	1205	1152	1285	1249	202	90	84	0	0	0	0
2	1194	0	1013	971	1084	1058	125	44	41	0	0	0	0
3	939	0	825	792	866	848	69	17	17	0	0	0	0
4	708	0	638	618	668	659	31	5	5	0	0	0	0
5	506	0	468	447	481	474	17	0	0	0	0	0	0
f	1704	0	1580	1516	1704	1704	0	0	0	0	0	0	0

Minimal Evaluation NEGRA-3-20

1sec

0	6606	0	4972	4613	5373	5040	2220	1836	1625	0	0	0	0
1	6053	0	4580	4328	5137	4929	910	549	487	0	0	0	0
2	5500	0	4227	4028	4791	4630	654	331	284	0	0	0	0
3	4947	0	3881	3701	4398	4263	477	219	183	0	0	0	0
4	4394	0	3484	3337	3979	3873	339	153	127	0	0	0	0
5	3870	0	3098	2972	3537	3456	271	110	94	0	0	0	0
f	6606	7	5439	5216	6606	6606	7	0	0	0	0	0	0

Maltparser without Reanalysis

0	6606	0	5649	3601	6166	4228	2691	2297	374	0	0	0	0
1	6053	0	5059	4248	5714	5060	1098	770	134	0	0	0	0
2	5500	0	4600	4054	5225	4844	718	471	80	0	0	0	0
3	4947	0	4146	3728	4712	4463	514	313	49	0	0	0	0
4	4394	0	3701	3365	4201	4024	386	222	33	0	0	0	0
5	3870	0	3261	2994	3710	3582	300	162	23	0	0	0	0
f	6606	0	5521	5240	6606	6606	0	0	0	0	0	0	0

Maltparser with Reanalysis

0	6678	0	5701	3641	6221	4270	2720	2308	373	0	0	0	0
1	6101	0	5095	4284	5752	5094	1109	770	134	0	0	0	0
2	5524	0	4623	4079	5249	4866	718	471	80	0	0	0	0
3	4947	0	4146	3741	4712	4462	514	313	49	0	0	0	0
4	4394	0	3701	3379	4201	4023	386	222	33	0	0	0	0
5	3870	0	3261	3001	3710	3581	300	162	23	0	0	0	0
f	6678	0	5589	5367	6678	6678	0	0	0	0	0	0	0

jwcdg-4

1sec

0	6606	0	5478	5104	5633	5290	2775	2274	2013	0	0	0	0
1	6053	0	4879	4631	5162	4983	1359	779	698	0	0	0	0
2	5500	0	4473	4289	4769	4640	932	470	418	0	0	0	0
3	4947	0	4054	3900	4358	4259	690	314	277	0	0	0	0
4	4394	0	3646	3511	3922	3846	508	218	191	0	0	0	0
5	3870	0	3233	3114	3489	3424	397	155	135	0	0	0	0
f	6606	0	5727	5508	6606	6606	0	0	0	0	0	0	0

2sec

0	6678	0	5592	5215	5766	5438	2767	2275	2008	0	0	0	0
1	6101	0	5145	4927	5414	5254	1201	778	709	0	0	0	0
2	5524	0	4676	4519	4958	4840	812	475	432	0	0	0	0
3	4947	0	4216	4068	4485	4386	580	319	281	0	0	0	0
4	4394	0	3770	3638	4004	3922	436	223	198	0	0	0	0
5	3870	0	3346	3233	3562	3495	339	165	151	0	0	0	0
f	6678	0	6037	5851	6678	6678	0	0	0	0	0	0	0

4sec

0	6678	0	5639	5287	5830	5508	2721	2254	2011	0	0	0	0
1	6101	0	5226	5011	5490	5338	1137	773	698	0	0	0	0
2	5524	0	4764	4612	5027	4916	748	468	427	0	0	0	0
3	4947	0	4297	4157	4555	4464	530	315	281	0	0	0	0
4	4394	0	3840	3722	4061	3991	392	221	196	0	0	0	0
5	3870	0	3402	3294	3608	3547	295	160	144	0	0	0	0
f	6678	0	6125	5956	6678	6678	0	0	0	0	0	0	0

8sec

0	6678	0	5685	5341	5870	5562	2702	2268	2030	0	0	0	0
1	6101	0	5278	5073	5531	5395	1108	780	715	0	0	0	0
2	5524	0	4799	4658	5063	4974	725	472	434	0	0	0	0
3	4947	0	4330	4196	4568	4489	508	318	285	0	0	0	0
4	4394	0	3880	3766	4088	4026	376	226	202	0	0	0	0
5	3870	0	3430	3333	3627	3573	272	158	144	0	0	0	0
f	6678	0	6143	5974	6678	6678	0	0	0	0	0	0	0

16sec

0	6678	0	5701	5357	5882	5578	2706	2268	2033	0	0	0	0
1	6101	0	5300	5102	5554	5426	1102	779	715	0	0	0	0
2	5524	0	4818	4682	5086	5003	713	470	434	0	0	0	0
3	4947	0	4349	4221	4594	4527	497	316	288	0	0	0	0
4	4394	0	3880	3773	4101	4047	351	218	200	0	0	0	0
5	3870	0	3438	3344	3642	3601	257	155	144	0	0	0	0
f	6678	0	6154	5990	6678	6678	0	0	0	0	0	0	0

jwcdg-3

1sec

0	16642	0	13488	12269	15833	15405	6776	5879	5068	0	0	0	0
1	15644	0	10804	9988	14931	14724	4972	2409	2029	0	0	0	0
2	14691	0	9818	9151	14183	14061	4467	1632	1323	0	0	0	0
3	13778	0	9162	8563	13460	13380	4022	1238	987	0	0	0	0
4	12889	0	8513	7973	12665	12608	3664	982	765	0	0	0	0
5	12029	0	7915	7428	11884	11837	3324	799	618	0	0	0	0
f	16642	3881	10254	9789	16642	16642	3881	0	0	0	0	0	0

