CuteForce – Deep Deterministic HPSG Parsing

Gisle Ytrestøl
Department of Informatics / University of Oslo
gisley@ifi.uio.no

Abstract

We present a deterministic HPSG parser capable of processing text incrementally with very fast parsing times. Our system demonstrates an efficient data-driven approach that achieves a high level of precision. Through a series of experiments in different configurations, we evaluate our system and compare it to current state-of-the-art within the field, and show that high quality deterministic parsing is realistic even for a 'deep' unification-based precision grammar.

1 Motivation

The traditional chart parsing paradigm for text parsing has been to process one sentence at a time, and through some type of dynamic programming algorithm derive the most probable analysis. For a formalism like HPSG, even short sentences can license hundreds of valid (i.e. grammatical) analyses, and at some point a statistical model has to be employed to reduce and pin down an n-best candidate list. In order to circumvent a costly extraction of the full parse forest, deterministic incremental parsing has enjoyed an increased interest in the research community. Deterministic parsing can mitigate both the time and space complexity challenges often associated with probabilistic chart parsing methods. The deterministic incremental approach is attractive both from a computational and a cognitive standpoint. Whereas traditional chart parsing approaches require the entire sentence as input before producing an analysis, an incremental algorithm will incrementally expand a syntactic/semantic derivation as it reads the input sentence one word/token at a time. There are several attractive features to this approach. The time-complexity will be linear when the algorithm is deterministic, i.e. when it does not allow for later changes to the partial derivation. For a number of applications, e.g. speech recognition, the ability to process input on the fly per word, and not per sentence, can also be vital.

This paper introduces a 'deep' incremental deterministic HPSG parser, dubbed *CuteForce*. It has an optional unification mode, ensuring that all parse derivations are valid HPSG analyses. Unification is a costly operation, and for certain applications it may be a desirable trade-off to gain a significant speed-up by replacing unification with less expensive constraint checking.

2 Related Work

While there is a rich research tradition in statistical parsing, the predominant approach derives from chart parsing and is inherently non-deterministic. In the following, we review alternative approaches to statistical parsing, with a focus on deterministic and incremental processing.

2.1 Deterministic Parsing

Deterministic parsing aims to derive one single analysis provided the input string and the grammar. As almost any medium to long sentence carries substantial inherent ambiguity given even a moderately simple grammar, this would in practice mean to disambiguate on the fly by making a sequence of local choices that continuously pursue the globally optimal derivation.

Dependency Grammar Kudo and Matsumoto (2002) introduced an efficient deterministic dependency parser for Japanese. Their parser outperformed previous probabilistic models with respect to accuracy and efficiency. Yamada and Matsumoto (2003) applied a similar method for English, and received a near state of the art accuracy compared to other dependency grammar parsers.

The method was further investigated with MaltParser (Nivre et al., 2007; Nivre and Scholz, 2004). MaltParser is a platform for data-driven dependency parsing, which can induce a parsing model from treebank data and parse new input using a pre-compiled parsing model. The parser is formalized as a transition-based instantiation of a shift-reduce parser. It has been applied to a range of different languages, and Nivre et al. (2007) report that dependency accuracy consistently ranges from 80 to 90 %, “usually with less than a 5% increase in error rate compared to state-of-the art parsers for the language in question.”

CFG Sagae and Lavie (2005) present a deterministic CFG parser grounded in the same classifier-based strategy that was pursued by MaltParser. Given that the search-space of constituent trees is larger than for dependency trees, the potential efficiency gain of deterministic parsing could be higher than for dependency parsing, but the margin of error would also be larger, given that a deterministic CFG parse is likely to reach more decision points, with more alternative paths than the same sentence would have encountered in dependency parsing. Although they do not reach the same precision/accuracy as ‘classic’ state-of-the-art parsers (e.g. Collins (1997), Charniak (2000), Charniak and Johnson (2005)), “the simplicity and efficiency of deterministic parsers make them attractive in a number of situations requiring fast, light-weight parsing, or parsing of large amounts of data.” (Sagae and Lavie, 2005)

HPSG The increased complexity of unification-based grammars like HPSG could potentially license a substantial increase in efficiency by parsing deterministically, but the inherent hard constraints of unification-based grammar could cause a high number of parse failures. Parallel to the development of the CuteForce parser, Ninomiya et al. (2010; 2009) provide a deterministic shift-reduce parser for HPSG where this issue is addressed. To mitigate the problem of parse failures, they suggest default unification (Ninomiya et al., 2002; Copestake, 1993; Carpenter, 1993) by overwriting inconsistent constraints in the grammar, outlining a deterministic and a non-deterministic configuration.

