Constraining robust constructions for broad-coverage parsing with precision grammars

Bart Cramer[†] **and Yi Zhang**[†‡]
Department of Computational Linguistics & Phonetics, Saarland University[†]
LT-Lab, German Research Center for Artificial Intelligence (DFKI)[‡]
{bcramer,yzhang}@coli.uni-saarland.de

Abstract

This paper addresses two problems that commonly arise in parsing with precision-oriented, rule-based models of grammar: lack of speed and lack of robustness. First, we show how we can reduce parsing times by restricting the number of tasks the parser will carry out, based on a generative model of rule applications. Second, we show that a combination of search space restriction and radically overgenerating robustness rules lead to a more robust parser, with only a small penalty in precision. Applying both the robustness rules and a fragment fallback strategy showed better recall than just giving fragment analyses, with equal precision. Results are reported on a medium-sized HPSG grammar for German. [1]

1 Introduction

In the field of natural language processing, it is common wisdom that handwritten, rule-based models generally perform poorly on complex problems, mainly due to the knowledge acquisition bottleneck: it is hard for the human modeller to conceive of all possible scenarios the model has to cope with. In parsing, many approaches have relied on hand-written grammars, and their fragility is one of their largest weaknesses. Such models can fail due to insufficiency of lexical entries or grammatical constructions, but also due to creative or ungrammatical input. In any case, the parser should always return a reasonable output. A very simple technique is partial or fragment parsing (Kiefer et al., 1999; Riezler et al., 2001; Zhang et al., 2007a): if there is no item in the chart that both spans the complete sentence and fulfills the root condition, several chunks that do conform to a root condition are combined by minimising a certain cost function (for instance to favour larger chunks, or more probable chunks).

A second problem with deep parsers is their relatively low efficiency. For online applications, it is impermissible to wait for longer than a minute before the system responds. Apart from studies that were aimed at increasing the efficiency of deep parsers by using smarter algorithms (e.g. using left-corner relations (Van Noord, 1997)), several studies in recent years have suggested that search space restriction can offer a beneficial balance between speed and accuracy as well. Techniques that have been proposed are, among others, supertagging (Clark and Curran, 2007), CFG filtering (Matsuzaki et al., 2007) and beam thresholding (Ninomiya et al., 2005).

A potential disadvantage of the latter technique is that the unifications have taken place by the time the value of the resulting chart item is investigated. One strategy that tries to prevent execution of unlikely tasks altogether is presented by van Noord (2009). In this method, the parser learns from an unannotated corpus which parse steps contributed to the solution as preferred by the disambiguation model (as opposed to a certain gold standard). Hence, this approach is self-learning.

Another study that is close to our approach

[1]The research reported on in this paper has been carried out with financial support from the Deutsche Forschungsgemeinschaft and the German Excellence Cluster of Multimodal Computing & Interaction.

Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 223–231,
Beijing, August 2010

to search space restriction is c-structure pruning (Cahill et al., 2008). The authors show that a large, hand-written, unification-based parser (the XLE LFG parser for English) can perform reasonably faster (18%) without losing accuracy, by not allowing the parser to unify if the resulting item will have a span that does not conform to a CFG tree that was generated from the sentence beforehand by a PCFG parser. Much better results (67% speed-up) are obtained by pruning chart items locally, based on their relative probabilities (Cahill et al., 2008). This is the approach that is closest to the one we present in this paper.

In this paper, we introduce a method that addresses robustness and efficiency concurrently. The search space is restricted by setting a maximum on the number of tasks per chart cell. Because tasks are carried out according to a priority model based on the generative probabilities of the rule applications, it is unlikely that good readings are dropped. More robustness is achieved by adding radically overgenerating rules to the grammar, which could cover all sentences, given an disproportionate amount of time and memory. By strongly restricting the search space, however, the computation requirements remains within bounds. Because the robustness rules are strongly dispreferred by both the priority model and the disambiguation model, all sentences that would be covered by the 'restricted' grammar remain high-precision, but sentences that are not covered will get an additional push from the robustness rules.

1.1 An HPSG grammar for German

The grammar we use (Cramer and Zhang, 2009) is the combination of a hand-written, constraint-based grammar in the framework of HPSG and an open word class lexicon extracted from the Tiger treebank (Brants et al., 2002) in a deep lexical acquisition step. One of the aims of this grammar is to be precision-oriented: it tries to give detailed analyses of the German language, and reject ungrammatical sentences as much as possible. However, this precision comes at the cost of lower coverage, as we will see later in this paper.

