Optimizing the Oslo-Bergen Tagger

Eckhard Bick
University of Southern Denmark
Odense
eckhard.bick@mail.dk

Kristin Hagen & Anders Nøklestad
University of Oslo, Norway
kristin.hagen@iln.uio.no
anders.noklestad@iln.uio.no

Abstract

In this paper we discuss and evaluate machine learning-based optimization of a Constraint Grammar for Norwegian Bokmål (OBT). The original linguist-written rules are reiteratively re-ordered, re-sectioned and systematically modified based on their performance on a hand-annotated training corpus. We discuss the interplay of various parameters and propose a new method, continuous sectionizing. For the best evaluated parameter constellation, part-of-speech F-score improvement was 0.31 percentage points for the first pass in a 5-fold cross evaluation, and over 1 percentage point in highly iterated runs with continuous resectioning.

1 Introduction and prior research

Typical Constraint Grammars consist of thousands of hand-written linguistic rules that contextually add, change or discard token-based grammatical tags for lemma, part-of-speech (POS), morphological feature-value pairs, syntactic function, semantic roles etc. Each rule interacts intricately with all other rules, because the application of a rule will change the grammatical sentence context for all subsequent rules, and section-based rule iteration further complicates this process. Thus, a CG grammarian can only trace rule effects for one token at a time, and only for rules that actually are used. As a consequence, improvements are made in a piecemeal fashion, while it is practically impossible for a human to meaningfully rearrange the grammar as a whole. We therefore believe that most CGs could potentially profit from data-driven, automatic optimization.

Early work in this direction was the μ-TBL system (Lager 1999), a transformation based learner that could be seeded with CG rule templates, for which it would find optimal variations and rule order with the help of a training corpus. However, μ-TBL did not perform as well as human grammars, and could only handle n-gram-type context conditions. Lindberg & Eineborg' Progol system (1998) induced CG REMOVE rules from annotated Swedish data and achieved a recall of 98%, albeit with a low precision (13% spurious readings). The first system to use automatic optimization on existing, linguist-written grammars, was described in Bick (2013), and achieved a 7% error reduction for the POS module of the Danish DanGram[1] parser. Results were twice as good for a randomly reduced grammar with only 50% of the original rules, indicating a potential for grammar boot-strapping and grammar porting to another language or genre, where only part of the existing rules would be relevant, which was later shown to be true for at least the Danish-English language pair (Bick 2014). In the work presented here we examine how Bick's optimization method holds up for a Norwegian Bokmål CG, and discuss different parameter options.

2 The Oslo-Bergen Tagger (OBT)

The Oslo-Bergen Tagger is a rule-based Constraint Grammar (CG) tagger for the Norwegian varieties Bokmål and Nynorsk. Below we will give a brief presentation of the OBT history, the architecture behind it, and the

1 DanGram is accessible on-line at http://visl.sdu.dk/visl/da/parsing/automatic/parse.php

OBT performance. The presentation will focus on the Bokmål tagger.

2.1 History

OBT was developed in 1996–1998 by the Tagger Project at the University of Oslo. The linguistic rules were written at the Text Laboratory in the CG1 rule framework (Karlsson et. al. 1995). Originally, the tagger used a rule interpreter from the Finnish company Lingsoft AB. The tagger performed both morphological and syntactic analysis (Johannessen et. al. 2000). In 2000 the preprocessor and the rule interpreter was replaced by a reimplementation in Allegro Common Lisp made by Aksis (now Uni Research Computing) in Bergen, and the tagger was named The Oslo-Bergen Tagger.

Within the project Norwegian Newspaper Corpus (2007-2009), OBT was once again converted. The CG rules were semi-automatically transformed from CG1 to the new CG3 formalism (Bick & Didriksen 2015), and Uni Research Computing made a stand-alone version of the preprocessor that could work together with the CG3 compiler[2].

Finally, a statistical module was trained to remove the last ambiguity left by OBT. This module also performed lemma disambiguation, a task the original OBT did not do. The new system was called OBT+stat and is described in more detail in Johannessen et. al. (2012).

In this article we will focus on the morphological CG part of OBT+stat since the optimization is performed on the morphological rules without regard to the statistical module.