Ninomiya et al. (2010) evaluate their parser on Enju2.3ß, an HPSG for English, which is extracted from Section 02-21 of the Penn Treebank (Miyao and Tsujii, 2005; Miyao et al., 2004). Their deterministic parser using default unification reaches a coverage of 95%. They further conclude that “[t]he experiments revealed that deterministic parsing with default unification achieved a high degree of accuracy of 87.6 per cent for the Penn Treebank with gold part-of-speech (POS) tags.” (Ninomiya et al., 2010)

Further, Matsuzaki et al. (2007) provide a fast, partially deterministic shift-reduce HPSG parser. The parser requires a preceding non-deterministic supertagging and CFG-filtering stage prior to the unification-based parsing which is done through a deterministic shift-reduce algorithm, and gives “comparable accuracy with a speed-up by a factor of six (30 msec/sentence) compared with the best published result using the same grammar.” (Matsuzaki et al., 2007)

2.2 Traditional Unification-Based Parsing

Whereas the parser proposed by Ninomiya et al. (2010; 2009) is to our knowledge the only other effort to deterministically parse HPSG, traditional chart parsing approaches for unification-based grammars have been presented in a number of research efforts (e.g. Riezler et al. (2000), Kaplan et al. (2004), Malouf and van Noord (2004), Miyao and Tsujii (2005), Toutanova et al. (2005)). Since unification is a destructive, hence non-reversible operation, a non-deterministic parsing algorithm will need to preserve the original feature structure in order to keep the entire parse forest intact. Keeping the parse forest intact is crucial when constructing an *n*-best list over the most probable HPSG analyses that is licensed by the grammar, as the individual parse tree will contain diverging feature structures. This has been addressed in several research papers, as this could potentially be a major bottleneck for non-deterministic HPSG parsing (Callmeier, 2000; Oepen and Carroll, 2000; Zhang et al., 2007).

PET HPSG Parser The PET platform was developed as a tool for experimentation with HPSG processing and implementation techniques (Callmeier, 2000). It has further been developed through subsumption-based packing (Oepen and Carroll, 2000), selective unpacking and statistical parse ranking (Zhang et al., 2007). It is maintained by DELPH-IN[1], which is a cooperation between academic institutions and researchers working with deep linguistic processing in HPSG.

The PET HPSG Parser will for each input sentence use a hand-crafted grammar (e.g. LinGO ERG) to retrieve the candidate analyses that are

[1]`http://www.delph-in.net/`

grounded in the grammar, thus being *well-formed*. During parsing it attempts to create the full packed parse forest for each input sentence, and produces an *n*-best list based on statistical parse ranking. Parsing will fail if the full parse forest cannot be constructed within a preset time limit (normally 60 seconds).

3 LinGO ERG and Redwoods Treebank

There exists a number of different HPSG grammars used for automatic parsing. The parser presented in this paper uses LinGO (Linguistic Grammars Online) ERG (English Resource Grammar), which is a broad-coverage, linguistically precise handcrafted grammar for English (Flickinger, 2000). It is continuously being developed and expanded to increase coverage for new domains.

The Redwoods treebank is a collection of annotated corpora from a variety of domains, including tourism guides, transcribed scheduling dialogs, ecommerce emails and Wikipedia articles (Oepen et al., 2002). Each analysis in the treebank is manually disambiguated and assigned an HPSG analysis in accordance with ERG. Its use in this project is recorded in Table 1. HPSG signs corresponding to lexical types are developed manually and assigned in accordance with the lexical coverage of the grammar. Each derivation within Redwoods is grounded in ERG, ensuring that the analyses from each individual treebank are kept consistent with the overall grammar.

WikiWoods WikiWoods is a collection of automatically annotated Wikipedia articles, in total 1.3 million articles and 55 million utterances. WeScience (*ws*) can be seen as a manually annotated subset of WikiWoods (Flickinger et al., 2010; Ytrestøl et al., 2009). The WikiWoods corpus is parsed with the PET HPSG parser using the 1010 release of ERG, and will contain a proportion of incorrect analyses. In a manual inspection of 1,000 random utterances, roughly 82 % of the analyses were deemed correct or nearly correct (Flickinger et al., 2010).