0	16642	0	13488	12269	15833	15405	6776	5879	5068	0	0	0	0
1	15644	0	10804	9988	14931	14724	4972	2409	2029	0	0	0	0
2	14691	0	9818	9151	14183	14061	4467	1632	1323	0	0	0	0
3	13778	0	9162	8563	13460	13380	4022	1238	987	0	0	0	0
4	12889	0	8513	7973	12665	12608	3664	982	765	0	0	0	0
5	12029	0	7915	7428	11884	11837	3324	799	618	0	0	0	0
f	16642	3881	10254	9789	16642	16642	3881	0	0	0	0	0	0

2sec

0	16642	0	13657	12540	14992	14281	6988	5987	5164	0	0	0	0
1	15644	0	11931	11226	14237	13882	4163	2412	2077	0	0	0	0
2	14691	0	11121	10565	13590	13347	3401	1622	1368	0	0	0	0
3	13778	0	10451	9970	12885	12686	2908	1230	1033	0	0	0	0
4	12889	0	9770	9335	12155	11988	2513	966	800	0	0	0	0
5	12029	0	9140	8759	11445	11325	2206	789	656	0	0	0	0
f	16642	1952	12200	11763	16642	16642	1952	0	0	0	0	0	0

0	16642	0	13657	12540	14992	14281	6988	5987	5164	0	0	0	0
1	15644	0	11931	11226	14237	13882	4163	2412	2077	0	0	0	0
2	14691	0	11121	10565	13590	13347	3401	1622	1368	0	0	0	0
3	13778	0	10451	9970	12885	12686	2908	1230	1033	0	0	0	0
4	12889	0	9770	9335	12155	11988	2513	966	800	0	0	0	0
5	12029	0	9140	8759	11445	11325	2206	789	656	0	0	0	0
f	16642	1952	12200	11763	16642	16642	1952	0	0	0	0	0	0

4sec

0	16642	0	13884	12843	14470	13589	7043	6034	5234	0	0	0	0
1	15644	0	12742	12146	13845	13388	3629	2422	2146	0	0	0	0
2	14691	0	12014	11541	13172	12835	2695	1607	1413	0	0	0	0
3	13778	0	11277	10876	12467	12211	2176	1208	1055	0	0	0	0
4	12889	0	10563	10213	11768	11553	1828	951	827	0	0	0	0
5	12029	0	9881	9561	11073	10893	1556	771	669	0	0	0	0
f	16642	496	13663	13227	16642	16642	496	0	0	0	0	0	0

0	16642	0	13884	12843	14470	13589	7043	6034	5234	0	0	0	0
1	15644	0	12742	12146	13845	13388	3629	2422	2146	0	0	0	0
2	14691	0	12014	11541	13172	12835	2695	1607	1413	0	0	0	0
3	13778	0	11277	10876	12467	12211	2176	1208	1055	0	0	0	0
4	12889	0	10563	10213	11768	11553	1828	951	827	0	0	0	0
5	12029	0	9881	9561	11073	10893	1556	771	669	0	0	0	0
f	16642	496	13663	13227	16642	16642	496	0	0	0	0	0	0

Minimal Evaluation NEGRA-3-30

jwcdg-2

1sec

0	12353	0	8901	8196	9965	9278	4168	3502	3057	0	0	0	0
1	11548	0	8227	7743	9644	9216	1969	1194	1034	0	0	0	0
2	10743	0	7789	7394	9237	8904	1492	771	652	0	0	0	0
3	9938	0	7324	6957	8698	8417	1170	561	470	0	0	0	0
4	9157	0	6800	6490	8139	7907	933	431	361	0	0	0	0
5	8405	0	6319	6047	7572	7383	784	344	292	0	0	0	0
f	12353	105	9351	8931	12353	12353	105	0	0	0	0	0	0

2sec

0	12353	0	9264	8564	9937	9276	4146	3574	3124	0	0	0	0
1	11548	0	8772	8323	9726	9322	1785	1194	1058	0	0	0	0
2	10743	0	8337	7978	9307	8998	1269	777	683	0	0	0	0
3	9938	0	7813	7501	8785	8514	974	570	494	0	0	0	0
4	9157	0	7266	6988	8202	7964	781	444	381	0	0	0	0
5	8405	0	6727	6476	7602	7405	643	355	306	0	0	0	0
f	12353	24	9964	9589	12353	12353	24	0	0	0	0	0	0

4sec

0	12353	0	9467	8813	10026	9424	4080	3574	3142	0	0	0	0
1	11548	0	9130	8713	9940	9582	1665	1216	1080	0	0	0	0
2	10743	0	8700	8358	9521	9237	1153	791	694	0	0	0	0
3	9938	0	8088	7794	8916	8668	893	572	502	0	0	0	0
4	9157	0	7481	7223	8308	8097	712	441	387	0	0	0	0
5	8405	0	6933	6696	7706	7532	571	349	312	0	0	0	0
f	12353	23	10421	10049	12353	12353	23	0	0	0	0	0	0

