(b) **Mistagging**: A dependency has the wrong POS label (`A Y:`$\underline{\texttt{X}}$` C` should be `A Y:`$\underline{\texttt{W}}$` C`).

2. **Substitution:** A dependency has the wrong attachment, but something else could go in this position with this head (`A` $\underline{\texttt{B}}$ `C` should be `A` $\underline{\texttt{D}}$ `C`).
3. **Comission:** A dependency has the wrong attachment, but nothing else goes in this position with this head (`A` $\underline{\texttt{B}}$ `C` should be `A C`).
4. **Omission:** There is a missing attachment (`A C` should be `A` $\underline{\texttt{B}}$ `C`).

We approach the errors by adding *n*-grams that reflect each type of change, essentially compiling out Levenshtein edits. Given the analysis of the method above, the effect of adding more *n*-grams should serve to increase *precision*, not recall, of errors found: since each dependency relation now has more chances to obtain a higher score, we expect fewer which are still low-scoring, but those cases should be more accurate. Note that: a) each solution is restricted to a single type of edit, some of which are partial substitutions; and b) modifications are for all *n*-grams, not just the rule as a whole [5]. The set-up of adding new *n*-grams also makes clear that the issue is fundamentally one of representation.

Focusing on representation, we hope, makes the insights applicable to other domains, in the same way that representations for corpus annotation error detection have led to, e.g., particular features for native language identification [3] and methods for uncovering annotation definitions [7]. Additionally, although we have only tried individual solutions, future work can explore combinations of solutions; initial results indicate that not all combinations are favorable.

Solution #1a: Removing a dependency label For potentially incorrect dependency labels, our solution is to include *n*-grams which replace a dependency:POS pair with only the POS. Recalling the base formula for calculating an anomaly score in (2) and using a set of modified *n*-grams N', the formula is given in (5) and an instantiation in (6). Here, the POS `CD` replaces `num:CD`.

(5) $$s(e_i) = s_{base} + \sum_{ngrm' \in N' \wedge ngrm' : e_i \in ngrm' \wedge n \geq 3} C(ngrm')$$

(6) s(prep:IN) = s_{base}(prep:IN)
+ C($\underline{\text{CD}}$ NNS-H **prep:IN**)
+ C(`START` $\underline{\text{CD}}$ NNS-H **prep:IN**)
+ C($\underline{\text{CD}}$ NNS-H **prep:IN** `END`)
+ C(`START` $\underline{\text{CD}}$ NNS-H **prep:IN** `END`)

The set of modifications N' is subject to constraints. First, only one item in a rule is replaced at a time. Thus, we obtain `START` $\underline{\texttt{CD}}$ `NNS-H prep:IN END` and `START num:CD NNS-H` $\underline{\texttt{IN}}$ `END`, but not `START` $\underline{\texttt{CD}}$ `NNS-H` $\underline{\texttt{IN}}$ `END`. This keeps the number of new *n*-grams managable and reflects the assumption that it is less likely for there to be multiple contextual errors. The second constraint is not to allow heads to be modified—more of an issue when using a dummy label (solution #2) or removing the item entirely (solution #3). We restrict this because the comparison of rules is based on having similar heads; e.g., only rules with `NNS` heads are compared to the rule in (1). The third constraint only applies when an item is removed (solution #3): items on the edge of an *n*-gram are not allowed to be removed. In this case, we wind up with an already-known bigram, e.g., `num:CD NNS-H prep:IN` becomes `NNS-H prep:IN`. Relatedly, for all solutions we do not modify any bigrams (e.g., `CD NNS-H`), given the questionable nature of bigrams.

Solution #1b: Removing a POS label Having a wrong POS label (`A Y:`$\underline{\texttt{X}}$` C` should be `A Y:`$\underline{\texttt{W}}$` C`) seems at first glance to be mainly a concern for when automatic POS tagging is employed, but, as example (4) illustrates, sometimes the root of an inconsistency can be traced to an improper POS tag (e.g., the problem for (4) is the presence of the Adv(erb) POS tag). We thus remove the POS tag and keep the dependency label (e.g., in (6) replace every `CD` with `num`).