In this paper we augment the training data with derivations from WikiWoods, using the derivations from Section 2000 and onwards. As demonstrated in this paper, both the supertagger as well as CuteForce benefits greatly from the augmented training data.

3.1 Statistics and Example

Table 1 provides an overview of the Redwoods treebanks which are used in this paper.

Name	#Sent	∼Length	Usage
ws1-11	7636	14.4	train
ws12	601	15.6	dev
ws13	785	14.1	test
logon	8501	13.3	train
vm	11116	6.8	train
sc	2564	15.1	train

Table 1: The corpora in Redwoods treebank used for training and testing in this paper.
#Sent refers to the number of sentences. ∼Length is the average number of tokens per sentence for the treebank.
ws – WeScience, Wikipedia articles
logon – Tourism guides
vm – Verbmobil corpus, transcribed dialogs
sc – SemCor, subset of English Brown Corpus

Figure 1 presents an HPSG derivation for a simple sentence. This tree of ERG rules can be presented as a feature structure, and a semantic MRS representation can be derived directly from its structure.

Lexical Rules and Lexical Types The derivation in Figure 1 is composed of lexical and syntactic rules. Further, each token is assigned a lexical type, which is conceptually equivalent to a supertag (see Section 4). From the leaves in Figure 1 we see that punctuation is considered a part of the token, e.g. for *RSS-*. The lexical types end with the *_le* suffix. The lexical type for “RSS-”, *n_-_pn_le*, provides the HPSG sign template for a proper noun. The conceptual sign is further augmented by the lexical ERG rules, which end with **lr*. The hyphen in “RSS-” is hence annotated with *w_hyphen_plr*.

For “specialized” the lexical type *v_np*_le* denotes the lexical category verb (*v*), with an optional noun phrase subcategorization argument. *v_pas_odlr* denotes a passive verb, and *v_j-nb-pas-tr_dlr* derives an attributive adjective from a transitive passive verb. Altogether there are 50 lexical rules in the ERG, and around 1,000 lexical types.

Syntactic Rules Syntactic ERG rules have a *_c* suffix. The rules can be unary or binary. The root node in Figure 1, *sb-hd_mc_c*, denotes the conventional *head-subject* main clause in HPSG, in

