Rule-based Dependency Parse Collapsing and Propagation for German and English

Eugen Ruppert and **Jonas Klesy** and **Martin Riedl** and **Chris Biemann**
FG Language Technology, Computer Science Department
Technische Universität Darmstadt
{eugen.ruppert,riedl,biem}@cs.tu-darmstadt.de, jonas.klesy@googlemail.com

Abstract

We present a flexible open-source framework that performs dependency parsing with collapsed dependencies. The parser framework features a rule-based annotator that directly works on the output of a dependency parser. Thus, it can introduce dependency collapsing and propagation (de Marneffe et al., 2006) to parsers that lack this functionality. Collapsing is a technique for dependency parses where words, mainly prepositions, are elevated into the dependency relation name. Propagation assigns syntactic roles to all involved items in conjunctions. Currently, only the Stanford parser features these abilities for the English language. Here we introduce a rule-based collapsing engine that can be applied on top of the output of a dependency parser and that was used to re-engineer the rules of the English Stanford parser. Furthermore, we provide the first dependency parser with collapsing and propagation for German. We directly compare our collapsing for English with the one from the Stanford parser. Additionally, we evaluate collapsed and non-collapsed syntactic dependencies extrinsically when used as features for building a distributional thesaurus (DT).

1 Introduction

Dependency parsing is a major pre-processing step for many applications like similarity computation, machine translation or semantic parsing. In de Marneffe et al. (2006), a technique called dependency collapsing was introduced into the Stanford parser. Collapsing is the process of reducing the number of dependencies, by inserting mostly function words into dependency relation names. Considering the sentence *They sit in the car.*, a dependency parser produces two dependency arcs between *sit* and *car* (cf. Figure 1). The first dependency arc connects *sit* to *in* – `prep(sit,in)`, the second connects *in* to *car* – `pobj(in,car)`. These relations offer only limited information; you can *sit in* something, and you can do something *in cars*. Collapsing makes the relation more informative, `prep_in(sit,car)`, indicating that you can *sit in a car*. Due to the enriched information and larger disambiguation capability of the relation, collapsed dependency parsing is often used for word sense disambiguation (Lin, 1997) and for computing similarities between terms (Biemann and Riedl, 2013).

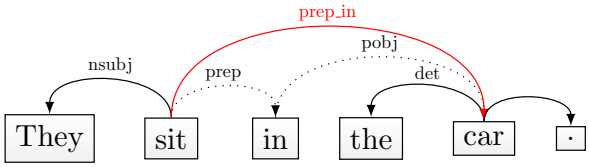

Figure 1: Dependency parser output of the sentence *They sit in the car.* Solid and dotted black lines indicate the standard dependencies, red lines indicate collapsed dependencies. Dotted black lines represent dependencies that are removed after collapsing.

Still, most work regarding collapsing has been done for English and only as variations of the Stanford parsing. Here we introduce a rule-based dependency collapsing framework. We contribute a ruleset for English and also, to our best knowledge, the first German collapsing ruleset.

The collapsing is performed on top of the parser output using our collapsing engine. Due to the rule-based nature of our engine, we can also generate rules for more complex collapsing strategies like dependency propagation. Propagation of dependencies is used to transitively propagate dependencies using conjunctions (de Marneffe et al., 2006). Words that are connected by a conjunction receive

