Getting the Most out of Transition-based Dependency Parsing

Jinho D. Choi
Department of Computer Science
University of Colorado at Boulder
choijd@colorado.edu

Martha Palmer
Department of Linguistics
University of Colorado at Boulder
mpalmer@colorado.edu

Abstract

This paper suggests two ways of improving transition-based, non-projective dependency parsing. First, we add a transition to an existing non-projective parsing algorithm, so it can perform either projective or non-projective parsing as needed. Second, we present a bootstrapping technique that narrows down discrepancies between gold-standard and automatic parses used as features. The new addition to the algorithm shows a clear advantage in parsing speed. The bootstrapping technique gives a significant improvement to parsing accuracy, showing near state-of-the-art performance with respect to other parsing approaches evaluated on the same data set.

1 Introduction

Dependency parsing has recently gained considerable interest because it is simple and fast, yet provides useful information for many NLP tasks (Shen et al., 2008; Councill et al., 2010). There are two main dependency parsing approaches (Nivre and McDonald, 2008). One is a transition-based approach that greedily searches for local optima (highest scoring transitions) and uses parse history as features to predict the next transition (Nivre, 2003). The other is a graph-based approach that searches for a global optimum (highest scoring tree) from a complete graph in which vertices represent word tokens and edges (directed and weighted) represent dependency relations (McDonald et al., 2005).

Lately, the usefulness of the transition-based approach has drawn more attention because it generally performs noticeably faster than the graph-based approach (Cer et al., 2010). The transition-based approach has a worst-case parsing complexity of $O(n)$ for projective, and $O(n^2)$ for non-projective parsing (Nivre, 2008). The complexity is lower for projective parsing because it can deterministically drop certain tokens from the search space whereas that is not advisable for non-projective parsing. Despite this fact, it is possible to perform non-projective parsing in linear time in practice (Nivre, 2009). This is because the amount of non-projective dependencies is much smaller than the amount of projective dependencies, so a parser can perform projective parsing for most cases and perform non-projective parsing only when it is needed. One other advantage of the transition-based approach is that it can use parse history as features to make the next prediction. This parse information helps to improve parsing accuracy without hurting parsing complexity (Nivre, 2006). Most current transition-based approaches use gold-standard parses as features during training; however, this is not necessarily what parsers encounter during decoding. Thus, it is desirable to minimize the gap between gold-standard and automatic parses for the best results.

This paper improves the engineering of different aspects of transition-based, non-projective dependency parsing. To reduce the search space, we add a transition to an existing non-projective parsing algorithm. To narrow down the discrepancies between gold-standard and automatic parses, we present a bootstrapping technique. The new addition to the algorithm shows a clear advantage in parsing speed. The bootstrapping technique gives a significant improvement to parsing accuracy.

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers, pages 687–692,
Portland, Oregon, June 19-24, 2011.

LEFT-POP$_{\mathrm{L}}$	$(\ [\lambda_1 \mid i],\ \lambda_2,\ [j \mid \beta],\ E\ ) \Rightarrow (\ \lambda_1\ ,\ \lambda_2,\ [j \mid \beta],\ E \cup \{i \overset{\mathrm{L}}{\leftarrow} j\}\ )$ $\exists i \neq 0, j.\ i \not\rightarrow^* j \quad \wedge \quad \nexists k \in \beta.\ i \rightarrow k$
LEFT-ARC$_{\mathrm{L}}$	$(\ [\lambda_1 \mid i],\ \lambda_2\ ,\ [j \mid \beta],\ E\ ) \Rightarrow (\ \lambda_1\ ,\ [i \mid \lambda_2],\ [j \mid \beta],\ E \cup \{i \overset{\mathrm{L}}{\leftarrow} j\}\ )$ $\exists i \neq 0, j.\ i \not\rightarrow^* j$
RIGHT-ARC$_{\mathrm{L}}$	$(\ [\lambda_1 \mid i],\ \lambda_2\ ,\ [j \mid \beta],\ E\ ) \Rightarrow (\ \lambda_1\ ,\ [i \mid \lambda_2],\ [j \mid \beta],\ E \cup \{i \overset{\mathrm{L}}{\rightarrow} j\}\ )$ $\exists i, j.\ i \not\leftarrow^* j$
SHIFT	$(\ \lambda_1\ ,\ \lambda_2,\ [j \mid \beta],\ E\ ) \quad \Rightarrow (\ [\lambda_1 \cdot \lambda_2 \mid j],\ [\ ]\ ,\ \beta\ ,\ E\ )$ DT: $\lambda_1 = [\ ]$, NT: $\nexists k \in \lambda_1.\ k \rightarrow j \vee k \leftarrow j$
NO-ARC	$(\ [\lambda_1 \mid i],\ \lambda_2\ ,\ [j \mid \beta],\ E\ ) \Rightarrow (\ \lambda_1\ ,\ [i \mid \lambda_2],\ [j \mid \beta],\ E\ )$ default transition