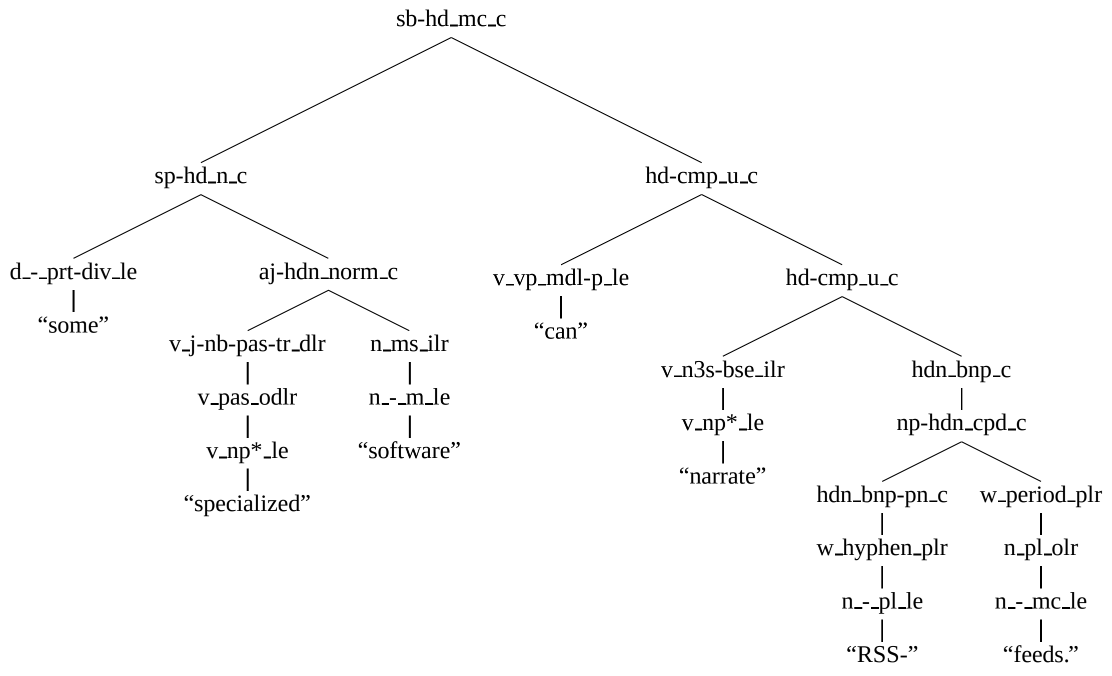

Figure 1: HPSG derivation from WeScience.

turn connecting a *head+specifier* (*sp-hd_n_c*) and a *head+complement* (*hd-cmp_u_c*) phrase in a binary production. There are in total 145 binary and 55 unary syntactic rules in the ERG.

4 Preprocessing and Supertagging

CuteForce assumes an input buffer consisting of the tokenized words, part-of-speech tags and lexical types. For tokenization we use the gold standard ERG tokenization provided by the treebank. HPSG lexical types are supertags providing a lexical template for the input token (Bangalore and Joshi, 1999). Currently, there are 1186 lexical types in the grammar, and these templates can be seen as an extension to the token, enriching the word with HPSG lexical information, corresponding to the HPSG sense of a word sign.[2] Dridan (2009) showed that supertagging contributes to both speed-up and increased coverage for the PET HPSG Parser. Although it would be feasible to integrate supertagging into the parsing oracle, assigning supertags incrementally during parsing, we have opted for a pre-processing stage for performance reasons, a design choice which is parallel to MaltParser.

Dridan (2009) investigated supertagging of lexical types within ERG. The thesis concludes that the TnT tagger (Brants, 2000) performs better than C&C Tagger (Clark and Curran, 2007) when trained on the limited training data that was available in 2009. However, the learning curve for the TnT tagger seemed to flatten out at around 150,000 tokens, whereas the learning curve for C&C was still rising.

We trained the C&C tagger on the Redwoods treebank, and augmented it with sentences from the WikiWoods corpus (see Table 1). The findings in Dridan (2009) suggest that non-gold standard data may improve the C&C tagger. Additionally we developed a customized feature model trained with SVM^{hmm}, which is an implementation of structural SVMs for sequence tagging which learns a hidden Markov model (Joachims et al., 2009). Whereas the C&C tagger is optimized for supertagging, and has a relatively fixed configuration, SVM^{hmm} requires that the user designs features specifically aimed for the classification task. This allows the user to address certain unconventional aspects of ERG supertagging, most especially the tokenization that includes punctuation. On smaller sets of training data, SVM^{hmm} outperformed C&C. However, C&C trains much faster, and is capable of training on much larger amounts of data than SVM^{hmm}.

Results The best single tag accuracy in Dridan (2009) refers to Section 2 of WeScience, since this was released prior to the final release of WeScience. In Table 2 we present the lexical type accuracies (*lt acc.*) on WeScience Section 13 for the

[2]892 different lexical types are found in the training data.

best configurations for SVM^{hmm} and the C&C tagger, along with the best single lexical type accuracy for WeScience 2 reported in Dridan (2009). *# sent* refers to the number of sentences in the training data, and *cpu hours* is the number of hours it took to train the model. Although we were able to train even larger models for the C&C tagger, the accuracy seemed to flatten out at approximately 6,800,000 tokens on our development set, so we have applied this model. For the C&C tagger, we used the default parameter settings.

	TnT [3]	C&C	SVM^{hmm}
lt acc.	0.864	0.953	0.934
# sent	≈10,000	6,800,000	300,000
cpu hours	<0.1	314	480

Table 2: Single tag accuracy on WeScience Section 2 (TnT) and Section 13 (C&C and SVM^{hmm}).

Consistent with the assumption in Dridan (2009), augmenting the training data contributes substantially to the performance of the tagger. C&C and SVM^{hmm} (Table 2) are trained on 6,800,000 and 300,000 sentences respectively, approximately 94 million/4 million tokens. C&C has a comparatively low accuracy when the amount of training data is limited. We assume the punctuation regime in ERG works is C&C's disadvantage when the training data is limited, since it will result in a large lexicon with comparatively low frequency per lexical entry. When increasing the training data, this sparsity problem will have less impact – it is even plausible that this contributes to the overall performance of C&C when the amount of training data is high, because the punctuation provides enriched information about the token.