8sec

0	12353	0	9574	8924	10052	9457	4048	3579	3148	0	0	0	0
1	11548	0	9289	8880	9973	9638	1601	1204	1078	0	0	0	0
2	10743	0	8880	8544	9574	9320	1112	806	717	0	0	0	0
3	9938	0	8291	8005	8991	8779	852	592	531	0	0	0	0
4	9157	0	7666	7402	8342	8163	687	456	404	0	0	0	0
5	8405	0	7083	6850	7712	7571	553	361	326	0	0	0	0
f	12353	1	10644	10317	12353	12353	1	0	0	0	0	0	0

16sec

0	12353	0	9659	9022	10115	9543	4055	3588	3179	0	0	0	0
1	11548	0	9416	9006	10070	9759	1561	1204	1077	0	0	0	0
2	10743	0	8979	8660	9653	9404	1106	814	722	0	0	0	0
3	9938	0	8372	8093	9054	8843	836	593	527	0	0	0	0
4	9157	0	7741	7496	8404	8236	651	452	402	0	0	0	0
5	8405	0	7154	6925	7775	7636	528	363	328	0	0	0	0
f	12353	0	10875	10549	12353	12353	0	0	0	0	0	0	0

MaltParser without Reanalysis

0	12353	0	10482	6434	11565	7744	5144	4505	659	0	0	0	0
1	11548	0	9636	7893	10950	9504	2237	1704	244	0	0	0	0
2	10743	0	9011	7784	10236	9324	1537	1110	147	0	0	0	0
3	9938	0	8319	7344	9482	8836	1173	806	94	0	0	0	0
4	9157	0	7672	6857	8757	8260	944	631	67	0	0	0	0
5	8405	0	7032	6343	8048	7657	787	509	52	0	0	0	0
f	12353	0	10106	9648	12353	12353	0	0	0	0	0	0	0

MaltParser with Reanalysis

0	12353	0	10482	6433	11565	7743	5144	4505	658	0	0	0	0
1	11548	0	9636	7899	10950	9501	2237	1704	244	0	0	0	0
2	10743	0	9011	7796	10236	9322	1537	1110	147	0	0	0	0
3	9938	0	8319	7357	9482	8835	1173	806	94	0	0	0	0
4	9157	0	7672	6871	8757	8259	944	631	67	0	0	0	0
5	8405	0	7032	6350	8048	7656	787	509	52	0	0	0	0
f	12353	0	10106	9724	12353	12353	0	0	0	0	0	0	0

jwcdg-4

1sec

0	12353	0	10122	9324	10384	9696	5307	4446	3844	0	0	0	0
1	11548	0	8919	8417	9550	9191	2989	1711	1479	0	0	0	0
2	10743	0	8409	8010	9040	8774	2211	1129	957	0	0	0	0
3	9938	0	7832	7485	8480	8271	1753	830	695	0	0	0	0
4	9157	0	7273	6973	7928	7759	1403	638	534	0	0	0	0
5	8405	0	6713	6456	7369	7223	1161	503	426	0	0	0	0
f	12353	57	9955	9538	12353	12353	57	0	0	0	0	0	0

2sec

0	12353	0	10259	9489	10532	9844	5313	4462	3867	0	0	0	0
1	11548	0	9463	9027	9981	9628	2627	1713	1530	0	0	0	0
2	10743	0	8852	8525	9414	9158	1883	1120	998	0	0	0	0
3	9938	0	8239	7948	8803	8596	1437	821	724	0	0	0	0
4	9157	0	7633	7376	8184	8011	1158	635	564	0	0	0	0
5	8405	0	7073	6838	7606	7455	955	513	465	0	0	0	0
f	12353	4	10564	10196	12353	12353	4	0	0	0	0	0	0

4sec

0	12353	0	10383	9658	10683	10009	5224	4436	3886	0	0	0	0
1	11548	0	9771	9345	10276	9928	2393	1705	1522	0	0	0	0
2	10743	0	9167	8843	9663	9396	1686	1117	992	0	0	0	0
3	9938	0	8507	8219	9028	8809	1283	819	719	0	0	0	0
4	9157	0	7854	7606	8366	8196	1039	636	558	0	0	0	0
5	8405	0	7250	7020	7751	7607	836	506	449	0	0	0	0
f	12353	17	10923	10597	12353	12353	17	0	0	0	0	0	0

0	12353	0	10383	9658	10683	10009	5224	4436	3886	0	0	0	0
1	11548	0	9771	9345	10276	9928	2393	1705	1522	0	0	0	0
2	10743	0	9167	8843	9663	9396	1686	1117	992	0	0	0	0
3	9938	0	8507	8219	9028	8809	1283	819	719	0	0	0	0
4	9157	0	7854	7606	8366	8196	1039	636	558	0	0	0	0
5	8405	0	7250	7020	7751	7607	836	506	449	0	0	0	0
f	12353	17	10923	10597	12353	12353	17	0	0	0	0	0	0

8sec

0	12353	0	9574	8924	10052	9457	4048	3579	3148	0	0	0	0
1	11548	0	9289	8880	9973	9638	1601	1204	1078	0	0	0	0
2	10743	0	8880	8544	9574	9320	1112	806	717	0	0	0	0
3	9938	0	8291	8005	8991	8779	852	592	531	0	0	0	0
4	9157	0	7666	7402	8342	8163	687	456	404	0	0	0	0
5	8405	0	7083	6850	7712	7571	553	361	326	0	0	0	0
f	12353	1	10644	10317	12353	12353	1	0	0	0	0	0	0