Solution #2: Skipping items To handle an incorrectly-attached sister item when some other item could attach in this position (e.g., A $\underline{\text{M:X}}$ C should be A $\underline{\text{N:Y}}$ C), we replace items in a rule with a dummy token (`SKIP`) (e.g., `START` $\underline{\texttt{SKIP}}$ `NNS-H prep:IN`). This solution is fairly broad, as the dummy `SKIP` stands in for any substituted item (including relabelings, as in solution #1a), as well as the original item.

Solution #3: Removing items When a dependency has the wrong attachment, but nothing else goes in this position (e.g., `A` $\underline{\texttt{B}}$ `C` should be `A C`), the simple modification is to remove a potentially erroneous context item (e.g., `B`). However, training and testing must be handled differently, as removing an item in training would be an incorrect sequence (e.g., `A C` in this case). Thus, we obtain training rules in the basic way (s_{base} in (3)), and then in testing add rules which remove an item.

Solution #4: Inserting items Items which should have been attached to a head (e.g., `A C` should be `A` $\underline{\texttt{B}}$ `C`) lead to positing the insertion of an item between two other items in a rule. While we might want to posit every single intervening token, with every possible label, as the inserted item (`B`), this is cumbersome. For this exploration, we simply ask whether *something* could be inserted between the two items. Training consists of generating extra *n*-grams with dummy `SKIP`s appropriately inserted, exactly as in solution #2. For testing, we then insert `SKIP` items between items in a rule if there is an item between them in the linear string (`A SKIP C`). If neighboring items in a rule are linearly adjacent, no `SKIP` is inserted.

Context changes vs. revision checking The methods of changing items are different than the revision checking algorithm in [8], in that they account for contextual errors, whereas revision checking rescores the item in focus. To examine the score of `A` in `A B C`, for instance, the current solutions simulate a change to `B` or `C` to see how that affects the score of `A`, whereas revision checking notes the effect of revising `A`. Context item changing is rather robust, in that it subsumes rules the system has not seen before. For example, generalizing `A B C` to `A SKIP C` covers `A D C`, too, which may have never been seen before. Thus, the scoring may also be of help for new rules within new kinds of data.

4 Evaluation

4.1 Experimental conditions

We use the WSJ corpus [12] for all experiments, converted to the Stanford dependencies [4], allowing us to vary the training data size and keep constant the testing data. In particular, we train both the parsers and the error detection methods on a small portion of the corpus (section 02) and a larger portion (sections 02–15), and then test on section 00. To increase the variety in parser quality, we use two parsers with differing methods, MaltParser [16] and MSTParser [13]. This set-up is in line with recommendations in [18] for selecting appropriate experimental conditions to robustly test error detection methods.

Error detection precision is given in table 1, i.e., the percentage of parser errors accurately identified for corpus positions under a threshold. Since each method identifies a differing number of positions at differing thresholds, we report precision for *segment sizes*: for the lowest-scoring 5% of tokens, for example, what is precision across the methods? Since the number of positions is fixed for a given segment, an increase in precision means there is a corresponding increase in recall for that segment [18]. We choose low segment sizes (1% and 5% of the testing corpus) because activities like manually correcting annotation for a huge amount of text rely on high precision for a relatively small percentage of the data [18].

4.2 Results

Turning to the trends in table 1, we note a number of things. 1) For small training (02), it almost always helps to have some new representation, i.e., multiple representations indeed work better for situations with little annotated data. This matches our analysis of the method (section 2.2), where multiple perspectives on the data provide more opportunities to pinpoint a spot as erroneous.

2) For large training (02-15), adding the new representations generally helps for checking a small amount of data (1% segment), but becomes less beneficial as more corpus positions are examined (5%). Indeed, precision goes *up* for the 1% segment, but significantly *down* for the 5% segment in moving from the small (02) to the large (02-15) training scenario.