Part-of-speech tags are assigned prior to supertagging. POS tags are not part of the final HPSG derivation, but are used both by the parsing oracle in CuteForce and by the supertagger. The POS tags used in training and test data are tagged using the TnT POS tagger, applying the pre-compiled English WSJ model bundled with the tagger.

[3]From the best single tag accuracy reported in Dridan (2009) on Section 2 of *ws*. Training time is not stated in Dridan (2009), but in TnT it should only be a matter of minutes for such a small data set. This model is trained on 1,941 sentences from WeScience, and additional data from the Redwoods treebank, approximately 10,000 sentences, and WeScience should hence be considered (mostly) out-of-domain for this model, compared to the models used in the training of the C&C and SVM^{hmm} tagger.

5 CuteForce – Deterministic Incremental HPSG Parsing

Our parser, CuteForce, employs a classifier-based *oracle* to guide the shift-reduce parser that incrementally builds a syntactic/semantic HPSG derivation that conforms to the LinGO English Resource Grammar (ERG).

Parser Layout The sentence input buffer β is a list of tuples with token, part-of-speech tags and HPSG lexical types (supertags).

Given a set of ERG rules R and a sentence buffer β, a parser configuration is a tuple $c = (\alpha, \beta, \iota, \pi)$ where:

- α is a stack of "active" edges[4]
- β is a list of tuples of word forms W, part of speech tags POS and lexical types LT derived from a sentence $x = ((W_1, POS_1, LT_1), ...(W_n, POS_n, LT_n))$.
- ι is the current input position in β
- π is a stack of passive edges instantiating an ERG rule

The stack of passive edges π makes up the full HPSG representation of the input string if the string is accepted.

Transition System The shift-reduce parser has four different transitions, two of which are parameterized with a unary or binary ERG rule, which are added to the passive edges, hence building the HPSG structure. The four transitions are:

- ACTIVE – (adds an active edge to stack α, and increments ι)
- UNIT(R^1) – (adds unary passive edge to π instantiating unary ERG rule (R^1))
- PASSIVE(R^2) – (pops α and adds binary passive edge to π instantiating binary ERG rule (R^2))
- ACCEPT – (terminates the parse of the sentence. π represents the HPSG derivation of the sentence)

[4]An "active" edges in our sense is a hypothesis of an application of a binary rule where the left daughter is known (an element of π), and the specific binary ERG rule and the right daughter is yet to be found.

Parsing Configuration Mode The parser can operate in three oracle configurations: HPSG Unification mode, CFG approximation mode and unrestricted mode.

In HPSG Unification mode, the parser validates that each transition implies a valid unification. This is done through an XML-RPC interface to LKB (Copestake, 2002). We expect that a substantial speedup could be obtained if this implementation was done natively in CuteForce[5]. All UNIT and PASSIVE transitions are implicit unifications. For each parsing stage, the parsing oracle returns a ranked list of transitions. The highest-ranked transition not violating a unification constraint will be executed. If no transitions yield a valid unification, parsing fails for the given sentence. All non-failing parses (i.e. parses that terminates with the ACCEPT transition) are ensured to produce a valid HPSG derivation.

In CFG mode, a naive CFG approximation of the ERG is employed to guide the oracle. The CFG approximation consists of CFG rules harvested from the parser's training data, augmented with derivations from WikiWoods, in total 300,000 sentences. Each ERG rule instantiation, using the identifiers shown in Figure 1 as non-terminal symbols, will be treated as a CFG rule, and each parser action will be validated against the set of CFG rules. If the parser action yields a CFG projection not found among the valid CFG rules in the CFG approximation, the CFG filter will block this transition. If the parser arrives at a state where the CFG filter blocks all further transitions, parsing fails.

In unrestricted mode, the oracle chooses the highest scoring transition without any further restrictions imposed. In this setting, the parser typically reaches close to 100 % coverage – the only sentences not covered in this setting are instances where the parser enters an infinite unit production loop, and the sentence is dismissed.[6]

[5]Specifically, the current unification back-end performs non-destructive unification, i.e. it does not take advantage of the deterministic nature of CuteForce.