16sec

0	12353	0	9659	9022	10115	9543	4055	3588	3179	0	0	0	0
1	11548	0	9416	9006	10070	9759	1561	1204	1077	0	0	0	0
2	10743	0	8979	8660	9653	9404	1106	814	722	0	0	0	0
3	9938	0	8372	8093	9054	8843	836	593	527	0	0	0	0
4	9157	0	7741	7496	8404	8236	651	452	402	0	0	0	0
5	8405	0	7154	6925	7775	7636	528	363	328	0	0	0	0
f	12353	0	10875	10549	12353	12353	0	0	0	0	0	0	0

Evaluation on CREG-109

jwcdg-0

16sec

pp	923	0	583	407	0	0	923	583	407	923	583	407	269
-5	360	339	18	9	13	4	21	18	9	0	0	0	0
-4	467	427	36	22	27	10	40	36	22	0	0	0	0
-3	574	499	67	42	56	32	75	67	42	0	0	0	0
-2	682	526	143	103	122	86	156	143	103	0	0	0	0
-1	791	469	286	217	264	209	322	286	217	0	0	0	0
0	900	0	686	619	699	670	323	226	203	0	0	0	0
1	791	0	628	569	672	656	115	51	44	0	0	0	0
2	682	0	560	513	596	583	75	26	21	0	0	0	0
3	574	0	474	435	514	504	51	13	10	0	0	0	0
4	467	0	387	352	423	418	32	4	2	0	0	0	0
5	360	0	301	272	330	327	26	4	2	0	0	0	0
f	900	0	783	722	900	900	0	0	0	0	0	0	0

8sec

pp	925	0	575	401	0	0	925	575	401	925	575	401	272
-5	360	339	18	9	13	4	21	18	9	0	0	0	0
-4	467	426	37	22	25	9	41	37	22	0	0	0	0
-3	574	496	68	42	54	30	78	68	42	0	0	0	0
-2	682	524	144	104	124	86	158	144	104	0	0	0	0
-1	791	475	276	211	255	201	316	276	211	0	0	0	0
0	900	0	682	615	703	667	316	226	201	0	0	0	0
1	791	0	620	561	670	649	114	49	40	0	0	0	0
2	682	0	552	506	592	574	76	26	20	0	0	0	0
3	574	0	473	435	513	500	53	12	10	0	0	0	0
4	467	0	388	353	421	415	33	4	2	0	0	0	0
5	360	0	303	275	331	327	23	4	2	0	0	0	0
f	900	0	774	713	900	900	0	0	0	0	0	0	0

4sec

pp	913	0	557	393	0	0	913	557	393	913	557	393	281
-5	360	341	16	8	11	3	19	16	8	0	0	0	0
-4	467	428	33	19	24	8	39	33	19	0	0	0	0
-3	574	499	66	42	51	28	75	66	42	0	0	0	0
-2	682	528	143	105	121	84	154	143	105	0	0	0	0
-1	791	480	272	208	249	197	311	272	208	0	0	0	0
0	900	0	677	608	694	662	308	223	196	0	0	0	0
1	791	0	612	551	659	639	115	46	37	0	0	0	0
2	682	0	537	490	584	566	79	22	17	0	0	0	0
3	574	0	454	418	503	490	56	9	7	0	0	0	0
4	467	0	378	344	415	409	38	4	2	0	0	0	0
5	360	0	292	265	322	316	29	4	2	0	0	0	0
f	900	0	759	703	900	900	0	0	0	0	0	0	0

2sec

pp	921	0	546	375	0	0	921	546	375	921	546	375	300
-5	360	341	16	8	12	3	19	16	8	0	0	0	0
-4	467	429	32	19	23	7	38	32	19	0	0	0	0
-3	574	499	66	36	52	24	75	66	36	0	0	0	0
-2	682	530	140	98	116	77	152	140	98	0	0	0	0
-1	791	486	266	203	242	193	305	266	203	0	0	0	0
0	900	0	661	591	689	657	297	216	191	0	0	0	0
1	791	0	598	536	653	634	110	45	36	0	0	0	0
2	682	0	530	480	580	567	78	24	19	0	0	0	0
3	574	0	447	406	500	492	56	8	7	0	0	0	0
4	467	0	367	334	411	407	42	4	3	0	0	0	0
5	360	0	283	258	320	315	35	3	2	0	0	0	0
f	900	0	751	690	900	900	1	0	0	0	0	0	0

1sec

pp	919	0	530	363	0	0	919	530	363	919	530	363	327
-5	360	343	14	8	9	3	17	14	8	0	0	0	0
-4	467	429	33	19	25	8	38	33	19	0	0	0	0
-3	574	502	64	39	49	27	72	64	39	0	0	0	0
-2	682	533	138	96	113	72	149	138	96	0	0	0	0
-1	791	508	254	189	228	172	283	254	189	0	0	0	0
0	900	0	645	577	702	664	283	204	180	0	0	0	0
1	791	0	559	503	653	625	116	40	35	0	0	0	0
2	682	0	497	450	576	559	88	20	16	0	0	0	0
3	574	0	425	384	498	484	68	7	6	0	0	0	0
4	467	0	351	321	406	399	54	5	4	0	0	0	0
5	360	0	275	248	317	311	40	3	2	0	0	0	0
f	900	0	718	647	900	900	18	0	0	0	0	0	0