In table 2, which reports the score thresholds for each condition, we can see part of the reason for the downward trends: as more information is added to the model—adding bigrams (moving from High to Wall), adding new representations

	1% seg.	Train: 02		Train: 02-15	
	Condition	Malt	MST	Malt	MST
High	Base	77.0	73.6	86.7	78.7
	No-dep	79.4	76.5	89.8	**83.4**
	No-POS	83.5	78.4	91.8	82.8
	Skip	**90.2**	**84.6**	**95.2**	74.4
	Remove	84.0	80.0	92.9	83.1
	Insert	78.7	75.2	88.0	78.5
Wall	Base	91.2	83.7	93.1	81.8
	No-dep	91.5	85.0	93.9	83.8
	No-POS	92.5	85.0	**94.6**	**84.0**
	Skip	**94.1**	**88.5**	94.3	75.5
	Remove	91.5	88.1	94.3	81.6
	Insert	92.5	84.4	92.8	65.7

	5% seg.	Train: 02		Train: 02-15	
	Condition	Malt	MST	Malt	MST
High	Base	77.0	73.6	**66.8**	**62.5**
	No-dep	79.4	76.5	65.7	61.3
	No-POS	**81.6**	**77.4**	65.4	62.1
	Skip	77.0	71.4	59.8	51.9
	Remove	79.8	74.0	66.0	57.0
	Insert	78.1	75.2	66.3	59.8
Wall	Base	80.4	76.0	**66.6**	60.7
	No-dep	80.4	77.1	65.9	60.5
	No-POS	**81.3**	**77.9**	65.2	**61.4**
	Skip	78.4	71.3	59.8	51.6
	Remove	81.0	75.2	65.4	56.9
	Insert	77.3	67.2	59.8	47.8

Table 1: Precision (%) for 1% & 5% segments (1342 & 6710 positions). Baseline LAS: Malt.02: 81.1%, Malt.02-15: 86.4%, MST.02: 80.5%, MST.02-15: 87.6%.

(comparing Base to other models), or, most crucially, adding more training data and thus more training rules (moving from 02 to 02-15)—the score threshold required to hit 1% or 5% of the data rises, in some cases dramatically. The reason for this is straightforward: by including information from a greater number of (training) rules, the chance of having a low score for a parsed rule is less likely. A score of 158.92, for the Malt.02-15.Wall.Skip case, for example, means that many of the rules flagged here as errors have a great deal of supporting evidence (roughly speaking, 158 instances from the training data support it), and so we should not be surprised that many of the cases are not errors, i.e., precision is low.

The upshot is that multiple representations work best for increasing precision when low-scoring positions account for a large percentage of the data to be corrected, and this tends to happen when: a) the amount of training data is small, and/or b) the amount of data to be corrected is a small percentage of the corpus. (For very large corpora, a small percentage of the corpus is still a large absolute number of positions.) Following the mathematical reasoning above, when more representations are added, a position is unlikely to get a low score by chance, and so low scores mean more, i.e., are more likely to truly indicate errors. The 1% segments confirm this: precision is high for every condition where the score threshold is around zero, whereas the higher thresholds for the Skip (14.11) and Insert (8.05) models lead to lower precision (74.4% and 65.7%, respectively).

Turning to the generally improved precision for the 02-15 training experiments with the 1% segments (vs. 02 training), we believe this improvement stems from the fact that the set of training rules is more accurate. The method is still examining zero-scoring rules, but, as compared to the 02 training experiments, more training data gives a better representation of what an acceptable rule is, and so the zero scores seem to be more meaningful.

	1% seg. Condition	Train: 02 Malt	Train: 02 MST	Train: 02-15 Malt	Train: 02-15 MST
High	Base	0	0	0	0
High	No-dep	0	0	0	0
High	No-POS	0	0	0	0
High	Skip	0	0	2	13
High	Remove	0	0	0	1
High	Insert	0	0	0	0
Wall	Base	0.00	0.00	0.03	0.37
Wall	No-dep	0.00	0.00	0.04	1.03
Wall	No-POS	0.00	0.00	0.13	1.33
Wall	Skip	0.00	0.00	3.14	14.11
Wall	Remove	0.00	0.00	0.20	2.31
Wall	Insert	0.00	0.00	1.05	8.05

	5% seg. Condition	Train: 02 Malt	Train: 02 MST	Train: 02-15 Malt	Train: 02-15 MST
High	Base	4	4	16	14
High	No-dep	4	4	42	42
High	No-POS	4	4	63	51
High	Skip	7	10	156	176
High	Remove	4	4	41	45
High	Insert	4	4	27	28
Wall	Base	0.19	0.45	20.81	19.97
Wall	No-dep	0.52	1.15	46.32	48.35
Wall	No-POS	2.01	1.83	67.20	59.42
Wall	Skip	7.04	11.04	158.92	182.29
Wall	Remove	1.29	2.13	45.23	49.51
Wall	Insert	3.05	6.05	80.59	95.44

Table 2: Score thresholds for 1% & 5% segments (1342 & 6710 positions).