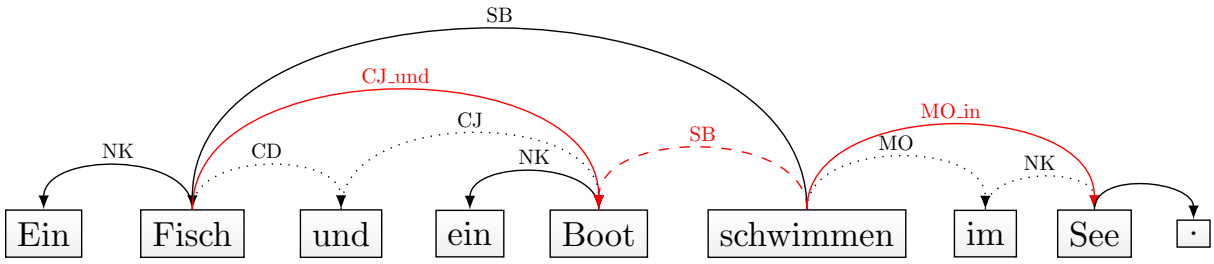

Figure 2: Depencency parser output of the sentence *Ein Fisch und ein Boot schwimmen im See. (A fish and a boat are swimming in the lake.)* Solid and dotted black lines indicate the standard dependencies, solid red lines indicate collapsed dependencies. Dotted black lines represent dependencies that are removed after collapsing. Dashed red line represents the propagated dependency, where the subject (SB) dependency was propagated to *Boot*, as it is connected via the conjunction *und* with *Fisch*, the subject identified by the parser.

the same dependencies, e.g. in Figure 2, *Fisch (fish)* and *Boot (boat)* are both subjects (SB) of the verb *schwimmen (to swim)*, even though only *Fisch* is directly connected to the verb in the original parse.

Furthermore, this approach allows rulesets to be adapted to the tagsets and dependency relation types of different parsers, allowing to apply the functionality to different parsers. We demonstrate the language independence of our framework by transforming the English ruleset to be applicable to German dependency parses. The impact of collapsing and dependency propagation is shown extrinsically based on a distributional similarity evaluation for different corpus sizes. Our collapsing technique is applied on uncollapsed Stanford output to demonstrate the quality of our collapsing rules and to compare the results with its built-in collapsing rules.

2 Related Work

Dependency parsing can help many linguistic applications, especially if they are geared towards semantic representation of text. Since sentences are represented as a lattice of dependencies, mostly originating from the verb, different surface forms of a sentence can result in the same dependency graph. This is important for languages with a variable word order (Dubey and Keller, 2003). In German, the sentence *I have seen the dog*, can have two surface forms: *Den Hund habe ich gesehen (the dog have I seen)* and *Ich habe den Hund gesehen (I have the dog seen)*, even though the first form is marked.

Klein and Manning (2003) introduced the Stanford parser, a dependency parser that extracts dependencies from probabilistic context free grammar (PCFG; Johnson, 1998) constituent parses. Dependency collapsing for this parser was added in (de Marneffe et al., 2006). The Stanford dependency representation (de Marneffe and Manning, 2008) connects two words with a directed and typed dependency relation.

Dubey and Keller (2003) introduced the first PCFG parser for German. The Mate-tools parser (Bohnet, 2010; Seeker and Kuhn, 2012) currently offers one of the best dependency parsing performances for German. However, it does not feature dependency collapsing. Therefore we introduce collapsing for German on top of the Mate-tools parser output.

Our collapsing engine is based on the UIMA framework (Ferrucci and Lally, 2004; Ogren and Bethard, 2009). While there exist generic frameworks to apply rules to UIMA annotations such as UIMA Ruta (Kluegl et al., 2014), we have opted to develop our own processing, in order to be able to tailor our framework specifically for the needs of dependency collapsing and propagation.

Dependency parsing is often used to generate features for information extraction tasks. E.g. in the event extraction task of BioNLP'09 (Kim et al., 2009), 80 % of the participants and all of the top performing teams used dependency features. The best team (Björne et al., 2009) directly worked on the dependency graph to identify relations and extract events. This could be facilitated with our dependency processing framework, where lists of trigger words and protein names can be identified with their relations.

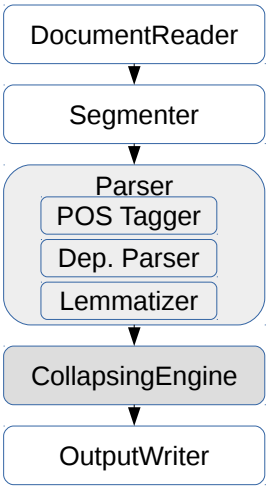

Figure 3: Processing pipeline in the UIMA framework, with exchangeable annotator components

3 Method

3.1 Overview

Our framework works with parsers for any language that produce uimaFIT (Ogren and Bethard, 2009) annotations and transforms the results according to the custom defined rules given in the rules file. We use the framework to perform dependency collapsing and propagation for the output of the Stanford parser and the German Mate-tools parser.