jwcdg-2

1sec

pp	959	0	562	386	0	0	959	562	386	959	562	386	336
-5	360	340	17	7	13	3	20	17	7	0	0	0	0
-4	467	428	34	20	27	8	39	34	20	0	0	0	0
-3	574	503	64	38	57	29	71	64	38	0	0	0	0
-2	682	527	147	103	132	86	155	147	103	0	0	0	0
-1	791	486	273	206	256	198	305	273	206	0	0	0	0
0	900	0	654	590	674	644	307	208	187	0	0	0	0
1	791	0	590	535	641	622	118	42	34	0	0	0	0
2	682	0	533	483	576	556	85	22	18	0	0	0	0
3	574	0	448	408	495	487	62	9	7	0	0	0	0
4	467	0	374	337	412	409	45	5	3	0	0	0	0
5	360	0	289	260	320	316	34	4	2	0	0	0	0
f	900	0	769	703	900	900	2	0	0	0	0	0	0

2sec

pp	927	0	569	396	0	0	927	569	396	927	569	396	283
-5	360	339	18	9	13	4	21	18	9	0	0	0	0
-4	467	429	35	21	28	11	38	35	21	0	0	0	0
-3	574	495	71	43	60	34	79	71	43	0	0	0	0
-2	682	524	145	103	129	86	158	145	103	0	0	0	0
-1	791	480	270	206	254	200	311	270	206	0	0	0	0
0	900	0	682	613	692	655	320	225	200	0	0	0	0
1	791	0	617	556	657	632	121	49	38	0	0	0	0
2	682	0	549	500	589	566	87	28	20	0	0	0	0
3	574	0	461	422	499	489	62	12	8	0	0	0	0
4	467	0	379	347	410	405	41	4	2	0	0	0	0
5	360	0	299	272	327	321	27	3	2	0	0	0	0
f	900	0	779	716	900	900	0	0	0	0	0	0	0

4sec

pp	913	0	571	407	0	0	913	571	407	913	571	407	269
-5	360	340	17	9	12	4	20	17	9	0	0	0	0
-4	467	426	36	23	28	12	41	36	23	0	0	0	0
-3	574	499	67	44	59	35	75	67	44	0	0	0	0
-2	682	525	144	103	129	90	157	144	103	0	0	0	0
-1	791	474	280	216	265	210	317	280	216	0	0	0	0
0	900	0	687	621	699	669	318	224	204	0	0	0	0
1	791	0	619	560	661	643	114	47	39	0	0	0	0
2	682	0	551	501	588	574	79	26	19	0	0	0	0
3	574	0	474	431	512	505	54	12	8	0	0	0	0
4	467	0	388	349	423	417	33	4	2	0	0	0	0
5	360	0	300	270	331	326	24	3	2	0	0	0	0
f	900	0	786	721	900	900	1	0	0	0	0	0	0

8sec

pp	921	0	591	422	0	0	921	591	422	921	591	422	266
-5	360	339	19	9	13	4	21	19	9	0	0	0	0
-4	467	427	36	23	27	11	40	36	23	0	0	0	0
-3	574	499	68	44	58	36	75	68	44	0	0	0	0
-2	682	521	149	108	133	93	161	149	108	0	0	0	0
-1	791	472	286	222	265	215	319	286	222	0	0	0	0
0	900	0	687	622	700	671	319	225	203	0	0	0	0
1	791	0	623	564	669	653	114	48	40	0	0	0	0
2	682	0	562	513	595	583	78	27	21	0	0	0	0
3	574	0	475	438	513	505	51	12	7	0	0	0	0
4	467	0	386	353	424	418	33	4	2	0	0	0	0
5	360	0	301	269	328	323	27	4	2	0	0	0	0
f	900	0	780	720	900	900	0	0	0	0	0	0	0

16sec

pp	924	0	587	411	0	0	924	587	411	924	587	411	269
-5	360	339	17	8	11	3	21	17	8	0	0	0	0
-4	467	427	35	22	27	11	40	35	22	0	0	0	0
-3	574	499	67	39	58	33	75	67	39	0	0	0	0
-2	682	522	149	103	131	92	160	149	103	0	0	0	0
-1	791	467	290	226	266	218	324	290	226	0	0	0	0
0	900	0	689	619	697	664	324	231	206	0	0	0	0
1	791	0	621	563	668	651	114	50	42	0	0	0	0
2	682	0	554	508	590	575	76	26	21	0	0	0	0
3	574	0	473	435	503	492	54	12	9	0	0	0	0
4	467	0	386	355	413	408	37	4	2	0	0	0	0
5	360	0	300	273	326	320	25	4	2	0	0	0	0
f	900	0	781	722	900	900	0	0	0	0	0	0	0

jwcdg-3

1sec

pp	1135	0	588	394	0	0	1135	588	394	1135	588	394	468
-5	360	342	16	8	11	4	18	16	8	0	0	0	0
-4	467	426	38	20	28	11	41	38	20	0	0	0	0
-3	574	495	71	41	56	28	79	71	41	0	0	0	0
-2	682	521	147	103	121	81	161	147	103	0	0	0	0
-1	791	453	292	210	261	191	338	292	210	0	0	0	0
0	900	0	691	613	678	646	371	251	217	0	0	0	0
1	791	0	582	522	614	598	185	59	49	0	0	0	0
2	682	0	505	462	546	535	132	29	22	0	0	0	0
3	574	0	435	398	475	467	93	8	6	0	0	0	0
4	467	0	365	332	396	392	67	6	4	0	0	0	0
5	360	0	282	258	311	308	49	4	3	0	0	0	0
f	900	0	726	659	900	900	63	0	0	0	0	0	0