3) The more beneficial models seem to be the ones which maintain the attachments in a rule but make them more abstract (No-dep, No-POS, Skip), and not the models which attempt to actually modify the attachments of the rules (Remove, Insert). This is not necessarily tied in to the score thresholds, either: notice how the No-POS model performs better than the Remove model for MST.02-15.Wall with a 5% segment (table 1: 61.4% vs. 56.9%), but actually has a higher score threshold (table 2: 59.42 vs. 49.51). The difference here is in how closely tied the representation is to the data: for No-POS, there is a slight bit of abstraction for an element which is already present in the parse, whereas the Remove model completely changes the elements in the rule. The simplicity of the DAPS method likely means that it is risky to deviate too far from the original parses without utilizing further information.

4) Comparing the High and Wall methods confirms a finding in [18], namely: Wall methods tend to work better than High when less training data is involved, but then the two methods are more equivalent for greater amounts of training data. This underscores the main thrust of our analysis here: when there is less training data, it helps to provide additional representations, whether in the form of bigrams or in abstracted *n*-gram representations.

This comparison also raises the issue of how best to use the abstract representations: for bigrams, we used a weight of 0.01 and found good results. While precision is not better for higher thresholds, it is at least not generally worse than the High method. It seems, then, that it might be fruitful to explore weighting the abstract representation; indeed, it may help for all *n*-grams to explore a more proper weighting system, e.g., experimentally determining the optimal weights.

5 Conclusion

Building from the simple DAPS method for dependency parse error detection, we developed representations which account for parse errors occurring in the context of other parse errors. The core insight is to combine both very detailed representations with more relaxed representations, i.e., ones which allow for errors. We have shown that for situations with little annotated data to begin with, error detection precision can be greatly increased. How one evaluates crucially affects the conclusions to be drawn, underscoring the point in [18] that many different corpus settings need to be employed.

Within the DAPS method, there are several avenues to explore: even richer representations, e.g., incorporating lexical information, grandparent information, etc.; different methods of comparing and combining information (e.g., tree kernels); schemes for weighting information; and so forth. Additionally, by accounting for incorrect context, we hope to not only develop better error detection models, but also to get one step closer to developing a parse corrector, which will need to account for multiple, interrelated errors.

Acknowledgements

We would like to thank the participants of the IU computational linguistics colloquium, as well as the three anonymous reviewers, for useful feedback.

References

[1] Rahul Agarwal, Bharat Ambati, and Dipti Misra Sharma. A hybrid approach to error detection in a treebank and its impact on manual validation time. *Linguistic Issues in Language Technology (LiLT)*, 7(20):1–12, 2012.

[2] Bharat Ram Ambati, Mridul Gupta, Samar Husain, and Dipti Misra Sharma. A high recall error identification tool for hindi treebank validation. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, 2010.

[3] Serhiy Bykh and Detmar Meurers. Native language identification using recurring *n*-grams – investigating abstraction and domain dependence. In *Proceedings of COLING 2012*, pages 425–440, Mumbai, India, December 2012.

[4] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC 2006*, 2006.

[5] Markus Dickinson. Ad hoc treebank structures. In *Proceedings of ACL-08*, Columbus, OH, 2008.

[6] Markus Dickinson. Detection of annotation errors in corpora. *Language and Linguistics Compass*, forthcoming.

[7] Markus Dickinson and Charles Jochim. A simple method for tagset comparison. In *Proceedings of the 6th Language Resources and Evaluation Conference (LREC 2008)*, Marrakech, Morocco, 2008.

[8] Markus Dickinson and Amber Smith. Detecting dependency parse errors with minimal resources. In *Proceedings of IWPT-11*, pages 241–252, Dublin, October 2011.

[9] Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. Descriptive and empirical approaches to capturing underlying dependencies among parsing errors. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1162–1171, Singapore, August 2009. Association for Computational Linguistics.