Along with the grammar, a treebank has been developed by re-parsing the Tiger treebank, and including those sentences for which the grammar was able to reproduce the original Tiger dependencies. The treebank's size is just over 25k sentences (only selected from the first 45k sentences, so they don't overlap with either the development or test set), and contains the correct HPSG derivation trees. These (projective) derivation trees will function as the training set for the statistical models we develop in this study.

2 Restriction of the search space

2.1 The PET parser

The parser we employ, the PET parser (Callmeier, 2000), is an agenda-driven, bottom-up, unification-based parser. In order to reduce computational demands, state-of-the-art techniques such as subsumption-based packing (Oepen and Carroll, 2000) and the quasi-destructive unification operator (Tomabechi, 1991) have been implemented.

A central component in the parser is the agenda, implemented as a priority queue of parsing tasks (unifications). Tasks are popped from the agenda, until no task is left, after which all passive items spanning the complete sentence are compared with the root conditions as specified by the grammar writer. The best parse is extracted from the parse forest by a Maximum Entropy parse disambiguation model (Toutanova et al., 2002), using selective unpacking (Zhang et al., 2007b).

Two different types of items are identified: passive items and active items. Passive items are 'normal' chart items, in the sense that they can freely combine with other items. Active items still need to combine with a passive item to be complete. Hence, the parser knows two types of tasks as well (see figure 1): *rule+passive* and *active+passive*.

Each time a task succeeds, the following happens:

- For each inserted passive item, add (rule+passive) tasks that combine the passive item with each of the rules, and add (active+passive) tasks that combine with each of the neighbouring active items.
- For each inserted active item, add (active+passive) tasks that combine the remain-

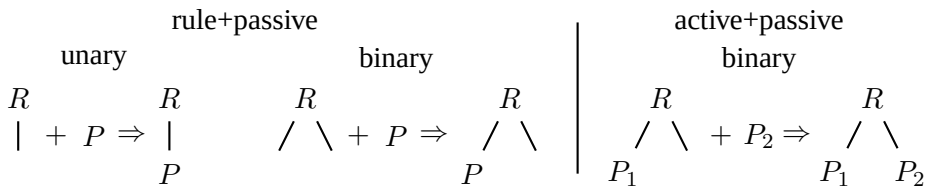


Figure 1: Depicted are the different types of tasks in the PET parser. Not shown are the features structures imposed by the rules and the chart items.

ing gaps in the active item with existing neighbouring passive items in the chart.

2.2 Defining priorities

The priorities of the parsing tasks are calculated based on a generative PCFG model extracted from the treebank by maximum likelihood estimation, smoothed by Lidstone smoothing. Each passive chart item receives a score based on its generative probability, calculated as the product of all applied rule probabilities. For active parsing items, we set the score to be the upper bound of this generative probability, if the item succeeds later in combining with other passive edge(s) to build a complete subtree. This is done by simply assuming the undetermined subtree in the active item receiving a generative score of 1.

The priorities that are assigned to both types of tasks are not yet conditioned on the probability of the topmost rule application. Hence, they are computed using the following simple formula:

$$Pr = p(R) \cdot p(P)$$

where Pr is the task's priority, $p(R)$ the prior probability of the rule category R; and $p(P)$ is the highest possible generative probability of the *resulting* passive item P.

2.3 Restriction strategies

It is a natural thought to allocate more computational resources to longer sentences, and this is exactly what happens in the restriction strategies we develop in this study. We define a cap on the number of tasks for a certain cell/span (i, j), which means that the number of cells is quadratically related to the number of words in a sentence: $n_{cells} = n(n+1)/2$.

We define three task restriction strategies: *all*, *success*, and *passive*. In *all*, the cap is defined for all tasks, whether the unification is successful or not. *Success* only counts tasks that are successful (i.e. lead to either an active or a passive item), and *passive* only counts tasks that lead to a passive item. In all strategies, morphological and lexical tasks are not counted, and hence not restricted. Unary phrasal rules (such as *empty-det*) are counted, though.

The implementation uses only one priority queue. Each time a task is popped from the agenda, it is checked whether the limit for this span has been reached or not. If so, the task is discarded; otherwise, it is executed.