2sec

pp	1036	0	585	405	0	0	1036	585	405	1036	585	405	368
-5	360	339	18	7	11	3	21	18	7	0	0	0	0
-4	467	425	37	18	28	10	42	37	18	0	0	0	0
-3	574	500	64	38	53	28	74	64	38	0	0	0	0
-2	682	520	148	107	125	87	162	148	107	0	0	0	0
-1	791	453	292	223	269	210	338	292	223	0	0	0	0
0	900	0	696	621	692	651	376	255	221	0	0	0	0
1	791	0	629	572	660	642	145	62	52	0	0	0	0
2	682	0	547	505	577	562	102	32	26	0	0	0	0
3	574	0	465	427	498	485	64	10	6	0	0	0	0
4	467	0	381	345	408	400	49	6	3	0	0	0	0
5	360	0	292	264	315	309	38	4	2	0	0	0	0
f	900	0	775	714	900	900	4	0	0	0	0	0	0

4sec

pp	1012	0	592	410	0	0	1012	592	410	1012	592	410	331
-5	360	340	17	9	11	3	20	17	9	0	0	0	0
-4	467	424	37	20	29	9	43	37	20	0	0	0	0
-3	574	492	69	40	59	29	82	69	40	0	0	0	0
-2	682	517	147	110	127	91	165	147	110	0	0	0	0
-1	791	451	298	221	275	206	340	298	221	0	0	0	0
0	900	0	701	629	702	668	366	252	219	0	0	0	0
1	791	0	647	589	666	650	138	66	53	0	0	0	0
2	682	0	563	521	586	574	94	33	26	0	0	0	0
3	574	0	472	436	504	491	57	11	7	0	0	0	0
4	467	0	382	349	415	409	44	6	3	0	0	0	0
5	360	0	294	266	324	317	33	4	2	0	0	0	0
f	900	0	779	715	900	900	0	0	0	0	0	0	0

8sec

pp	992	0	598	423	0	0	992	598	423	992	598	423	311
-5	360	339	19	10	14	5	21	19	10	0	0	0	0
-4	467	424	37	22	30	12	43	37	22	0	0	0	0
-3	574	493	70	41	61	32	81	70	41	0	0	0	0
-2	682	518	148	114	130	96	164	148	114	0	0	0	0
-1	791	450	300	226	278	217	341	300	226	0	0	0	0
0	900	0	703	632	709	671	372	255	222	0	0	0	0
1	791	0	645	586	677	658	131	64	53	0	0	0	0
2	682	0	566	521	594	582	88	34	28	0	0	0	0
3	574	0	472	436	505	495	53	10	6	0	0	0	0
4	467	0	384	351	410	404	39	6	3	0	0	0	0
5	360	0	294	265	318	312	33	4	2	0	0	0	0
f	900	0	787	721	900	900	0	0	0	0	0	0	0

16sec

pp	983	0	598	423	0	0	983	598	423	983	598	423	301
-5	360	339	18	10	12	3	21	18	10	0	0	0	0
-4	467	424	38	23	31	11	43	38	23	0	0	0	0
-3	574	495	68	41	58	32	79	68	41	0	0	0	0
-2	682	519	147	110	128	94	163	147	110	0	0	0	0
-1	791	449	300	227	281	215	342	300	227	0	0	0	0
0	900	0	709	631	714	671	368	256	220	0	0	0	0
1	791	0	647	586	674	652	129	62	51	0	0	0	0
2	682	0	561	520	591	575	88	31	26	0	0	0	0
3	574	0	478	443	503	491	50	11	7	0	0	0	0
4	467	0	390	356	413	406	39	6	3	0	0	0	0
5	360	0	298	270	321	314	32	4	2	0	0	0	0
f	900	0	788	720	900	900	0	0	0	0	0	0	0

jwcdg-3

1sec

pp	1088	0	591	406	0	0	1088	591	406	1088	591	406	423
-5	360	339	19	7	14	2	21	19	7	0	0	0	0
-4	467	426	36	18	30	9	41	36	18	0	0	0	0
-3	574	499	67	37	55	26	75	67	37	0	0	0	0
-2	682	522	148	109	125	89	160	148	109	0	0	0	0
-1	791	455	294	223	265	204	336	294	223	0	0	0	0
0	900	0	675	605	682	641	374	244	211	0	0	0	0
1	791	0	601	543	640	618	161	60	50	0	0	0	0
2	682	0	529	483	567	551	110	30	23	0	0	0	0
3	574	0	447	409	485	475	77	9	7	0	0	0	0
4	467	0	371	332	395	390	57	6	3	0	0	0	0
5	360	0	286	254	305	299	43	4	2	0	0	0	0
f	900	0	767	697	900	900	0	0	0	0	0	0	0

2sec

pp	1012	0	596	419	0	0	1012	596	419	1012	596	419	336
-5	360	339	17	8	10	3	21	17	8	0	0	0	0
-4	467	424	39	19	30	11	43	39	19	0	0	0	0
-3	574	493	71	42	57	31	81	71	42	0	0	0	0
-2	682	518	149	113	129	95	164	149	113	0	0	0	0
-1	791	455	294	224	273	212	336	294	224	0	0	0	0
0	900	0	704	630	702	665	367	254	221	0	0	0	0
1	791	0	636	573	664	639	141	64	51	0	0	0	0
2	682	0	555	511	585	571	92	31	27	0	0	0	0
3	574	0	471	430	494	481	60	11	7	0	0	0	0
4	467	0	385	349	405	399	43	6	3	0	0	0	0
5	360	0	295	265	314	309	32	4	2	0	0	0	0
f	900	0	776	713	900	900	0	0	0	0	0	0	0