Table 1: Transitions in our algorithm. For each row, the first line shows a transition and the second line shows preconditions of the transition.

2 Reducing search space

Our algorithm is based on Choi-Nicolov's approach to Nivre's list-based algorithm (Nivre, 2008). The main difference between these two approaches is in their implementation of the SHIFT transition. Choi-Nicolov's approach divides the SHIFT transition into two, deterministic and non-deterministic SHIFT's, and trains the non-deterministic SHIFT with a classifier so it can be predicted during decoding. Choi and Nicolov (2009) showed that this implementation reduces the parsing complexity from $O(n^2)$ to linear time in practice (a worst-case complexity is $O(n^2)$).

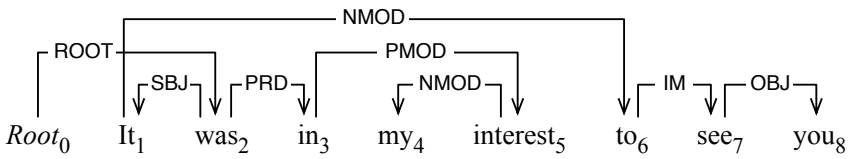
We suggest another transition-based parsing approach that reduces the search space even more. The idea is to merge transitions in Choi-Nicolov's non-projective algorithm with transitions in Nivre's projective algorithm (Nivre, 2003). Nivre's projective algorithm has a worst-case complexity of $O(n)$, which is faster than any non-projective parsing algorithm. Since the number of non-projective dependencies is much smaller than the number of projective dependencies (Nivre and Nilsson, 2005), it is not efficient to perform non-projective parsing for all cases. Ideally, it is better to perform projective parsing for most cases and perform non-projective parsing only when it is needed. In this algorithm, we add another transition to Choi-Nicolov's approach, LEFT-POP, similar to the LEFT-ARC transition in Nivre's projective algorithm. By adding this transition, an oracle can now choose either projective or non-projective parsing depending on parsing states.[1]

[1]We also tried adding the RIGHT-ARC transition from Nivre's projective algorithm, which did not improve parsing performance for our experiments.

Note that Nivre (2009) has a similar idea of performing projective and non-projective parsing selectively. That algorithm uses a SWAP transition to reorder tokens related to non-projective dependencies, and runs in linear time in practice (a worst-case complexity is still $O(n^2)$). Our algorithm is distinguished in that it does not require such reordering.

Table 1 shows transitions used in our algorithm. All parsing states are represented as tuples (λ_1, λ_2, β, E), where λ_1, λ_2, and β are lists of word tokens. E is a set of labeled edges representing previously identified dependencies. L is a dependency label and i, j, k represent indices of their corresponding word tokens. The initial state is ([0], [ ], [1,...,n], $\emptyset$). The 0 identifier corresponds to an initial token, w_0, introduced as the root of the sentence. The final state is (λ_1, λ_2, [ ], E), i.e., the algorithm terminates when all tokens in β are consumed.

The algorithm uses five kinds of transitions. All transitions are performed by comparing the last token in λ_1, w_i, and the first token in β, w_j. Both LEFT-POP$_{\mathrm{L}}$ and LEFT-ARC$_{\mathrm{L}}$ are performed when w_j is the head of w_i with a dependency relation L. The difference is that LEFT-POP removes w_i from λ_1 after the transition, assuming that the token is no longer needed in later parsing states, whereas LEFT-ARC keeps the token so it can be the head of some token $w_{j<k\leq n}$ in β. This $w_i \rightarrow w_k$ relation causes a non-projective dependency. RIGHT-ARC$_{\mathrm{L}}$ is performed when w_i is the head of w_j with a dependency relation L. SHIFT is performed when λ_1 is empty (DT) or there is no token in λ_1 that is either the head or a dependent of w_j (NT). NO-ARC is there to move tokens around so each token in β can be compared to all (or some) tokens prior to it.