The UIMA framework allows to create flexible annotation pipelines, where each component can be exchanged by other components. This enables users to exchange components like tokenizers and parsers to create individual processing pipelines. In Figure 3 we show the pipeline used in our framework. Documents are read by a document reader and first segmented into sentences and tokens. Afterwards, we run a dependency parser component (that can include a part-of-speech tagger and a lemmatizer) and apply the collapsing engine on the dependency parser annotations. Last, the collapsed dependencies are written to the disk i.e. in the CoNLL format. The pipelines for English and German are freely available for download[1].

3.1.1 Collapsing Engine

The main contribution of our collapsing framework is the collapsing engine that operates on UIMA annotations produced by dependency parsers[2]. It uses rule files that define collapsing triggers and the instructions for dependency collapsing[3]. A collapsing trigger can be defined on word, lemma and POS level and the relationship of multiple tokens. If a match is found by the collapsing engine, the system applies the operations defined in the rule file to the parse, most commonly creating a new dependency with a specified name.

Since we have found it advantageous to perform collapsing and propagation in several stages, we have introduced the capability to create different stages that are executed in succession. Each stage matches dependencies and creates new ones, as specified in a rule. The matched dependencies can be kept for later stage or marked to be removed after the stage is finished.

3.2 Rule Format

3.2.1 Stages

A rule file consists of multiple stages that are executed sequentially. Each stage consists of a set of rules that are applied synchronously. Every rule has access to the dependency annotations that were present when the previous stage finished. Multiple stages are needed, if an already processed annotation should be further processed. For example, in one stage the collapsed dependency `prep_such` is generated from the phrase *animals such as birds*. In the next stage, this should be refined to `prep_such_as`, which creates a direct dependency relation between *animals* and *birds*.

3.2.2 Rules

A rule has the following scheme:

```
##<Rule name>
<element><ID>:<regex>
<r,d><FromID>_<ToID>:<regex>
<element><ID>:<regex>
...
from:<FromID>
to:<ToID>
relationName:name
```

It defines a collapsing match and the new dependency annotation that is created from this match. A collapsing match is defined between multiple `element` items and their relations. An element

[1] We provide the framework with rulesets for English and German under the permissive ASL 2.0 license at `http://jobimtext.org/dependency-collapsing/`

[2] The DKPro Core framework (Eckart de Castilho and Gurevych, 2014) provides UIMA wrappers for a large number of NLP components.

[3] Dependency propagation is also defined in the rulesets. The framework allows to create and modify custom operations on the dependency graph. For brevity, we only mention dependency collapsing in the rule description.

can be matched by word (w), POS tag (p) or a lemma (l). Matching of elements is performed using regular expressions and thus allows any matching, i.e. lists or string matches. Each element also contains an integer ID that is used to identify relations and to create new ones. Relations between elements (specified by ID), are matched by a relation name, which can also be defined as a regular expression.

If all of the conditions are met, then a new relation with a custom relationName is created between the elements specified in from and to. A matched relation will be removed after a stage, if it is identified as a removable relation (r). Durable relations (d) are available in subsequent stages.

Dependency Collapsing Here, we present an exemplary rule file for dependency collapsing:

```
STAGE:CollapsePrepositions
{
  ##prep_in Rule
  w1:.*
  r1_2:prep
  p2:IN|TO|VBG
  r2_3:pobj
  w3:.*
  relationName:prep_{w2}
  from: 1
  to: 3
}
```

In this example ruleset that collapses prepositions, only one stage is defined and in this stage only one rule exists. This rule will match all dependencies that have the dependency type prep, originate from any word (w1:.*) and end in a word with any of the specified POS tags (p2:IN|TO|VBG). Additionally, there should be another pobj dependency from p2 to any other word (w3:.*).