2.4 Methodology

All our experiments are based on the Tiger treebank (Brants et al., 2002). The grammar's lexicon is based on the first 45k sentences in the treebank, and so are the MaxEnt disambiguation model (Toutanova et al., 2002) and the generative model we developed for this study. The development set (s45001-s47500) was used to fine-tune the methods, but all final results presented in this paper are with respect to the test set (s47501-s50000). The maximum time for building up the packed parse forest is 60 seconds, after which unpacking is started. Unpacking the first reading usually has negligible computation costs, and is not reported on. Along with the best reading's derivation, the dependencies are output, and com-

Strategy	exhaustive	all	success	passive
Cap size		3000	200	100
Time (s)	7.20	1.04	0.92	1.06
Coverage	59.4%	60.5%	60.0%	59.0%
Exact	17.6%	17.6%	17.4%	17.4%
Recall	37.6%	39.5%	38.9%	38.0%
Precision	80.7%	80.3%	80.1%	80.4%
F-score	51.3%	52.9%	52.4%	51.6%

Table 1: A more detailed look into some data points from figure 2. 'Coverage' and 'Exact' are sentential percentages, showing how many sentences receive at least one or the exactly correct reading. Recall, precision and f-score are on a per-dependency basis.

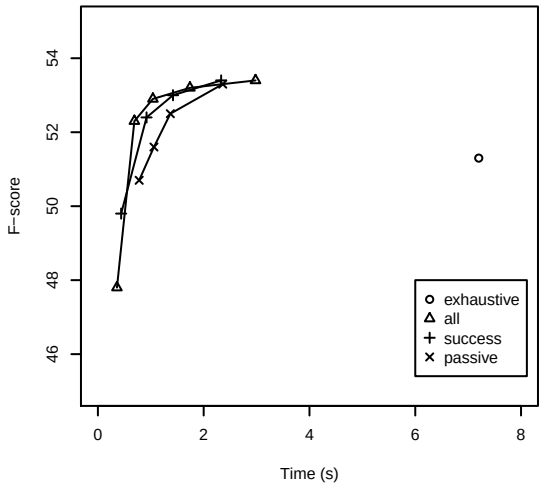


Figure 2: This figure shows the tradeoff between speed and f-score for the standard grammar, using the restriction strategies with different cap sizes.

pared to the gold standard dependencies from the Tiger treebank.

2.5 Results

The results of the experiments, with different cap sizes, are summarized in table 1 and figure 2. As expected, for all strategies it holds that longer computation times lead to higher coverage numbers. The interesting thing is that the restriction of the search space doesn't affect the parses' precision, indicating that the priorities work well: the tasks leading to good solutions are indeed given high priority scores.

A striking observation is that the coverage numbers go up by about 1%, with reductions in parse times of more than 80%. This is due to the use of the timeout, and the generic tendency of our definition of the priorities: because less rule applications lead to higher log probabilities, the agenda will favour tasks with smaller span size. If the agenda doesn't apply too strong a restriction on those tasks, the parser might not create any items spanning the whole sentence after the full 60 seconds, and hence produce no parse. This is mitigated by stronger restriction, leading to a quicker path upwards in the chart.

No large differences of success are found between the different strategies. The intuition behind the *success* and *passive* strategies was that only more effort should be invested into a particular span if not enough chart items for that span have been created. However, the time/quality trade-offs are very similar for all strategies, as shown in figure 2[2].

The strategies we have reported on have one thing in common: their counters are with respect to one particular span, and therefore, they have a very local scope. We have tried other strategies that would give the algorithm more flexibility by defining the caps on more global scale, for instance per span length or for the entire chart. However, this degraded the performance severely, because the parser was not able to divide its attention properly.

[2]One might be tempted to consider the *all* strategy as the best one. However, the time/f-score tradeoff curves look slightly different on the development set.