[10] Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. Effective analysis of causes and inter-dependencies of parsing errors. In *Proceedings of IWPT-09*, pages 180–191, Paris, October 2009.

[11] Mohammad Khan, Markus Dickinson, and Sandra Kübler. Does size matter? text and grammar revision for parsing social media data. In *Proceedings of the Workshop on Language Analysis in Social Media*, Atlanta, GA USA, 2013.

[12] M. Marcus, Beatrice Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[13] Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of CoNLL-X*, pages 216–220, New York City, June 2006.

[14] Seyed Abolghasem Mirroshandel and Alexis Nasr. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149, Dublin, Ireland, October 2011. Association for Computational Linguistics.

[15] Alessandra Moschitti. Making tree kernels practical for natural language learning. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pages 113–120, Trento, Italy, 2006.

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

[17] Adam Przepiórkowski. What to acquire from corpora in automatic valence acquisition. In Violetta Koseska-Toszewa and Roman Roszko, editors, *Semantyka a konfrontacja jezykowa, tom 3*, pages 25–41. Slawistyczny Ośrodek Wydawniczy PAN, Warsaw, 2006.

[18] Amber Smith and Markus Dickinson. Evaluating parse error detection across varied conditions. In *Proceedings of the 13th International Workshop on Treebanks and Linguistic Theories (TLT13)*, Tübingen, Germany, 2014.

Querying topological fields in the TIGER scheme with TIGERSearch

Stefanie Dipper

Institute of Linguistics
Ruhr-University Bochum
E-mail: dipper@linguistics.rub.de

Abstract

The TIGER corpus is a German treebank with a hybrid dependency-constituency annotation. In this paper, I address the question how well topological fields (e.g. Vorfeld, Verb second) can be searched in this treebank, using the search tool TIGERSearch. For most queries, a version without crossing branches is used. It turns out that queries can be formulated that result in quite good F-scores for the Vorfeld and left and right brackets. Mittelfeld and Nachfeld are hard to query. This is due partly to properties of the language, partly to design decisions in the TIGER scheme, and partly to restrictions imposed by the search tool.

1 Introduction

This paper deals with the TIGER scheme [1], one of the two main annotation schemes for German syntax. The other main scheme is the TüBa-D/Z scheme [18]. The two schemes implement quite different design principles. The TIGER scheme is famous for its extensive use of *crossing branches* for encoding non-local dependencies. The TüBa-D/Z scheme is special in that it puts a layer with *topological fields* on top of the constituency structure.[1]

Topological fields are widely acknowledged as a useful concept by modern syntactic theories of German. Hence, linguists using German treebanks often would like to refer to these notions in formulating a query expression. The TIGER corpus [5] was created to serve both as training data for automatic applications and as a source for linguistic investigations. The question, addressed in this paper, is then whether linguist users are able to query topological fields not only in the TüBa-D/Z treebank, where they are explicitly encoded, but also in the TIGER treebank. The TIGER corpus comes with its own search tool, TIGERSearch [10], which is also used in this paper for searching the corpus.

[1]For a comparison of the two schemes, see [6].

VF	**LK**	**MF**			**RK**	**NF**		
Hans	*hat*	*heute*	*Maria*		*getroffen*	*die*	*einkaufen*	*war*
H.	has	today	M.		met	who	shopping	was
Hans	*traf*	*heute*	*Maria*			*die*	*einkaufen*	*war*
H.	met	today	M.			who	shopping	was
	dass	*Hans*	*heute*	*Maria*	*getroffen hat*	*die*	*einkaufen*	*war*
	that	H.	today	M.	met has	who	shopping	was

Figure 1: Topological field analysis of different sentences (‘(that) Hans met Maria today, who was shopping’)

The paper first gives a short introduction to German syntax (Sec. 2). Sec. 3 introduces the TIGER annotation scheme, and Sec. 4 presents the evaluation, followed by the conclusion (Sec. 5). The appendix contains sample templates.