2.2 The architecture behind OBT

The morphological part of OBT consists of two modules:

a) Preprocessor: The preprocessor is a combined tokenizer, morphological analyzer and guesser that segments the input text into sentences and tokenizes their content. There are special rules for names and various kinds of abbreviations. Each token is assigned all possible tags and lemmas from the electronic lexicon *Norsk ordbank* (Norwegian Word Bank). The Bokmål part of this lexicon contains more than 150 000 lemmas together with inflected forms (Hagen & Nøklestad 2010). The guesser includes a compound word analyzer and manages productively formed compounds as well as unknown words (Johannessen & Hauglin 1998).

b) Morphological disambiguator: The morphological disambiguator is based on CG3 rules that select or remove tags attached to the input tokens. There are 2279 linguistic rules in this module. 693 of them are rules for specific word forms. 1371 are SELECT rules and 908 are REMOVE rules.

The tag set is rather large, consisting of 358 morphological tags. The part of speech classifications, which include information about morphosyntactic features, are performed in accordance with Norsk referansegrammatikk (Faarlund et. al. 1997).

2.3 OBT performance

The original tagger was tested on an unseen evaluation corpus of 30,000 words taken from a wide variety of material such as literary fiction, magazines and newspapers. The recall and precision were 99.0 and 95.4 percent respectively, with a combined F-measure of 97.2 (Hagen & Johannessen 2003:90)

After the conversion to CG3 format, recall remained at 99.0 percent while precision increased to 96.0 percent, resulting in an F-measure of 97.5 (see the OBT homepage).

3 Grammar optimization

For our experiments we used a 155.000 word[3] corpus with hand-corrected OBT tags covering POS and inflection. The original corpus was divided into a (larger) development section and a (smaller) testing section. We used this division for the parameter-tuning experiments, but for the final results, in order to avoid a bias from human rule development, we fused these sections and created sections of equal size to be used in 5-fold cross evaluation. We also adapted the OBT grammar itself, because its rules have operators, targets and contexts on separate lines, sometimes with interspersed comments. Although CG3-compatible, this format had to be changed into 1-line-per-rule in order to make rule movements and rule ordering possible.

3.1 Optimization technique and parameters

The optimization engine works by computing for

[2] CG3 (or vislcg-3) is open source and available at SDU: http://visl.sdu.dk/constraint_grammar.html

[3] when counting all tokens, including punctuation

each rule in the grammar a performance value based upon how often it selects or removes a correct reading in the training corpus. Rules are then reordered, removed or modified in reaction to this value, and the effect measured on the test corpus. After this, the process is repeated with the new grammar, and so on. As in Bick (2013, 2014), we investigated the following actions and parameters:

(1) Sorting rules according to performance

(2) Promoting good rules the next-higher section (good = error percentage lower than T/2)

(3) "Demoting" bad rule to the next-lower section (bad = error percentage higher than T)

(4) "Killing" very bad rules (error percentage is over 0.5, doing more bad than good)

(5) Add relaxed versions of good rules, by removing C- (unambiguity-) conditions and, conversely, changing BARRIERs into CBARRIERS

(6) Replace bad rules with stricter versions, by adding C conditions and turning CBARRIER into BARRIER.

(7) "Wordform stripping" - i.e. adding a relaxed version of a wordform-conditioned rule without this condition, thus allowing it to work on all tokens.

Performance values (recall, precision and F-score) for the modified grammars, in the tables below, are for POS (i.e. without inflection), and because OBT has more than one tag for a comma, evaluation also includes punctuation.

3.2 What did not work, or in limited ways

The maybe most obvious step, sorting rules section-internally after each iteration, decreased performance. Sorting was only helpful in dry-run[4] mode, for an initial ordering of the original grammar and after the 1. iteration's addition of modified rules[5], and only in the combination of complete sorting plus resectioning. regardless if sorting was performed for sections individually or for the whole grammar together. The importance of re-sectioning indicates that the existence and placement of sections is an important parameter, that the concept of a good/bad rule is section-dependent[6], and that it is an important optimization parameter. All runs in the table below were with standard PDK and 1-time dry-run sorting and examine different combinations of iterative sorting. As can be seen, section-internal sorting worked worst (F=96.26), having a kill-section helped more (F=96.49) than factoring in human sectioning as a weight (F=96.26). But in all instances, iterated sorting was worse than 1-time sorting (F=96.56). Increasing the number of sections to 11 led to a certain recovery of F-scores in hight iterations, but could not beat the first run in our experiment window (50 runs).