	Transition	λ_1	λ_2	β	E
0		[0]	[ ]	[1|β]	$\emptyset$
1	SHIFT (NT)	[λ_1|1]	[ ]	[2|β]	
2	LEFT-ARC	[0]	[1]	[2|β]	$E \cup \{1 \leftarrow_{\text{SBJ}} 2\}$
3	RIGHT-ARC	[ ]	[0|λ_2]	[2|β]	$E \cup \{0 -_{\text{ROOT}}\rightarrow 2\}$
4	SHIFT (DT)	[λ_1|2]	[ ]	[3|β]	
5	RIGHT-ARC	[λ_1|1]	[2]	[3|β]	$E \cup \{2 -_{\text{PRD}}\rightarrow 3\}$
6	SHIFT (NT)	[λ_1|3]	[ ]	[4|β]	
7	SHIFT (NT)	[λ_1|4]	[ ]	[5|β]	
8	LEFT-POP	[λ_1|3]	[ ]	[5|β]	$E \cup \{4 \leftarrow_{\text{NMOD}} 5\}$
9	RIGHT-ARC	[λ_1|2]	[3]	[5|β]	$E \cup \{3 -_{\text{PMOD}}\rightarrow 5\}$
10	SHIFT (NT)	[λ_1|5]	[ ]	[6|β]	
11	NO-ARC	[λ_1|3]	[5]	[6|β]	
12	NO-ARC	[λ_1|2]	[3|λ_2]	[6|β]	
13	NO-ARC	[λ_1|1]	[2|λ_2]	[6|β]	
14	RIGHT-ARC	[0]	[1|λ_2]	[6|β]	$E \cup \{1 -_{\text{NMOD}}\rightarrow 6\}$
15	SHIFT (NT)	[λ_1|6]	[ ]	[7|β]	
16	RIGHT-ARC	[λ_1|5]	[6]	[7|β]	$E \cup \{6 -_{\text{IM}}\rightarrow 7\}$
17	SHIFT (NT)	[λ_1|7]	[ ]	[8|β]	
18	RIGHT-ARC	[λ_1|6]	[7]	[8|β]	$E \cup \{7 -_{\text{OBJ}}\rightarrow 8\}$
19	SHIFT (NT)	[λ_1|8]	[ ]	[ ]	

Table 2: Parsing states for the example sentence. After LEFT-POP is performed (#8), [w_4 = *my*] is removed from the search space and no longer considered in the later parsing states (e.g., between #10 and #11).

During training, the algorithm checks for the preconditions of all transitions and generates training instances with corresponding labels. During decoding, the oracle decides which transition to perform based on the parsing states. With the addition of LEFT-POP, the oracle can choose either projective or non-projective parsing by selecting LEFT-POP or LEFT-ARC, respectively. Our experiments show that this additional transition improves both parsing accuracy and speed. The advantage derives from improving the efficiency of the choice mechanism; it is now simply a transition choice and requires no additional processing.

3 Bootstrapping automatic parses

Transition-based parsing has the advantage of using parse history as features to make the next prediction. In our algorithm, when w_i and w_j are compared, subtree and head information of these tokens is partially provided by previous parsing states. Graph-based parsing can also take advantage of using parse information. This is done by performing 'higher-order parsing', which is shown to improve parsing accuracy but also increase parsing complexity (Carreras, 2007; Koo and Collins, 2010).[2] Transition-based parsing is attractive because it can use parse information without increasing complexity (Nivre, 2006). The qualification is that parse information provided by gold-standard trees during training is not necessarily the same kind of information provided by automatically parsed trees during decoding. This can confuse a statistical model trained only on the gold-standard trees.

To reduce the gap between gold-standard and automatic parses, we use bootstrapping on automatic parses. First, we train a statistical model using gold-

[2]Second-order, non-projective, graph-based dependency parsing is NP-hard without performing approximation.

standard trees. Then, we parse the training data using the statistical model. During parsing, we extract features for each parsing state, consisting of automatic parse information, and generate a training instance by joining the features with the gold-standard label. The gold-standard label is achieved by comparing the dependency relation between w_i and w_j in the gold-standard tree. When the parsing is done, we train a different model using the training instances induced by the previous model. We repeat the procedure until a stopping criteria is met.