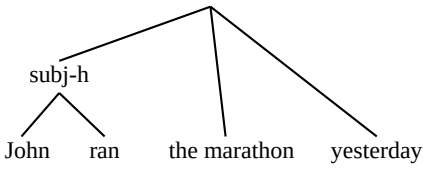
3 Increasing robustness

For hand-written deep parsers, efficiency and coverage are often competing factors: allowing more items to be created might be beneficial for recall, but the parser will also be too slow. However, because the search space can be restricted so rigidly, we can make the grammar more permissive to accept more sentences, hopefully without a heavy efficiency penalty. One way to do this is to remove constraints from the grammar rules. However, that would infringe on the precision-oriented nature of the grammar. Instead, we will keep the normal grammar rules as they are, and create a small number of additional, super-accepting *robustness rules*. The intuition is that when the restricted part of the grammar can find a solution, that solution will indeed be found, and preferred by the statistical models. On the other hand, when the sentence is extragrammatical, the robustness rules may be able to overcome the barriers.

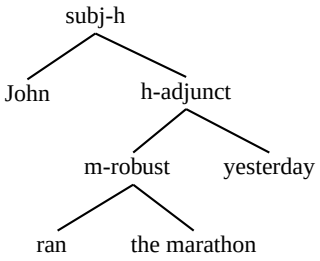
Let's consider the following example, assuming that the grammar only lists 'to run' as an intransitive verb:

'John ran the marathon yesterday'

A fragment approach would come up with the following solution:



'John' will correctly be identified as the subject of 'ran', but that is all. No dependencies are established between 'the marathon' and 'ran', or 'yesterday' and 'ran'. The former is hard to establish, because of the missing lexical item. However, the latter should be doable: the lexicon knows that 'yesterday' is an adverb that modifies verbs. If we could create a robustness rule that would absorb the object ('the marathon') without assigning a dependency, it would at least be able to identify the modifier dependency between 'ran' and 'yesterday'.



In other words, a fragment analysis solely combines items at the top level, whereas a robust parser would ideally be able to overcome barriers in both the lower and the higher regions of the chart, meaning that the damage can be localised and thus minimised. The robustness rules we propose are intended to achieve that.

How does this idea interact with the restriction mechanism explained in the previous section? Robustness rules get an inhibitively large, constant penalty in both the priority model and the disambiguation model. That means that at first the parser will try to build the parse forest with the restricted set of rules, because tasks involving subtrees with only rules from the standard grammar will always have a higher priority than tasks using an item with a robustness rule application in its subtree. When this is finished, the robustness rules try to fill the gaps. Especially in the *success* and *passive* strategies, tasks with robustness rules are discarded if already enough chart items are found for a particular span, meaning that the parser automatically focusses on those parts of the chart that haven't been filled before.

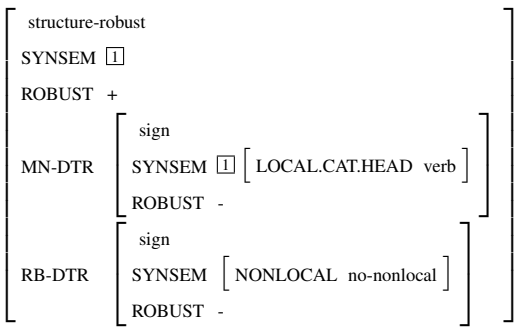
3.1 Defining robustness rules

Defining robustness rules is a sort of grammar engineering, and it took a bit of experimentation to find rules that worked well. One of the factors was the interaction between the subsumption-based packing and the robustness rules. When the chart is built up, items that are subsumed by an existing item are marked as 'frozen', and the latter (more general) item functions as the representative node in the remainder of the parsing process. When unpacking the best solution, the best derivation tree is extracted from the packed forest, which

might include a frozen node. Because this frozen node has more constraints than its representative, this derivation tree is not guaranteed to be free of unification failures, and hence, before outputting, this is checked by replaying all the unifications in the derivation tree. This procedure is repeated until a sound derivation has been found.

So what happens when the representative nodes are very general? Many nodes will be packed, and hence the chart will remain compact. However, the unpacking process will become problematic, because many of the proposed derivation trees during unpacking will be incorrect, leading to excessive computation times (in the order of minutes).

Therefore, we chose to define robustness rules such, that the resulting chart items will be equally constrained as their daughters. They are all binary, and have one common ancestor in the type hierarchy:



All rules have a *main* daughter and a *robust* daughter. The co-indexation of the SYNSEM of the main daughter and the SYNSEM of the rule itself has the effect that the resulting chart item will have the exact same syntactic properties as its main daughter, whereas the robust daughter does not contribute to the syntactic properties of the mother node. The ROBUST feature is used to prevent the application of two robust rules consecutively. Additional constraints (not shown) make sure that morphological processing is finished, and that both parts are not involved in a coordination. Robustness rules do not yield a dependency triple (although they mght be guessed accurately by a few heuristics).