[6]One would require additional heuristics to avoid unary loops in an incremental parsing scheme that allows for such productions. In Sagae and Lavie (2005) they force a non-unary production after three consecutive unary transitions in order to break a potential loop.

5.1 Machine Learning Model

A discriminative machine learning model is used to predict the parser action. The model is trained and tested on the WeScience Treebank, a branch of the hand-annotated LinGO Redwoods treebank (Ytrestøl et al., 2009; Oepen et al., 2002). Section 1-11 is used for training, Section 12 is used in development and Section 13 is held-out for testing. The training data is further augmented with additional sentences from other Redwoods treebanks (see Table 1), and derivations from the automatically annotated Wikiwoods corpus. Parsing results for CuteForce initially improve when the size of the training data increases, but the full extent of the effect of training on partially incorrect training data is not yet clear. Section 3 provides a more in-depth presentation of the training data used in this paper. The parsing is reduced to a classification problem. Each HPSG analysis is a derivation from one unique sequence of parser actions $T = t_1, t_2 \dots t_n$ The input histories are feature vectors representing a parser state, and the output class is a transition t. This can be seen as a deterministic implementation of a History-based model, introduced by Black et al. (1993). Formally, a derivation D is a sequence of the highest scoring transitions T:

$$\arg\max_{t_i} P(t_i \mid \Phi(t_1, .., t_{i-1}))$$

The function Φ maps the current state, or *history*, to a feature vector. The vector is further defined through a set of feature functions, see Section 5.2.

For training we use LibLinear (Fan et al., 2008), which provides a number of solvers. In CuteForce, we have used the implementation of SVM multi-class classification of Crammer and Singer's formula (Vapnik, 1995; Keerthi et al., 2008; Crammer and Singer, 2002), which has given better results than other learners evaluated during the development.

5.2 Feature Model

CuteForce is equipped with a rich feature model optimized for a large (100,000+) set of training derivations. In our training data of 150,000 training derivations, we have approximately 6 million training instances, where each instance represents a parser action and is mapped to a feature vector. We distinguish between *static* and *dynamic* features, where the static features are defined prior to parsing and only depend on the properties of the input buffer, whereas the dynamic features are defined by the HPSG derivation that is partially built during parsing. Part-of-speech tags and ERG lexical types (supertags) are annotated in a preprocessing stage.

For the history-based model, we expand the formal parser definition in Section 5 with a position index j for α and π, where 0 denotes the top of the stack, -1 denotes the next stack element etc.

Feature Functions We define a set of feature functions to describe the features used in the feature vector:

- $\beta_{(\iota)}$ is the ιth $W/POS/LT$ tuple in the input buffer, where ι denotes the current buffer position.
- BL is the length of buffer β
- $W_{(\iota)}$ is the ιth word form in β.
- $LT_{(\iota)}$ is the ιth lexical type in β.
- $POS_{(\iota)}$ is the ιth Part-of-Speech tag in β, annotated by the preprocessor (TNT).
- $LC_{(\iota)}$ is the lexical category tag derived from $LT_{(\iota)}$.
- $SubCat_{(\iota)}$ is the Subcategorization field derived from $LT_{(\iota)}$.
- $IP_{(\iota)}$ and $FP_{(\iota)}$ denote word-initial and word-final punctuation in the word $W_{(\iota)}$, respectively.
- $\alpha_{(j)}$ is the jth unification candidate edge from the top of the active edge stack.
- $\pi_{(j)}$ is the jth edge from the top of the passive edge stack.
- $l_{(e)}$ is the left-branched daughter of the edge e.
- $r_{(e)}$ is the right-branched daughter of the (passive) edge e.
- $h_{(e)}$ is the HPSG head of the edge e. $h^*_{(e)}$ denotes the head-relation down to the preterminal.
- $ER_{(e)}$ is the ERG rule of the (passive) edge e.
- $S_{(\alpha/\pi)}$ denotes the size of the α and π stacks.

Feature Templates Each training instance maps 70 features to the feature vector. In this paper we limit ourselves to presenting the feature functions corresponding to the atomic features extracted for each parsing state (see Table 3). In the feature vector, most of the atomic features occur in conjunction with other features, and only a few of the features will occur by themselves. A combination of two or more features is necessary to represent the inherent dependence many features have on one another, given that we train the model linearly. For our model derived from 150,000 sentences, we extracted approximately 6 million features, using a frequency cutoff of 3.