The stopping criteria is determined by performing cross-validation. For each stage, we perform cross-validation to check if the average parsing accuracy on the current cross-validation set is higher than the one from the previous stage. We stop the procedure when the parsing accuracy on cross-validation sets starts decreasing. Our experiments show that this simple bootstrapping technique gives a significant improvement to parsing accuracy.

4 Related work

Daumé et al. (2009) presented an algorithm, called SEARN, for integrating search and learning to solve complex structured prediction problems. Our bootstrapping technique can be viewed as a simplified version of SEARN. During training, SEARN iteratively creates a set of new cost-sensitive examples using a known policy. In our case, the new examples are instances containing automatic parses induced by the previous model. Our technique is simplified because the new examples are not cost-sensitive. Furthermore, SEARN interpolates the current policy with the previous policy whereas we do not perform such interpolation. During decoding, SEARN generates a sequence of decisions and makes a final prediction. In our case, the decisions are predicted dependency relations and the final prediction is a dependency tree. SEARN has been successfully adapted to several NLP tasks such as named entity recognition, syntactic chunking, and POS tagging. To the best of our knowledge, this is the first time that this idea has been applied to transition-based parsing and shown promising results.

Zhang and Clark (2008) suggested a transition-based projective parsing algorithm that keeps B different sequences of parsing states and chooses the one with the best score. They use beam search and show a worst-case parsing complexity of $O(n)$ given a fixed beam size. Similarly to ours, their learning mechanism using the structured perceptron algorithm involves training on automatically derived parsing states that closely resemble potential states encountered during decoding.

5 Experiments

5.1 Corpora and learning algorithm

All models are trained and tested on English and Czech data using automatic lemmas, POS tags, and feats, as distributed by the CoNLL'09 shared task (Hajič et al., 2009). We use Liblinear L2-L1 SVM for learning (L2 regularization, L1 loss; Hsieh et al. (2008)). For our experiments, we use the following learning parameters: $c = 0.1$ (cost), $e = 0.1$ (termination criterion), $B = 0$ (bias).

5.2 Accuracy comparisons

First, we evaluate the impact of the LEFT-POP transition we add to Choi-Nicolov's approach. To make a fair comparison, we implemented both approaches and built models using the exact same feature set. The 'CN' and 'Our' rows in Table 3 show accuracies achieved by Choi-Nicolov's and our approaches, respectively. Our approach shows higher accuracies for all categories. Next, we evaluate the impact of our bootstrapping technique. The 'Our+' row shows accuracies achieved by our algorithm using the bootstrapping technique. The improvement from 'Our' to 'Our+' is statistically significant for all categories (McNemar, $p < .0001$). The improvment is even more significant in a language like Czech for which parsers generally perform more poorly.

	English		Czech	
	LAS	UAS	LAS	UAS
CN	88.54	90.57	78.12	83.29
Our	88.62	90.66	78.30	83.47
Our+	**89.15***	**91.18***	**80.24***	**85.24***
Merlo	88.79 (3)	-	80.38 (1)	-
Bohnet	89.88 (1)	-	80.11 (2)	-

Table 3: Accuracy comparisons between different parsing approaches (LAS/UAS: labeled/unlabeled attachment score). * indicates a statistically significant improvement. (#) indicates an overall rank of the system in CoNLL'09.

Finally, we compare our work against other state-of-the-art systems. For the CoNLL'09 shared task, Gesmundo et al. (2009) introduced the best transition-based system using synchronous syntactic-semantic parsing ('Merlo'), and Bohnet (2009) introduced the best graph-based system using a maximum spanning tree algorithm ('Bohnet'). Our approach shows quite comparable results with these systems.[3]

5.3 Speed comparisons

Figure 1 shows average parsing speeds for each sentence group in both English and Czech evaluation sets (Table 4). 'Nivre' is Nivre's swap algorithm (Nivre, 2009), of which we use the implementation from MaltParser (`maltparser.org`). The other approaches are implemented in our open source project, called ClearParser (`code.google.com/p/clearparser`). Note that features used in MaltParser have not been optimized for these evaluation sets. All experiments are tested on an Intel Xeon 2.57GHz machine. For generalization, we run five trials for each parser, cut off the top and bottom speeds, and average the middle three. The loading times for machine learning models are excluded because they are independent from the parsing algorithms. The average parsing speeds are 2.86, 2.69, and **2.29** (in milliseconds) for Nivre, CN, and **Our+**, respectively. Our approach shows linear growth all along, even for the sentence groups where some approaches start showing curves.