4sec

pp	1001	0	601	432	0	0	1001	601	432	1001	601	432	321
-5	360	340	17	9	11	3	20	17	9	0	0	0	0
-4	467	423	38	23	31	11	44	38	23	0	0	0	0
-3	574	493	69	41	59	28	81	69	41	0	0	0	0
-2	682	519	148	115	128	95	163	148	115	0	0	0	0
-1	791	451	303	231	281	219	340	303	231	0	0	0	0
0	900	0	704	632	709	670	369	253	219	0	0	0	0
1	791	0	642	579	671	650	136	62	51	0	0	0	0
2	682	0	563	516	591	575	89	31	25	0	0	0	0
3	574	0	476	434	504	493	58	12	7	0	0	0	0
4	467	0	389	350	416	412	41	6	3	0	0	0	0
5	360	0	298	268	322	318	33	4	2	0	0	0	0
f	900	0	786	717	900	900	0	0	0	0	0	0	0

8sec

pp	995	0	591	425	0	0	995	591	425	995	591	425	319
-5	360	340	17	10	11	3	20	17	10	0	0	0	0
-4	467	424	37	21	30	10	43	37	21	0	0	0	0
-3	574	496	67	44	55	33	78	67	44	0	0	0	0
-2	682	517	148	113	125	95	165	148	113	0	0	0	0
-1	791	454	296	225	277	214	337	296	225	0	0	0	0
0	900	0	711	637	705	669	371	258	222	0	0	0	0
1	791	0	644	582	665	646	135	63	51	0	0	0	0
2	682	0	562	520	583	570	91	32	26	0	0	0	0
3	574	0	480	442	502	490	52	11	7	0	0	0	0
4	467	0	387	353	410	404	39	5	2	0	0	0	0
5	360	0	300	272	322	316	30	4	2	0	0	0	0
f	900	0	780	715	900	900	0	0	0	0	0	0	0

16sec

pp	977	0	595	423	0	0	977	595	423	977	595	423	301
-5	360	340	18	11	12	4	20	18	11	0	0	0	0
-4	467	423	38	24	31	12	44	38	24	0	0	0	0
-3	574	494	68	41	58	31	80	68	41	0	0	0	0
-2	682	519	149	111	128	94	163	149	111	0	0	0	0
-1	791	456	295	224	272	212	335	295	224	0	0	0	0
0	900	0	706	633	708	670	371	254	221	0	0	0	0
1	791	0	644	583	670	651	136	62	51	0	0	0	0
2	682	0	565	521	588	575	91	33	26	0	0	0	0
3	574	0	476	435	504	495	57	12	8	0	0	0	0
4	467	0	389	353	417	411	41	6	3	0	0	0	0
5	360	0	301	271	326	321	29	4	2	0	0	0	0
f	900	0	782	716	900	900	0	0	0	0	0	0	0

Minimal Evaluation

1sec

0	900	0	708	615	743	677	374	277	221	0	0	0	0
1	791	0	607	544	663	633	161	66	51	0	0	0	0
2	682	0	532	484	580	559	110	33	24	0	0	0	0
3	574	0	448	409	494	479	77	10	7	0	0	0	0
4	467	0	371	332	401	393	57	6	3	0	0	0	0
5	360	0	286	254	309	302	43	4	2	0	0	0	0
f	900	0	767	697	900	900	0	0	0	0	0	0	0

2sec

0	900	0	611	517	760	700	367	284	231	0	0	0	0
1	791	0	589	524	685	654	141	69	54	0	0	0	0
2	682	0	508	462	596	577	92	34	28	0	0	0	0
3	574	0	427	386	500	484	60	11	7	0	0	0	0
4	467	0	351	314	410	401	43	7	3	0	0	0	0
5	360	0	265	235	316	311	32	4	2	0	0	0	0
f	900	0	653	590	900	900	0	0	0	0	0	0	0

MaltParser without Reanalysis

0	900	0	624	300	823	534	381	300	20	0	0	0	0
1	791	0	589	474	731	654	141	71	4	0	0	0	0
2	682	0	506	430	633	592	89	37	0	0	0	0	0
3	574	0	422	374	532	517	57	13	0	0	0	0	0
4	467	0	348	312	436	428	38	6	0	0	0	0	0
5	360	0	268	237	337	331	29	4	0	0	0	0	0
f	900	0	638	576	900	900	0	0	0	0	0	0	0

Bibliography

Aist, Gregory et al. (2007). "Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods". In: *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue*. Ed. by Ron Artstein and Laure Vieu. Trento, Italy, pp. 149–154 (cit. on p. 5).

Beuck, Niels (forthcoming). "Anticipatory Incremental Language Processing in Multi-Modal Context". PhD thesis. Universität Hamburg (cit. on pp. 11, 19, 21).

Beuck, Niels and Arne Köhn (2012). "Incremental Dependency Parsing with WCDG". Talk given at the Oberseminar "Linguistic Modeling and its Interfaces", Universität Tübingen (cit. on pp. 20, 46).