We define two pairs of robustness rules, each pair consisting of a rule with MN-DTR first and RB-DTR second, and one rule in the other order:

+V The robust daughter is a verb, which is still allowed to have valence, but cannot have any features in NONLOCAL.

+NV The robust daughter is anything but a verb, cannot have any non-empty valence list, and cannot have any features in NONLOCAL.

3.2 Fragment parsing

As a baseline for comparison, we investigate the existing partial parsing algorithms that pick fragmented analyses from the parse forest as a fallback strategy when there is no full parse available. Kiefer et al. (1999) took a shortest-path approach to find a sequence of fragment analysis that minimizes a heuristics-based cost function. Another variation of the algorithm (Riezler et al., 2001) is to pick fewest chunks that connect the entire sentence. While these early approaches are based on simple heuristics, more sophisticated parse selection methods also use the statistical models to rank the partial analyses. For example, Zhang et al. (2007a) proposed several ways of integrating discriminative parse ranking scores with the partial parse selection algorithm.

In this experiment, we first use the shortest path algorithm to find candidate chunks of partial analysis. All phrasal constituents were given equal weights, and preferred over input and lexical edges. For each chunk (edges spanning the same sub-string of the input sentence), the edge with the highest generative probability is picked. Consequently, the best partial reading (covering that edge) is decoded by the selective unpacking algorithm using the MaxEnt parse ranking model. With each fragment, the partial semantic representations were extracted. Similar to the robustness rules, no cross-fragment dependencies are recovered in this approach. Due to the limited number of chart items and the use of selective unpacking, the computation times for the shortest-path algorithm are marginal.

3.3 Results

The results of this experiment are listed in table 2. For the robust versions of the grammar, no exhaustive parsing results are reported, because they take too long to compute, as can be expected. Coverage number are on a per-sentence

		standard		+V	+NV	+V+NV
		exhaustive	restricted	restricted		
	time (s)	7.20	0.92	4.10	1.42	4.09
no fragment	coverage	59.3%	60.0%	72.6%	69.9%	78.6%
	recall	37.6%	38.9%	48.4%	47.0%	53.8%
	precision	80.7%	80.1%	78.6%	78.2%	77.7%
	f-score	51.3%	52.4%	59.9%	58.7%	63.6%
fragment	coverage	94.3%	98.3%	98.5%	98.7%	98.5%
	recall	50.4%	53.6%	59.5%	56.9%	61.3%
	precision	75.4%	75.0%	75.0%	74.5%	74.7%
	f-score	60.4%	62.5%	66.3%	64.5%	67.3%

Table 2: Results for experiments with different robustness rules, and with or without fragment fallback strategy.

basis, whereas the other percentages are on a per-dependency basis. Time denotes the average number of seconds it takes to build the parse forest. All results under 'restricted' are carried out with the *success* strategy, with a cap of 200 tasks (*success-200*). '(No) fragment' indicates whether a fragment parse is returned when no results are obtained after selective unpacking.

The robustness rules significantly increase the sentential coverage, in the case of +V+NV almost 20 percent points. The gains of +V and +NV are fairly additive: they seem to cover different sets of extragrammatical sentences. In the most permissive setting (+V+NV), dependency recall goes up by 16 percent point, with only a 3 percent point decrease of precision, showing that the newly-covered sentences still receive fairly accurate parses. Also, it can be seen that the +V pair of rules is more effective than +NV to increase coverage. The robust grammars are certainly slower than the standard grammar, but still twice as fast as the standard grammar in an exhaustive setting.

Coverage numbers are approximating 100% when the fragment parsing fallback strategy is applied, in all settings. However, it is interesting to see that the recall numbers are higher when the robustness rules are more permissive, but that no significant effect on the precision is observed. This suggests that the lumps that are connected by the fragment parsing mechanism are larger, due to previous applications of the robustness rules. From this, we conclude that the connections made by the robustness rules are of relatively high quality.

We have also tried the *all*-3000 and *passive*-100 settings (the same as listed in table 1). That yielded very similar results, except on the grammar with both +V and +NV enabled. With *passive*-100, there was a small decrease in coverage (76.0%), but this drop was much more pronounced for *all*-3000: 72.0%. This suggests that, if the pressure on the generative model is larger due to heavier overgeneration, counting successful tasks or passive items performs better than just counting the number of executed tasks.