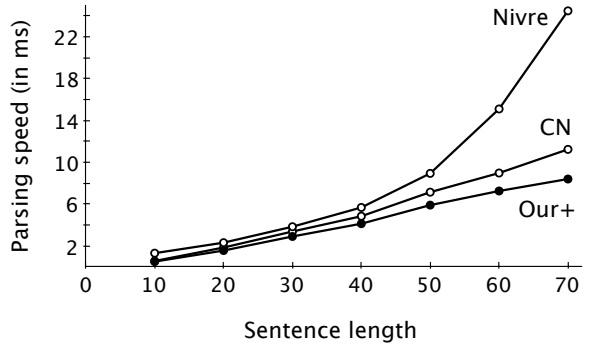


Figure 1: Average parsing speeds with respect to sentence groups in Table 4.

[3]Later, 'Merlo' and 'Bohnet" introduced more advanced systems, showing some improvements over their previous approaches (Titov et al., 2009; Bohnet, 2010).

< 10	< 20	< 30	< 40	< 50	< 60	< 70
1,415	2,289	1,714	815	285	72	18

Table 4: # of sentences in each group, extracted from both English/Czech evaluation sets. '$< n$' implies a group containing sentences whose lengths are less than n.

We also measured average parsing speeds for 'Our', which showed a very similar growth to 'Our+'. The average parsing speed of 'Our' was 2.20 *ms*; it performed slightly faster than 'Our+' because it skipped more nodes by performing more non-deterministic SHIFT's, which may or may not have been correct decisions for the corresponding parsing states.

It is worth mentioning that the curve shown by 'Nivre' might be caused by implementation details regarding feature extraction, which we included as part of parsing. To abstract away from these implementation details and focus purely on the algorithms, we would need to compare the actual number of transitions performed by each parser, which will be explored in future work.

6 Conclusion and future work

We present two ways of improving transition-based, non-projective dependency parsing. The additional transition gives improvements to both parsing speed and accuracy, showing a linear time parsing speed with respect to sentence length. The bootstrapping technique gives a significant improvement to parsing accuracy, showing near state-of-the-art performance with respect to other parsing approaches. In the future, we will test the robustness of these approaches in more languages.

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation Grants CISE-IIS-RI-0910992, Richer Representations for Machine Translation, a subcontract from the Mayo Clinic and Harvard Children's Hospital based on a grant from the ONC, 90TR0002/01, Strategic Health Advanced Research Project Area 4: Natural Language Processing, and a grant from the Defense Advanced Research Projects Agency (DARPA/IPTO) under the GALE program, DARPA/CMO Contract No. HR0011-06-C-0022, subcontract from BBN, Inc. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Bernd Bohnet. 2009. Efficient parsing of syntactic and semantic dependency structures. In *Proceedings of the 13th Conference on Computational Natural Language Learning: Shared Task (CoNLL'09)*, pages 67–72.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *The 23rd International Conference on Computational Linguistics (COLING'10)*.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL'07 (CoNLL'07)*, pages 957–961.

Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. 2010. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'10)*.

Jinho D. Choi and Nicolas Nicolov. 2009. K-best, locally pruned, transition-based dependency parsing using robust risk minimization. In *Recent Advances in Natural Language Processing V*, pages 205–216. John Benjamins.

Isaac G. Councill, Ryan McDonald, and Leonid Velikovich. 2010. What's great and what's not: Learning to classify the scope of negation for improved sentiment analysis. In *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing (NeSp-NLP'10)*, pages 51–59.

Hal Daumé, Iii, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325.

Andrea Gesmundo, James Henderson, Paola Merlo, and Ivan Titov. 2009. A latent variable model of synchronous syntactic-semantic parsing for multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning: Shared Task (CoNLL'09)*, pages 37–42.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL'09): Shared Task*, pages 1–18.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. 2008. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning (ICML'08)*, pages 408–415.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP'05)*, pages 523–530.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL:HLT'08)*, pages 950–958.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03)*, pages 23–25.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP'09)*, pages 351–359.

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL:HLT'08)*, pages 577–585.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1562–1567.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'08)*, pages 562–571.