If such a match is found, the collapsing engine will create a new relation between w1 and w3, collapsing the preposition word into the relation name (relationName:prep_{w2}). And as the matched dependencies are marked as 'removable' (*r*1_2:prep, *r*2_3:pobj), they will be deleted after the stage is finished. To keep the dependency for later stages, the relation should be specified as durable, e.g. *d*1_2:prep.

Dependency propagation A rule set for propagation is shown below. It matches any words (wN:.*) with N=1,2,3 that form the following dependency constellation: there is a dobj or nsubj relation between words 1 and 2; additionally there should be a conjunction (conj) between words 2 and 3. If these conditions apply, the same relation as between words 1 and 2 (relationName:d1_2) is propagated as a new relation between w1 and w3.

```
STAGE:Propagation
{
  ##subj/obj propagation
  w1:.*
  d1_2:dobj|nsubj
  w2:.*
  d2_3:conj.*
  w3:.*
  relationName:{d1_2}
  from:1
  to:3
}
```

3.3 Rulesets

We have compiled rulesets for English and German, each with and without dependency propagation. For re-engineering the collapsing rules for English, we compared the collapsing from the Stanford parser to our collapsing rules and stopped after we achieved sufficient overlap. Starting from preposition collapsing, we addressed the largest error class in each rule writing stage to identify more complex dependency matchings. Propagation rules were added in an additional stage.

German rules are corresponding to the English rules, with adjusted dependency types, and – for lexicalized rules – translated words. In few cases, the parsers produce slightly different dependency structures (e.g. switch of governor / dependent). Also, some rules only apply for English and are thus left out of the German ruleset.

Overall, the rulesets now feature 3 stages for collapsing and 4 stages for collapsing with propagation. For English, we have compiled 43 rules for collapsing (46 with propagation). For German, there are 26 rules for collapsing (29 with propagation).

4 Evaluation

To evaluate our collapsing tool and the rulesets, we performe two kinds of evaluation. First, we intrinsically compare to the English Stanford parser[4]. Second, we use the dependency parser output for similarity computation (see Section 5). This allows us to show the positive impact of collapsing and dependency propagation on semantic tasks.

[4] As there exists no parser that performs collapsing for German, we cannot compare to a German parser intrinsically.

Table 1: Number of dependencies of our method and Stanford dependency parsing

	both	not in our method	not in Stanford
collapsed dep.	27,602	405	664
all dep.	165,594	1,067	1,850

Table 2: Number of dependencies of our method and Stanford dependency parsing, with dependency propagation

	both	not in our method	not in Stanford
propagated dep.	29,271	611	566
all dep.	168,201	7,377	4,036

4.1 Intrinsic Evaluation of the English Ruleset

As there is no evaluation set for collapsing available, we perform an intrinsic evaluation. We compare directly to the Stanford parser on a sample of 10,000 unannotated English sentences. On these sentences, we perform dependency parsing using 1) the Stanford dependency parser with collapsing and 2) the Stanford dependency parser without any collapsing, where we apply the collapsing using our tool.

From these sentences, 927 do not contain any collapsed dependency. The number of dependency relations is reduced from 200,541 to 194,669 with collapsing from the Stanford dependency parser. For dependency collapsing, we observe that our method only misses 1.45 % of the collapsed dependencies (see Table 1). Additionally, we observe that our collapsing engine performs collapsing more often than the Stanford collapsing, e.g. we always collapse prepositions, which is often not performed by the Stanford collapsing. On the other hand, we most commonly miss collapsed dependency relations which collapse more than one word into the relation name, like `prep_away_from` or `prep_out_of`.

In a further analysis we checked the different dependency types for each sentence. For this, we extracted all differences between the Stanford collapsed dependencies and our collapsed dependencies and counted how often they occur. Manually checking the most frequent 100 of these differences (occurring in 592 sentences), we figured out that in 27 of these discrepancies (represented by 32 sentences) we could not decide which system performs better, when checking instances manually. For 72 dependency patterns (204 sentences), the Stanford collapsing performs better, whereas our system yields better results for 78 patterns (168 sentences).