PDK	ite-ration	Recall (%)	Precision (%)	F-score
original grammar		98.08	94.27	96.13
no iterated sorting	3	98.72	94.50	96.56
sorting 5 sections	1	98.23	94.66	96.41
sorting 11 sections (--> F-score recovery)	1	98.22	94.66	96.41
sorting11, +kill	1	98.33	94.72	96.49
sorting2, +kill	1	98.32	94.72	96.49
sorting11, -kill, human section-weighting	11	98.25	94.36	96.26
sorting5 sect.-internal	14	96.99	94.35	95.65

Effects of iterated sorting

Word form stripping was reported in Bick (2014) to have had a positive effect in cross language grammar porting, but we could not reproduce this effect in the monolingual setup with the same parameters (PDK, 1 dry-sorting). Cross-language, the method can possibly offset the problem that wordforms from one language don't exist in the other, creating versions of the rules

[4] In a dry-run call, CG3 applies all rules once, but without making changes to the input. Rule tracing in a dry-run will therefore is a way to measure how rules would perform in isolation without actually running thousands of 1-rule minigrammars.

[5] The rule change that profited from sorting was stripping of wordform target conditions, because this change creates very unrestrained and dangerous rules.

[6] 10% errors, for instance, is good in a heuristic section, where most of the disambiguation has been done already, but may be bad if the rule is run too early, where the error rate may be the same in relative terms, but worse in absolute terms, because it will apply to more cases.

that work at least in terms of POS etc., while this effect is not relevant monolingually, were wordform rules just get more risky by losing their pivotal wordform condition. However, with a subsequent second dry-run sorting and very high, section-growing iteration counts, the optimizer seems to be able to identify the useful subset of wordform-stripped rules and find acceptable section placements for them (cf ch. 4).

In terms of numerical parameters we experimented with the error threshold, but failed to find a better value the 0.25 suggested by (Bick 2013) - both lower and higher thresholds decreased performance, independently of other parameter settings[7].

3.3 What did work

While the negative effect of sorting confirms results for Danish, there was a surprising difference regarding promoting, demoting and killing. For DanGram, demoting and killing rules had a beneficial effect, while promoting rules had almost no effect. For OBT, however, rule promotion (P) was important, and for a combination of only demoting (D) pseudokilling (k3) extra iterations did not yield a better effect than the initial one-time sectioned rule-sorting (S0).

	R	dR	P	dP	F	dF
unaltered grammar	98.08		94.27		96.13	
S0D(s)k3, -w (i=1/50[8])	98.22	0.14	94.66	0.39	94.41	0.28
S0PDsk3, -w (i=45/50)	98.71	0.63	94.85	0.58	96.75	0.62

Effect of rule promoting

This can possibly be explained by different grammar properties: OBT is recall-optimized (there are about 4 times as many spurious readings than errors[9]), while DanGram resolves almost all ambiguity (i.e. one reading per token), so that precision will roughly equal recall. Therefore OBT profits from promoting good rules (so they can do more disambiguation work), while DanGram, on the recall side, profits from demoting and killing rules (preventing them from removing correct reading). Another difference between the two grammar is that OBT has a higher proportion of SELECT rules and a lower proportion of C contexts and BARRIERs. Because C and BARRIER contexts are harder to instantiate[10] (needing more supporting context disambiguation first), and because SELECT resolves ambiguity in one go that REMOVE rules would have needed several steps for, it can be said that DanGram's rules work more incrementally and indirectly, while OBT is more direct in its disambiguation. This harmonizes with the finding that promoting helped OBT, but not DanGram, because promoting makes sense for rules that are formulated as "absolute truths" (SELECT rules), but doesn't help for rules whose C and BARRIER contexts force them to wait for other rules to work first anyway.