2 German syntax: topological fields

German has a relatively free constituent order. Following a long tradition, German sentences are usually analyzed and split into different *topological fields* [9]. The element that functions as the separator between these fields is the verb or verbal parts (in most cases). The verb can be located in two different positions, either the second (“verb second”) or the final position (“verb final”) of the clause. Fig. 1 shows three sentences with their field analyses. “LK” and “RK” (“Linke/Rechte Klammer”, ‘left/right bracket’) indicate the two verbal positions (LK can also be occupied by subordinating conjunctions). The brackets divide the sentences into “VF” (“Vorfeld”, ‘prefield’), containing exactly one constituent, “MF” (“Mittelfeld”, ‘middle field’) with multiple constituents, and “NF” (“Nachfeld”, ‘postfield’), which often contains clausal constituents (which can be assigned a separate layer with topological fields). The brackets and the fields can also stay empty.

If the sentence contains only a simple verb form, one of the brackets remains empty, possibly resulting in ambiguous structures, see Fig. 2: (ia/b) and (iia/b) contain identical strings each, which can be analyzed by different brackets and fields, though. To (manually) disambiguate such structures, the simple verb form is replaced by some complex verb form, e.g. a particle verb or a combination of an auxiliary or modal plus verb. In (i’) the simple verb *ging* ‘went’ has been replaced by the particle verb *ging weg/wegging* ‘went away’; in (ii’) the simple form of the preterite *traf* ‘met’ has been replaced by perfect tense *hat getroffen* ‘has met’. The test paraphrases in (i’) reveal that (i) can be a verb-second (a) or verb-final (b) clause. The two options in (ii’) are stylistic variants, and it is sometimes hard to tell which is “the right” one. The TüBa-D/Z scheme defines a default rule for such cases [18, p. 93]: *Unless there is strong evidence for a position in MF, the relative clause is located in NF.* In the TIGER scheme, which does not annotate topological fields, the variants result in the same analysis.

The different topological slots — fields and brackets — are highly relevant

	VF	LK	MF	RK	NF
(i) a	***wer***	***ging***			
(i) b	***wer***			***ging***	
	who	went		went	
(i') a	*wer*	*ging*			*weg*
	who	went			away
(i') b	*wer*			*wegging*	
	who			away-went	
(ii) a	*Hans*	*traf*	***Leute,*** ***die*** ...		
(ii) b	*Hans*	*traf*	***Leute,***		***die*** ...
	H.	met	people who		who
(ii') a	*Hans*	*hat*	*Leute, die* ...	*getroffen*	
(ii') b	*Hans*	*hat*	*Leute*	*getroffen,*	*die* ...
	H.	has	people who	met	who

Figure 2: (Fragments of) syntactically-ambiguous (i/ii) and non-ambiguous (i'/ii') sentences ('who went (away)?'; 'Hans met people who ... ')

for research in German syntax. E.g. the Vorfeld often serves as a test position for constituency because it usually contains exactly one constituent — there are exceptions, though (see e.g. [11]). The Vorfeld is also interesting from an information-structural point of view because it seems to be the prime position for sentence topics — it often contains constituents with other information-structural functions, though (e.g. [16, 7]). Constituent order ("scrambling") within the Mittelfeld has been investigated extensively (e.g. [2]), as well as the question which constituents can occur *extraposed*, i.e. in the Nachfeld slot (e.g. [17]). Finally, the relative order of verbal elements in the Rechte Klammer has been researched a lot (e.g. [8]).

3 The TIGER annotation scheme

The TIGER scheme implements a hybrid approach to syntactic structure, combining features from constituency and dependency structures. On the one heand, it uses virtual nodes like "NP" and "VP" for constituents. On the other hand, non-local dependents are connected by crossing branches, directly linking the head and its dependent; edges are labeled by grammatical functions such as "SB" (subject) or "MO" (modifier).

The TIGER scheme omits "redundant" nodes, assuming that these nodes can be recovered automatically by combining information from the POS tags and/or the functional labels. This concerns two types of nodes: *unary* nodes, i.e. non-branching nodes like NP nodes that dominate one terminal node only; and NP nodes dominated by PPs.