On the basis of sentences, this gives a balanced picture of our system, with Stanford being correct in 50.6 % of the cases and 41.6 % where our system is correct. Our system often misses collapsed dependency relations that collapse more then one word into the relation name. On the other hand, the Stanford collapsing often misses collapsing the construct *according to* into the dependency type, which is performed more often with our system.

We also analyzed the performance of dependency propagation, as shown in Table 2. Here we miss slightly more dependencies and we also observe that the total number of dependencies increases. This is because most propagated dependencies are added to the dependency graph, so that mostly no other dependencies are deleted by propagation.

4.2 Extrinsic Evaluation on Similarity Computation

We follow Riedl et al. (2014) to evaluate the performance of collapsing extrinsically: We use dependency features to build distributional thesauri (DT) and evaluate the thesauri against the lexical resources WordNet (Fellbaum, 1998) and GermaNet (Hamp and Feldweg, 1997; Henrich and Hinrichs, 2010). Evaluation is performed by the WordNet-based Path measure (Pedersen et al., 2004). The Path measure is the inverse path length between two words in WordNet / GermaNet. We use a word list of 1000 frequent and 1000 infrequent nouns for each language. The word lists were sampled from the corpora with the requirement that they are also present in WordNet / GermaNet. For English, we use the same words that were also used in Weeds and Weir (2003). From the DT entries of these words, we extract the top 5 similar words and calculate the average inverse path length. Since the similarities in a DT entry should be high, this means that the path length should remain relatively low, e.g. synonyms (length 1, same synset), hyponyms/hypernyms (length 2, term – hypernym / hyponym) or co-hyponyms (length 3, term – hypernym – related term). The Path measure

Table 3: Extracted context features for *Mary and John sit in the car*, using the Stanford parser, without collapsing

Term	Context Feature	
Lemma	Lemma	Dependency
Mary#NP	sit#VB	-nsubj
Mary#NP	John#NP	conj
Mary#NP	and#CC	cc
and#CC	Mary#NP	-cc
John#NP	Mary#NP	-conj
sit#VB	Mary#NP	nsubj
sit#VB	in#IN	prep
in#IN	sit#VB	-prep
in#IN	car#NN	pobj
the#DT	car#NN	-det
car#NN	the#DT	det
car#NN	in#IN	-pobj

Table 4: Extracted context features for *Mary and John sit in the car*, using the Stanford parser, with collapsing and propagation (propagated dependencies in italics)

Term	Context Feature	
Lemma	Lemma	Dependency
Mary#NP	John#NP	conj_and
Mary#NP	sit#VB	-nsubj
John#NP	Mary#NP	-conj_and
John#NP	*sit#VB*	*-nsubj*
sit#VB	Mary#NP	nsubj
sit#VB	*John#NP*	*nsubj*
sit#VB	car#NN	prep_in
the#DT	car#NN	-det
car#NN	the#DT	det
car#NN	sit#VB	-prep_in

is an established way for relative comparison of DTs against lexical-semantic networks. Since the measure is heavily dependent on the structure and coverage of the semantic network, scores are not comparable across languages.

5 Experimental Settings

5.1 Context Feature Representations

To evaluate the performance of our custom collapsing rules, we create distributional thesauri (DT) using dependency parse relations as context features. We use the PCFG Stanford parser (Klein and Manning, 2003) to parse English. As the Stanford parser has options for collapsing and propagation (de Marneffe et al., 2006), we can directly compare DTs that are computed with different Stanford parser settings (collapsing, propagation) with the performance achieved when using our collapsing rules. For German, we use the Mate-tools parser (Bohnet, 2010; Seeker and Kuhn, 2012). As the Mate-tools parser does not feature collapsing, we cannot directly compare results. In fact, the lack of collapsing for German was the main motivation of our work. Instead, we can measure the quality improvements that collapsing introduces for similarity computation.

For higher precision, words are lemmatized leading to context features as shown in Tables 3 and 4. For the purpose of computing semantic similarity, we model dependencies in both directions by adding 'inverse' dependencies (e.g. `-nsubj`). As dependency relations are directed, not using inverse dependencies affects similarity computations of words commonly used as dependents in dependencies (e.g. *Mary* in Table 3).