	DanGram	OBT
morph. rules	5120	2215
REMOVE	2837 (55.4%)	898 (40.5%)
SELECT	2178 **(42.5%)**	1312 **(59.2%)**
OTHER	105 (2.1%)	5 (0.2%)
wordform rules	1808 (35.3%)	777 (35.1%)
contexts	17539 (3.48/rule)	11525 (5.55/rule)
C conditions	4549 **(25.9%)**	1303 **(11.3%)**
NOT	3239 (18.5%)	5007(43.4%)
NEGATE	477 (2.7%)	5 (0.0%)
LINK	4192 (23.9%)	2705 (23.5%)
BARRIERS	3554 **(20.3%)**	927 **(8.0%)**
CBARRIERS	276 (1.6%)	44 (0.4%)
global contexts	4597 (26.2%)	3326 (28.9%)

Grammar properties DanGram - OBT

7 However, we did not have computational resources to experiment with exponent changes or the *0.5 difference between good rule and bad rule thresholds.

8 F-Score stabilized below the initial sorting optimization, independently of whether new sections were added or not, for the latter from iteration 11, at 96.224, for the former from iteration 5, at 96.222.

9 This 1:4 ratio holds for both for our own tests (R=98, P=92) and Hagen & Johannessen's (2003) evaluation (R=99, P=96)

10 The effect may be somewhat compensated for, however, by the higher percentage of NOT rules in OBT, which also makes rules more "cautious".

Killing rules does have a slight initial effect in OBT, but over several iterations, the effect is negligible. We therefore introduced a compromise parameter into the optimizer, defining "killing" as moving a rule 10[11] sections down, rather than removing it completely it from the grammar. This did not have an adverse effect, and while most of these rules never moved up again in later iterations, a few of them can do a little late heuristic work in low sections. The method also preserves the rules in question for use with other text types, reducing the risk of over-fitting a very lean grammar to a specific training corpus. Metaphorically speaking, we preserve a varied "gene pool" of rules as a reserve for a new data environment. In the same vein we decided to preserve unused rules (rather than going for an efficiency gain in a leaner grammar)[12].

We also noted a positive effect, especially on precision, from adding relaxed versions of good rules and a smaller positive effect from making bad rules stricter.

3.4 Sectionizing

Sectionizing the grammar is not only a key parameter with any sorting configuration, but also in general. The standard optimizer from (Bick 2013) adds one new first section for moving up rules from the original first section, and had a positive effect from this. But we wanted to test the hypothesis that more sections will lead to a more fine-grained quality ordering of rules and exploit the fact that the CG compiler will try rules *twice* (or even three times) within the same section, and rerun higher-section rules before it starts on the next lower (= more heuristic) section. So we added new top and bottom sections at each iteration, allowing the grammar to differentiate more when promoting and demoting existing and changed rules (PDN).

	R	dR	P	dP	F	dF
	98.08		94.27		96.13	
S0PDk3, -w (i=1)	98.22	0.14	94.66	0.39	94.41	0.28
S0PDNk3, -w (i=45/50)	98.71	0.63	94.85	0.58	96.75	0.62

Effect of sectionizing (S0=1 dry sorting, k3=killing by moving 3 sections down)

Continually adding extra sections to the grammar had a marked dampening effect on the performance oscillation of the iteration curve. Also, performance kept increasing, with late stabilization, and a maximum at iteration 45 in the example, whereas most runs with a stable section number had their F-Score maximum already in the first iteration, and stabilized somewhere between the unoptimized performance and this first maximum. In theory, the section-adding technique can end up section-separating individual rules, making the process equivalent to precise one-by-one rule ordering, which conceptually beats the group ordering achieved by fixed-section optimization.

The obvious price for adding new sections was slower execution - with a worst case ceiling at quadratic growth in time consumption (because the compiler reruns lower sections before embarking on a new one). In practice, however, the distribution of rules across sections was lumpy. For instance, when adding sections for top/bottom-moved rules but not removing empty sections, the grammar from iteration 51 in the above test had 20 used and 32 empty sections, grammar 100 had 15 used and 62 empty sections[13].

It should be noted that once a grammar is optimized, execution time can be improved at a fairly small price by reducing the number of sections. Thus, F-score decreased only marginally when we resectioned an optimized 50-section grammar to 6 equal sections. However, the effect on recall and precision was unequal - the former fell by 0.5 percentage points, the latter rose by 0.4 percentage points. Both effects can be explained by strong but dangerous rules acting too early. Still, in average F-score terms, optimization with a very fine-grained section skeleton will more or less amount to individual rule ordering and therefore

[11] We also tried a lower number, 3, which did not work as well.