Beuck, Niels, Arne Köhn, and Wolfgang Menzel (2011a). "Decision Strategies for Incremental POS Tagging". In: *Proceedings of the 18th Nordic Conference of Computational Linguistics NODALIDA 2011*. Ed. by Bolette Sandford Pedersen, Gunta Nešpore, and Inguna Skadina. Vol. 11. NEALT Proceedings. Northern European Association for Language Technology (NEALT), pp. 26–33 (cit. on p. 9).

– (2011b). "Incremental parsing and the evaluation of partial dependency analyses". In: *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011 (cit. on pp. 7, 11, 26, 46).

Boyd, Adriane (2012). "Detecting and Diagnosing Grammatical Errors for Beginning Learners of German: From Learner Corpus Annotation to Constraint Satisfaction Problems". PhD thesis. Ohio State University (cit. on p. 9).

Brants, Thorsten (2000). "TnT: a statistical part-of-speech tagger". In: *Proceedings of the sixth conference on Applied natural language processing*. Seattle, Washington: Association for Computational Linguistics, pp. 224–231 (cit. on p. 25).

Brants, Thorsten, Wojciech Skut, and Hans Uszkoreit (2003). "Syntactic Annotation of a German Newspaper Corpus". In: *Treebanks*. Ed. by Anne Abeillé and Nancy Ide. Vol. 20. Text, Speech and Language Technology. Springer Netherlands, pp. 73–87. ISBN: 978-94-010-0201-1 (cit. on p. 25).

Demberg-Winterfors, Vera (2010). "A Broad-Coverage Model of Prediction in Human Sentence Processing". PhD thesis. University of Edinburgh (cit. on p. 10).

Foth, Kilian A. (1999). "Transformationsbasiertes Constraint-Parsing". Diplomarbeit. Universitaet Hamburg, Fachbereich Informatik (cit. on pp. 11, 14).

– (2006). "Hybrid Methods of Natural Language Analysis". PhD thesis. Universität Hamburg (cit. on p. 18).

Foth, Kilian A. 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 Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, pp. 321–328 (cit. on p. 43).

Gómez-Rodríguez, Carlos 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*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1492–1501 (cit. on pp. 43, 60).

Hagenström, Jochen (2001). *Constraint Compiler*. Studienarbeit. AB Natürlichsprachliche Systeme, Universität Hamburg (cit. on p. 59).

Hagenström, Jochen and Kilian A. Foth (2002). "Tagging for robust parsers". In: *2nd Workshop on Rbust Methods in Analysis of Natural language Data (ROMAND 2002)* (cit. on p. 25).

Hahn, Michael and Detmar Meurers (2011). "On deriving semantic representations from dependencies: A practical approach for evaluating meaning in learner corpora". In: *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011 (cit. on p. 8).

Khmylko, Lidia, Kilian A. Foth, and Wolfgang Menzel (2009). "Co-Parsing with Competitive Models". In: *Proceedings of the International Conference RANLP-2009*. Borovets, Bulgaria: Association for Computational Linguistics, pp. 173–179 (cit. on p. 43).

Kuhlmann, Marco and Joakim Nivre (2006). "Mildly non-projective dependency structures". In: *Proceedings of the COLING/ACL on Main conference poster sessions*. COLING-ACL '06. Sydney, Australia: Association for Computational Linguistics, pp. 507–514 (cit. on p. 60).

McCrae, Patrick, Kilian A. Foth, and Wolfgang Menzel (2008). "Modelling Global Phenomena with Extended Local Constraints". In: *Proceedings of the 5^{th} International Workshop on Constraints and Language Processing (CSLP 2008, Hamburg, Germany)*. Ed. by Jørgen Villadsen and Henning Christiansen. Roskilde University, pp. 48–60 (cit. on p. 18).

McDonald, Ryan, 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. New York City, New York: Association for Computational Linguistics, pp. 216–220 (cit. on p. 43).

Menzel, Wolfgang and Ingo Schröder (1998a). "Constraint-Based Diagnosis for Intelligent Language Tutoring Systems". In: *Proceedings IT & KNOWS, XV. IFIP World Computer Congress*. Wien und Budapest, pp. 484–497 (cit. on p. 9).

– (1998b). "Decision Procedures for Dependency Parsing Using Graded Constraints". In: *Proc. Coling-ACL Workshop on Processing of Dependency-based Grammars*. Ed. by Sylvain Kahane and Alain Polguère. Montreal, Canada, pp. 78–87 (cit. on p. 13).

Meurers, Detmar, 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*. Tübingen, pp. 214–217 (cit. on pp. 55, 56).

Nivre, Joakim (2003). "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Nancy, France, pp. 149–160 (cit. on p. 43).

Nivre, Joakim et al. (2007). "MaltParser: A language-independent system for data-driven dependency parsing". In: *Natural Language Engineering* 13.2, pp. 95–135 (cit. on pp. 11, 44).

Schröder, Ingo (2002). "Natural Language Parsing with Graded Constraints". PhD thesis. Universität Hamburg (cit. on p. 11).

Skantze, Gabriel and David Schlangen (2009). "Incremental Dialogue Processing in a Micro-Domain". In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*. Athens, Greece, pp. 745–753 (cit. on p. 9).

Sturt, Patrick and Vincenzo Lombardo (2005). "Processing Coordinated Structures: Incrementality and Connectedness". In: *Cognitive Science* 29.2, pp. 291–305. ISSN: 1551-6709 (cit. on p. 5).

Ich versichere, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Diese Arbeit wurde in keinem anderen Prüfungsverfahren eingereicht. Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden. Dies gilt ausdrücklich auch für die digitale Version dieser Arbeit.