Tables 3 and 4 demonstrate the impact of collapsing and propagation. Even though Table 3 contains more dependency relations, they are less discriminative than the collapsed dependencies in Table 4. Dependency propagation adds a dependency from *John* to *sit*, leading to a higher recall in the similarity computation.

5.1.1 Trigram Baseline

As a baseline system, we use a context representation of trigrams. This is an unsupervised, language-independent feature extractor that uses the left and right neighboring words as a combined context feature, e.g. extracting the term *likes* and the context feature *Mary_@_John* from the phrase *Mary likes John*. This feature representation is language-agnostic and thus it can be used for most languages with an established tokenization. For a fair comparison, we run this baseline in two configurations: first, without any linguistic processing and second, for an analysis of the impact of dependency features, we lemmatize all words.

5.1.2 Stanford Parser

stanford_basic Dependency parsing without collapsing or propagation

stanford_collapsed Dependency parsing with built-in collapsing

stanford_collapsed_prop Dependency parsing with built-in collapsing and propagation

stanford_basic_custom_collapsing Dependency parsing without built-in collapsing, applying our English collapsing rules afterwards

stanford_basic_custom_collapsing_prop
Dependency parsing without built-in collapsing, applying our English collapsing rules with propagation afterwards

5.1.3 Mate-tools Parser

matetools Dependency parsing without collapsing (no built-in collapsing available)

matetools_custom_collapsing Dependency parsing without collapsing, applying our German collapsing rules afterwards

matetools_custom_collapsing_prop Dependency parsing without collapsing, applying our German collapsing rules with propagation afterwards

5.2 Data

We chose datasets of different sizes for German and English. Both of these sets consist of randomly sampled sentences from news articles from the Leipzig Corpora Collection (Richter et al., 2006).

To assess the impact of training data size, we have taken samples of different sizes. These samples were taken from the full sets and include the following sizes: 0.1M, 1M and 10M sentences.

5.3 Similarity Computation

The similarity computation is performed using the JoBimText framework (Biemann and Riedl, 2013). It incorporates UIMA annotators for feature extraction, so we can add the collapsing annotator to the feature extraction pipeline on top of the parser output. We use the settings from Riedl and Biemann (2013). Terms and their context features are extracted from the input text. In our experiments, we use neighboring words and dependency parse features. We calculate the corpus frequencies for terms, context features and the term–context feature combinations. Using these frequencies, we compute the Lexicographers Mutual Information significance measure (Evert, 2005) between terms and contexts. We prune context features that occur with more than 1000 words and only keep the top 1000 most significant context features per word. Similarity between words is computed by counting the number of context features that two words share. The result is a distributional thesaurus where for each word we obtain up to 200 similar words, of which we evaluate on the top 5 only.

6 Results

Tables 5 and 6 show the evaluation results. In line with previous results, a larger corpus size results in more accurate similarities. Corpus size has a larger impact on rare nouns, where more input text is required to get sufficient 'signals' for similarity calculation. This becomes most apparent for German, where, due to the more complex morphology and noun compounding, we find about twice as many different word forms as in English (2.8M vs. 1.4M words) in the 10M corpora.

Overall, a larger corpus size leads to better DTs, which is expected. Compared to the baselines, dependency path features also improve the DTs. This is due to the structured, more accurate features and the fact that with dependency parsing, we obtain several context features for most words in a sentence (cf. Table 3).

Of the structural alternatives, collapsing brings a large boost and propagation usually adds a small improvement on top, especially for rare nouns. Therefore, we conclude that collapsing and propagation help improve the similarity computation for both languages. Some example DT entries can be seen in Table 7. Even though the DT entries do not change much, using collapsing and propagation puts the more similar terms on top (e.g. *office, bank* or *Bahnhof (railway station)*), while less similar terms like *student or Straße (street)* are ranked lower. Collapsing and propagation do not only lead to more accurate similarities, they also improve the recall. For rare German nouns, the distributional thesaurus contains similarities for only 592 out of 1000 test nouns, when computed using the Mate-tools parser alone. Collapsing improves the recall to 700 nouns and propagation offers an additional increase to 703 words.