After manual inspection, we found out that the kind of constructions the robustness rules created were very diverse. Most of the rule applications were not in the top of the tree, as was intended. There also seemed to be a correlation between the length of the robust daughter and the quality of the parse. When the robust daughter of the rule was large, the application of the robustness rule looked like an emergency strategy, with a corresponding quality of the parse. However, when the robustness rule connects a verb to a relatively small constituent (a particle or an NP, for example), the resulting derivation tree was of reasonable quality, keeping most of the other dependencies intact.

4 Discussion

Achieving broad coverage in deep parsing while maintaining high precision is difficult. Until now, most existing hand-written grammar-based parsing systems rely on fragment analyses (or various ways of putting fragments together to compose

partial readings), but we argued (with the example in section 3) that such an approach delivers inferior results when the tree falls apart at the very bottom. The use of robust constructions offers a way to keep the damage local, but can create an intractable search space. The proposed pruning strategies carefully control the bound of overgeneration, resulting in improvements on both parsing efficiency and coverage, with a significantly smaller degradation in f-score than a pure fragment approach. The combination of grammar engineering, statistical modelling and algorithmic design in the parser brings the parser performance to a new level.

Although the experiments were carried out on a specific grammar framework, we consider the techniques put forward in this paper to be applicable to other linguistic frameworks. The robustness rules are easy to construct (with the precautions from section 3.1 in mind), and all modern deep parsers have a treebank to their disposal, from which the generative model can be learned.

There are still points that can be improved on. Currently, there is no way to determine which of the *robust* rule applications are more promising than others, and the decision to try one before the other is solely based on the the probabilities of the passive items, and not on the generative model. This can be inefficient: for instance, all robustness rules presented in this paper (both +V and +NV) requires the main daughter to be a verb. It would be straightforward to learn from a small treebank that trying to unify the main daughter of a robustness rules (which should have a verbal head) with a specifier-head rule application does not have a high chance on succeeding.

Another possible improvement is to differentiate between different robustness rules. We presented a two-tier system here, but the framework lends itself naturally to more layers with differing degrees of specificity, creating a smoother scale from specific/prioritised to robust/non-prioritised.

References

Brants, S., S. Dipper, S. Hansen, W. Lezius, and G. Smith. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41.

Cahill, A., J.T. Maxwell III, P. Meurer, C. Rohrer, and V. Rosén. 2008. Speeding up LFG parsing using c-structure pruning. In *Proceedings of the Workshop on Grammar Engineering Across Frameworks*, pages 33–40. Association for Computational Linguistics.

Callmeier, U. 2000. PET–a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(01):99–107.

Clark, S. and J.R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Cramer, B. and Y. Zhang. 2009. Construction of a German HPSG grammar from a detailed treebank. In *Proceedings of the GEAF workshop ACL-IJCNLP 2009*, pages 37–45.

Kiefer, B., H.U. Krieger, J. Carroll, and R. Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 473–480. Association for Computational Linguistics.

Matsuzaki, T., Y. Miyao, and J. Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1671–1676, Hyderabad, India.

Ninomiya, T., Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 103–114. Association for Computational Linguistics.

Oepen, S. and J. Carroll. 2000. Ambiguity packing in constraint-based parsing: practical results. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 162–169. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Riezler, S., T.H. King, R.M. Kaplan, R. Crouch, J.T. Maxwell III, and M. Johnson. 2001. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 271–278.

Tomabechi, H. 1991. Quasi-destructive graph unification. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 315–322. Association for Computational Linguistics.

Toutanova, K., C.D. Manning, S. Shieber, D. Flickinger, and S. Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263.

Van Noord, G. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456.

van Noord, G. 2009. Learning efficient parsing. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 817–825, Athens, Greece, March. Association for Computational Linguistics.

Zhang, Y., V. Kordoni, and E. Fitzgerald. 2007a. Partial parse selection for robust deep processing. In *Proceedings of ACL 2007 Workshop on Deep Linguistic Processing*, pages 128–135, Prague, Czech.

Zhang, Y., S. Oepen, and J. Carroll. 2007b. Efficiency in Unification-Based N-Best Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 48–59. Association for Computational Linguistics.