[12] Rules may be unused only because they are placed in a certain section, so they can also come back into play in later iterations during training, when other - higher - rules, that did their work for them, are demoted to a section below a given inactive rule.

[13] In a final, working grammar, empty sections should of course be removed, but during optimization empty in-between sections allow more fine-grained rule differentiation and seemed to have a slight positive effect

tolerate de-sectioning. In our experiment, even removing *all* sections borders still did not seriously harm F-Score. Thus, sections are important for the optimization process, but in an optimized grammar they are less important than in the human original.

4 Final results

To achieve as reliable results as possible, we used a 5-fold cross evaluation for the final evaluation. For the best parameter setting (dry sorting, promoting and demoting with new sections, pseudokilling), the initial F-score optimization gain (1. iteration) varied from 0.27 to 0.36 percentage points between the 5 combinations, with an absolute F-score spread of 95.58 to 96.60. As might be expected, the weakest sections (3 and 4) profited the most from optimization.

S0PDNk3, -w	R	dR	P	dP	F	dF
1: (i=1/6)	98.63	0.16	94.71	0.43	96.60	0.30
2: (i=1/6)	98.42	0.20	94.95	0.35	95.65	0.28
3: (i=1/6)	97.67	0.22	93.58	0.48	95.58	0.36
4: (i=1/6)	97.38	0.26	93.9	0.44	95.60	0.36
5: (i=1/6)	98.24	0.16	94.7	0.38	96.41	0.27
average (i=1/6)	98.07	0.20	94.35	0.42	96.17	0.31

Table: Performance spread across the corpus

(S0=dry run sorting only (with resectioning=5 and 10-killing), PDs=Promoting & Demoting with new sections per iteration, k3=3-section demoting instead of killing, -w=no wordform stripping)

With new promoting/demoting sections for every round, performance maxima tend to occur late in the iteration cycle, so to investigate the ultimate improvement potential, we used the section with the weakest dry run (3) and let iteration run for 100 rounds. Because each such run took over half a day on our hardware, we were only able to investigate few parameter settings at the time of writing. The best result, an F-Score improvement (dF) of 1.31, was achieved with reintroducing ordinary killing at iteration 15, and a maximum in round 86. When introducing a second "dry" sorting in iteration 2, i.e. after the addition of relaxed and stricted rules, and ordinary killing from iteration 4, much shorter training runs were needed (with an asymptotic maximum already in round 16), albeit at a slightly lower level (dF=1.09). With this setting, even word form-stripping could be tolerated, with a maximal dF of 0.51 in round 63. Here, too, the growth curve was asymptotic, but it still oscillated until the end, so later maxima can't be entirely ruled out - and would make sense, given the very "un-cautious" character of wordform-stripped rules. It is probably these "un-cautious" rules that explain why wordform-stripped rules benefited precision twice as much as recall, while high iterations otherwise had a strong recall bias.

Performance oscillations for the training corpus correlated with performance on the test set, but test corpus results for grammars with training corpus maxima[14] could deviate up to 0.1% from the actual test corpus peak, which is a rough measure for the expected "performance impredictability" when using ML-optimized grammars on unknown data.

test chunk 3[15]	R	dR	P	dP	F	dF
i=0	97.46		93.10		95.23	
S0PDNk3K15, -w (i-max=86/100)	99.09	1.63	94.11	1.01	96.54	1.31
training i=86	99.26		84.99		91.57	
S2PDNk3K4, -w (imax = 16 const.)	98.86	1.40	93.90	0.80	96.32	1.09
training i=16	99.07		85.41		91.73	
S2PDNk3K4, -w (i-max=63/100)	97.80	0.34	93.76	0.66	95.74	0.51
training i=63	98.04		89.18		93.40	

Best case scenario - "Unlimited" iterations

5 Conclusion

We have demonstrated that ML-optimization can be successfully performed for a Norwegian constraint grammar, and explored a new sectioning strategy and the respective influences

[14] F-scores for training runs appear to be lower than for the test corpus, but only because the optimizer in the training runs evaluates against full tag lines, i.e. with inflection and secondary tags, not just POS.