The extrinsic comparison of our collapsing engine with the custom rules shows a comparable performance to Stanford parsing with collapsing. The scores show almost no difference for collapsing and up to 0.002 score points difference for propagation, indicating – as in Section 4.1 – that the dependencies produced by our rulesets are very similar to Stanford parser dependencies.

7 Conclusion

In this paper, we have presented a flexible framework for collapsing, which can be applied on top of arbitrary dependency parser outputs. We release, to our knowledge, the first dependency collapsing and

Table 5: Average WordNet path scores for the top 5 most similar words in a DT, considering different corpus sizes and word lists of frequent / rare English nouns

Method	Corpus Size (freq. nouns) 0.1M	1M	10M	Corpus Size (rare nouns) 0.1M	1M	10M
baseline	0.178	0.228	0.280	0.044	0.129	0.190
baseline_lemma	0.187	0.240	0.280	0.065	0.137	0.194
stanford_basic	0.210	0.272	0.302	0.096	0.183	0.229
stanford_collapsed	*0.225*	*0.292*	*0.322*	0.100	0.193	*0.241*
stanford_collapsed_prop	0.222	0.291	0.319	*0.106*	*0.200*	*0.241*
stanford_basic_custom_coll	0.224	0.291	0.321	0.101	0.193	*0.241*
stanford_basic_custom_coll_prop	0.224	0.290	0.319	0.104	0.195	*0.241*

Table 6: Average GermaNet path scores for the top 5 most similar words in a DT, considering different corpus sizes and word lists of frequent / rare German nouns

Method	Corpus Size (freq. nouns) 0.1M	1M	10M	Corpus Size (rare nouns) 0.1M	1M	10M
baseline	0.127	0.165	0.229	0.000	0.009	0.056
baseline_lemma	0.128	0.183	0.254	0.000	0.009	0.062
matetools	0.144	0.208	0.265	*0.001*	0.017	0.081
matetools_custom_coll	*0.149*	*0.217*	0.273	*0.001*	0.022	0.103
matetools_custom_coll_prop	0.147	*0.217*	*.0.274*	*0.001*	*0.023*	*0.104*

Table 7: Top 5 most similar words for English and German nouns, and their average Path scores

English: *branch*			German: *Bahnhofsplatz (station square)*		
stanf_basic	stanf_coll	stanf_coll_prop	matetools	matetools_coll	matetools_coll_prop
student	office	office	Innenstadt (city)	Bahnhof (station)	Bahnhof (station)
area	bank	bank	Bahnhof (station)	Straße (street)	Straße (street)
official	company	company	Hauptbahnhof (station)	Innenstadt (city)	Innenstadt (city)
bank	project	group	Straße (street)	Hauptbahnhof (station)	Hauptbahnhof (station)
business	director	director	Stadtteil (district)	Schulhof (school yard)	Platz (square)
0.152	0.153	0.170	0.152	0.169	0.219

propagation mechanism for German. This framework was used to implement collapsing and transitive propagation of dependencies over conjunctions for English and German. We have shown that collapsing improves the quality of distributional models. Also, our English ruleset offers comparable performance to the built-in Stanford collapsing rules. Since it is a separate component, it can be used to add collapsing functionality to any other English dependency parser. The parser framework is freely available for download and is accessible under a permissive license. We supply extensible rulesets for English based on the Stanford dependency tagset and for German based on the TIGER (Brants et al., 2002) tagset.

Since we have demonstrated the utility of dependency collapsing and propagation for semantic tasks such as distributional similarity, we believe that the German collapsing and propagation rules are a valuable addition for German NLP. For the future, we would hope that other groups might add collapsing rules for more languages.

Acknowledgment

This work has been supported by the German Federal Ministry of Education and Research (BMBF) within the context of the Software Campus project LiCoRes under grant No. 01IS12054.

References

Chris Biemann and Martin Riedl. 2013. Text: Now in 2D! a framework for lexical expansion with contextual similarity. *Language Modelling*, 1(1):55–95.