This design principle — omitting redundant nodes — poses obvious problems for treebank users. If users are interested e.g. in VPs with an NP daughter, they have

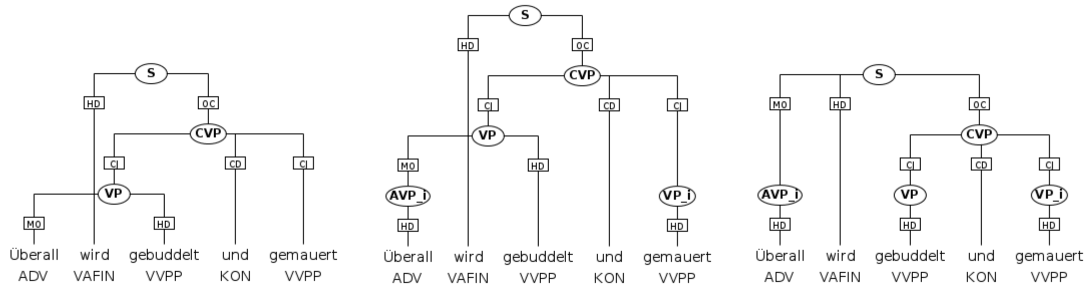

Figure 3: TIGER sentence no. 291 in original version (left), enriched version (“ENR”, center), and endriched context-free version (“CF”, right) (‘Everywhere people are digging and constructing’)

to make additional efforts to retrieve all actual NPs. The search tool that comes with the TIGER corpus, TIGERSearch [10], allows the user to define *templates*, which facilitates such queries enormously. (1a) defines a sample template for pronouns. The first conjunct exploits the fact that all pronominal POS tags start with “P” (e.g. “PPER” for personal pronouns, “PPOSAT” for attributive possessive pronouns, etc. [15]). Other tags that start with “P” (pronominal and interrogative adverbs and particles) are excluded by the second conjunct. (1b) shows how to use the template in a query to constrain the otherwise unspecified node variable “#a” to pronouns. The query searches for VPs that directly dominate some pronoun.

(1) a. `PRON(#x) <- #x: [pos=/P.*/ & pos!=/PROAV|PWAV|PTK.*/];`

b. `[cat="VP"] > #a:[] & PRON(#a)`

In a similary way, a template for NPs in general could be defined. An alternative way is to apply a script that expands TIGER’s minimalistic structures and inserts such redundant nodes, thus creating an enriched, user-friendly version of the treebank, as has been suggested e.g. by [14]. Fig. 3 illustrates both formats. The figure shows a TIGER structure in the original version (left) and in the enriched version (center), with two inserted nodes: AVP_i and VP_i.[2]

Non-local dependencies are encoded by crossing branches in the TIGER scheme. Such structures are difficult to process automatically, so scripts have been created to re-attach these branches in a way to avoid crossings. I call the resulting structures “context-free” because they could have been created by a context-free grammar. The rightmost structure shown in Fig. 3 is such a context-free structure.[3] It attaches the AVP_i node higher up, eliminating the crossing branch. The evaluation

[2]The enriched version of the corpus has been created by the tool *TIGER Tree Enricher* [13]. The marker “_i” for inserted nodes is optional, and is used here to highlight inserted nodes. All sentence numbers in this paper refer to the TIGER corpus, release 2.2; URL: `http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html`.

[3]The context-free version of the corpus has been created by the program *treetools* by Wolfgang Maier, URL: `https://github.com/wmaier/treetools`.

Field	Template description
VF	– In main clauses: leftmost constituent of a sentence, preceding a finite verb; coordinating conjunctions may precede – In subord. clauses: leftmost constituent of a sentence, dominating a relative or interrogative word
LK	– In main clauses: the finite verb following VF – In subord. clauses: the subordinating conjunction
MF	The part between LK and RK (i.e. both brackets must be filled); the template marks the beginning (MFB) and end (MFE) of MF
RK	– Single-element RKs (RKS): the finite verb in a subordinate clause, or a verb particle, infinitive or participle – Multi-element RKs: a cluster of several verbs; the template marks the beginning (RKB) and end (RKE) of complex RKs
NF	The part following a verb particle, infinitive, participle or verb cluster; the template only marks the beginning (NFB) of NF

Table 1: Description of the topological templates

will show that querying the topological fields is rather difficult if not impossible if crossing branches may occur.[4]

4 Querying the scheme: an evaluation

The TIGER corpus has been used successfully to search for elements in specific topological fields. For instance, [12] investigates (a subset of) extraposed clauses, i.e. clauses in the Nachfeld. However, [12] only considers clauses that are dependent from a noun (object or relative clauses), which facilitates querying enormously. As we see below, querying for Nachfeld constituents *in general* is actually very hard.