[15] We ran the second parameter setting for the original training/test-split too, with the same asymptotic result. F-Sore topped at 96.67, 0.53 percentage point above the unaltered grammar (F=96.14), but the lower increase has to be interpreted on the basis of a higher base line.

of rule sorting and rule movements. For most parameter constellations, repetition of optimization runs did not lead to a better performance than a single pass, unless each iteration is allowed to add new sections. The first-pass average improvement in 5-fold cross evaluation was 0.31 percentage points (F-Score 96.17), similar to Danish results reported in Bick (2013), but with added sectionizing and long iterations, improvements of over 1 percentage point were seen, corresponding to a 30% improvement in relative terms. The immediate effect was best for precision, but with high iterations, recall was affected most, with a 60% improvement in relative terms.

Future work would certainly profit from access to a large computer cluster, to investigate the millions of possible combinations of incremental parameter changes. Also, it would be interesting to get the human linguist back into the loop, to see if some of the rules slated for killing or demotion by the optimizer can be "saved" by additional context conditions instead, and if the best selected generalized variants of wordform rules can be used for further development.

References

Bick, Eckhard. 2013. ML-Tuned Constraint Grammars. In: Proceedings of the 27th Pacific Asia Conference on Language, Information and Computation, pp. 440-449. Taipei: Department of English, National Chengchi University.

Bick, Eckhard. 2014. ML-Optimization of Ported Constraint Grammars. In: Calzolari, Nicoletta et al. (eds.), Proceedings of the 9th International Conference on Language Resources and Evaluation, LREC2014 (Reykjavik, May 28-30, 2014). pp. 3382-3386.

Bick, Eckhard & Didriksen, Tino. 2015. CG-3 - Beyond Classical Constraint Grammar. In: Proceedings of NoDaLiDa 2015 (forthcoming).

Faarlund, Jan Terje, Lie, Svein & Vannebo, Kjell Ivar. 1995. Norsk referansegrammatikk. Oslo: Universitetsforlaget.

Hagen, Kristin & Nøklestad, Anders. 2010. Bruk av et norsk leksikon til tagging og andre språkteknologiske formål. *LexicoNordica* 2010 (17) pp. 55-72.

Hagen, Kristin & Johannessen, Janne Bondi. 2003. Parsing Nordic Languages (PaNoLa) - norsk versjon. Nordisk Sprogteknologi 2002. Museum Tusculanums Forlag, Københavns universitet.

Johannessen, Janne Bondi and Helge Hauglin. 1998. An Automatic Analysis of Norwegian Compounds. In Haukioja, T. (ed.): Papers from the 16th Scandinavian Conference of Linguistics, Turku/Åbo, Finland 1996 : 209-220.

Johannessen, Janne Bondi, Hagen, Kristin & Nøklestad, Anders. 2000. A Constraint-based Tagger for Norwegian. In 17th Scandinavian Conference of Linguistics [Odense Working Papers in Language and Communication 19].

Johannessen, Janne Bond; Hagen, Kristin; Lynum, André; Nøklestad, Anders. 2012. OBT+stat: A combined rule-based and statistical tagger. In Andersen, Gisle (ed.). Exploring Newspaper Language: Using the web to create and investigate a large corpus of modern Norwegian, s. 51–66.

Karlsson, Fred, Voutilainen, Atro, Heikkilä, Juha & Anttila, Arto. 1995. Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text. In Natural Language Processing, No 4. Berlin and New York: Mouton de Gruyter.

Lager, Torbjörn. 1999. The µ-TBL System: Logic Programming Tools for Transformation-Based Learning. In: Proceedings of CoNLL'99, Bergen.

Lindberg, Nikolaj & Eineborg, Martin. 1998. Learning Constraint Grammar-style Disambiguation Rules using Inductive Logic Programming. COLING-ACL 1998: 775-779

Norsk Ordbank 'Norwegian Word Bank'. 2010. http://www.hf.uio.no/iln/om/organisasjon/edd/forsking/norsk-ordbank/.

Oslo-Bergen Tagger homepage. <http://tekstlab.uio.no/obt-ny/>.