Jari Björne, Juho Heimonen, Filip Ginter, Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2009. Extracting complex biological events with rich graph-based feature sets. In *Proc. Workshop on Current Trends in Biomedical NLP: Shared Task*, pages 10–18, Boulder, CO, USA.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proc. International Conference on Computational Linguistics (COLING 2010)*, pages 89–97, Beijing, China.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proc. Workshop on Treebanks and Linguistic Theories*, pages 24–41, Sofia, Bulgaria.

Marie-Catherine de Marneffe and Christopher D Manning. 2008. The Stanford typed dependencies representation. In *Proc. International Conference on Computational Linguistics (COLING '08)*, pages 1–8, Manchester, UK.

Marie-Catherine de Marneffe, Bill MacCartney, and Christoper D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. Language Resources and Evaluation (LREC 2006)*, pages 449–454, Genova, Italy.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proc. Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 96–103, Sapporo, Japan.

Richard Eckart de Castilho and Iryna Gurevych. 2014. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proc. Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT) at COLING 2014*, pages 1–11, Dublin, Ireland.

Stefan Evert. 2005. *The statistics of word cooccurrences: word pairs and collocations*. Ph.D. thesis, IMS, Universität Stuttgart.

Christiane Fellbaum. 1998. *Wordnet. An Electronic Lexical Database*. MIT Press, Cambridge, MA.

David Ferrucci and Adam Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering 2004*, 10(3-4):327–348.

Birgit Hamp and Helmut Feldweg. 1997. GermaNet – a lexical-semantic net for German. In *Proc. ACL Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications (ACL-EACL '97)*, pages 9–15, Madrid, Spain.

Verena Henrich and Erhard Hinrichs. 2010. GernEdiT – the GermaNet editing tool. In *Proc. Language Resources and Evaluation (LREC 2010)*, pages 2228–2235, Valletta, Malta.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Compututational Linguistics*, 24(4):613–632.

Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. 2009. Overview of BioNLP'09 shared task on event extraction. In *Proc. Workshop on Current Trends in Biomedical NLP: Shared Task*, pages 1–9, Boulder, CO, USA.

Dan Klein and Christoper D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 423–430, Sapporo, Japan.

Peter Kluegl, Martin Toepfer, Philip-Daniel Beck, Georg Fette, and Frank Puppe. 2014. UIMA Ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering*, pages 1–40.

Dekang Lin. 1997. Using syntactic dependency as local context to resolve word sense ambiguity. In *Proc. Meeting of the Association for Computational Linguistics and Conference of the European Chapter of the ACL (ACL-EACL '97)*, pages 64–71, Madrid, Spain.

Philip Ogren and Steven Bethard. 2009. Building test suites for UIMA components. In *Proc. Workshop on Software Engineering, Testing, and Quality Assurance for NLP (SETQA-NLP 2009)*, pages 1–4, Boulder, CO, USA.

Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. 2004. Wordnet::similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004*, pages 38–41, Boston, MA, USA.

Matthias Richter, Uwe Quasthoff, Erla Hallsteinsdóttir, and Chris Biemann. 2006. Exploiting the Leipzig Corpora Collection. In *Proc. IS-LTC 2006*, pages 68–73, Ljubljana, Slovenia.

Martin Riedl and Chris Biemann. 2013. Scaling to large^3 data: An efficient and effective method to calculate distributional thesauri. In *Proc. Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 884–890, Seattle, WA, USA.

Martin Riedl, Irina Alles, and Chris Biemann. 2014. Combining supervised and unsupervised parsing for distributional similarity. In *Proc. International Conference on Computational Linguistics (COLING 2014)*, pages 1435–1446, Dublin, Ireland.

Wolfgang Seeker and Jonas Kuhn. 2012. Making ellipses explicit in dependency conversion for a German treebank. In *Proc. Language Resources and Evaluation (LREC 2012)*, pages 3132–3139, Istanbul, Turkey.

Julie Weeds and David Weir. 2003. A general framework for distributional similarity. In *Proc. Empirical methods in Natural Language Processing (EMNLP 2003)*, pages 81–88, Sapporo, Japan.