For the evalution, a student of linguistics annotated the first 600 sentences of the TIGER corpus with topological fields.[5] Sentences 1–100 were used in the development of the query templates, sentences 101–600 were reserved for the evaluation.

Of course, the results of the evaluation heavily depend on the quality of the templates. Table 1 summarizes the most relevant properties of the individual templates. In the definitions of the templates, I tried to avoid exploiting linguistic knowledge about the topological fields, such as "sentential constituents often are located in the NF" (because this is a statement that one might want to verify). However, I did use information such as "relative clauses are verb-final clauses".

I evaluate the scheme by applying the templates to the TIGER corpus in an

[4]This observation can be transferred to treebanks annotated with pure dependency structures.

[5]The annotation tool was WebAnno [19]. The fields were annotated mainly according to the TüBa-D/Z scheme. However, the student located interrogative and relative pronouns in VF, and subordinating conjunctions in LK (rather than C).

enriched version with redundant nodes and in context-free format ("CF"). Querying the version with crossing branches ("ENR") results in highly complex (and inefficient) queries, so I defined only the template for VF in ENR.[6]

The appendix displays the template definitions for the Vorfeld position and the precedence relation in the CF version.[7,8] For efficiency reasons, the VF template is split in two parts: VF in main (VFmain) and subordinate (VFsub) clauses. VFmain also covers the verb-second (V2) position in the LK slot, since VF and V2 depend on each other. A query using the VF template is shown in (2).

(2) `#vf:[] & #v2:[] & VFmain_cf(#vf,#v2)`

The templates are designed to result in high precision rather than high recall. For instance, only VF instances are covered where the sentence either directly starts (i) with the VF or (ii) with a coordinating conjunction that directly precedes the VF. Other sentence-initial elements or elements following the VF are not allowed to maintain the constraint that there is exactly one constituent preceding the finite verb. This constraint, e.g., excludes VF in sentences with preposed material (3a) or with parentheticals intervening between VF and the finite verb (3b).[9]

(3) a. [$_{AVP}$ Gewiß] — [$_{NP}$ die wirtschaftliche Liberalisierung und Öffnung des Landes$_{VF}$] schreiten voran . (s62)

b. [$_{CAP}$ Früher oder später$_{VF}$] , [$_{S}$ da sind sich alle einig] , muß Perot Farbe bekennen und Konzepte vorlegen . (s47)

4.1 Qualitative results

Qualitative results from the development process show that there are certain types of constructions that cannot be handled properly by the templates. The problems can be traced back to (i) difficult constructions, (ii) systematic ambiguities of the language, (iii) constraints of the search tool, and (iv) the design of the annotation scheme, in particular (v) crossing branches.

[6]Using the enriched versions facilitates querying since we do not have to care about omitted NP nodes etc. The vast majority of the conversion steps of the enrich-script are trivial so they do not affect the evaluation, cf. [13]. Creating the context-free version involves more complex operations, see Fn. 4. Still, the conversion does not seem to introduce problematic structures.

[7]All template definitions used in this paper can be found at `http://www.linguistics.ruhr-uni-bochum.de/~dipper/tiger-templates.html`.

[8]TIGERSearch uses a purely left corner-based definition of precedence, which is not sufficient in most cases (a node #n1 is said to precede another node #n2 if the left corner of #n1 precedes the left corner of #n2 [10, p. 80]; according to this definition, a node consisting of two or more words does not precede its following sibling). In addition, the precedence template allows for intervening quotes (via the template "prec_quotes"; and similarly with "prec_comma"). The VF template further refers to a template "hasLeftChild", which defines left-corner dominance. This template extends the corresponding TIGER relation to one that holds between terminal or non-terminal nodes.

[9]The parenthetical sentence in (3b) contains a VF, which is correctly found by the VF template. Here and in the following examples, the underlined, labeled part indicates the "target" slot, as annotated in the gold data, and the part in boldface indicates the string matched by the template (if any).