SF	$W_{(\iota-1,\iota,\iota+1)}$,$POS_{(\iota-1,\iota,\iota+1,\iota+2)}$ $LC_{(\iota-1,\iota,\iota+1,\iota+2)}$,$\iota$, $BL-\iota$ $LT_{(\iota-1,\iota,\iota+1,\iota+2)}$,$FP_{(\iota)}$,$IP_{(\iota)}$ $SubCat_{(\iota,\iota+1,\iota+2)}$
DF	$S_{(\alpha)}$,$ER_{(\pi_{(0)})}$,$ER_{(h_{(\pi_{(0)})})}$,$ER_{(h_{(h_{(\pi_{(0)})})})}$ $ER_{(h^*_{(\pi_{(0)})})}$,$ER_{(h_{(l_{(\alpha_{(0)})})})}$,$ER_{(h_{(h_{(l_{(\alpha_{(0)})})})})}$ $ER_{(h^*_{(l_{(\alpha_{(0)})})})}$,$ER_{(l_{(\pi_{(0)})})}$,$ER_{(r_{(\pi_{(0)})})}$ $ER_{(l_{(l_{(\alpha_{(0)})})})}$,$ER_{(r_{(l_{(\alpha_{(0)})})})}$,$ER_{(l_{(\alpha_{(-1)})})}$

Table 3: Static and dynamic features.

5.3 Training Data

Given the available treebanks and corpora we have at our disposal (see Section 3), we evaluated a number of different training data configurations. In addition to corpora from the Redwoods treebank, we can extend the training model with WikiWoods data.

In Figure 2 we observe that the accuracy of CuteForce improves when extending the number of training derivations. Figure 2 presents results for parsing without CFG or Unification filtering using gold standard lexical types.

Training data from the WeScience treebank amounts to 7,636 sentences. When training on the same amount of sentences from other (out of domain) treebanks from Redwoods, we see a clear drop in precision – both the in-domain WeScience treebank, and maybe more surprisingly, the annotated WikiWoods corpus is a better source for training than the out-of-domain, yet gold standard, Redwoods data.

Training solely on WikiWoods annotation yields lower performance when the amount of training data is limited, but from 30,000 sentences there are only minor differences in the per-

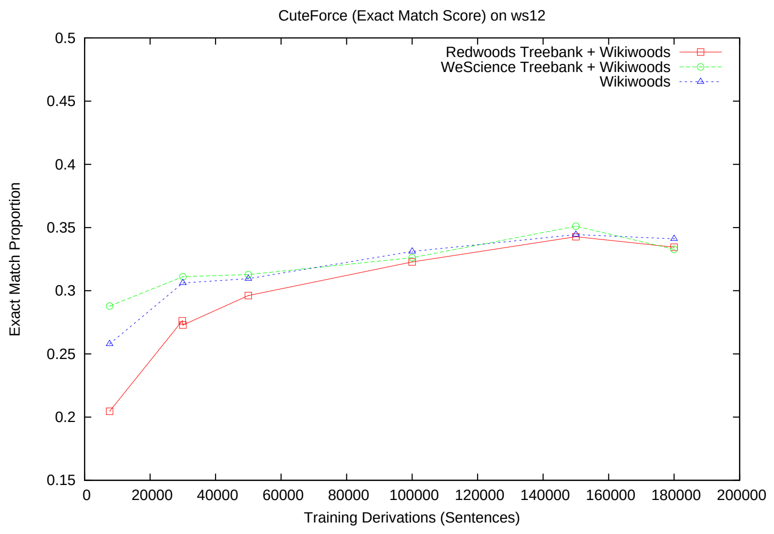

Figure 2: The exact match proportion (proportion of sentences with an HPSG analysis identical to the gold standard) increases when adding more training sentences. The WeScience Treebank augumented with WikiWoods derivations, totally 150,000 sentences, reaches the highest accuracy.

formance for the two in-domain configurations. There are at least two trends in Figure 2 that seem obvious: First, in-domain data is a better training source than out-of-domain data – even when the training data contains a proportion of errors. Second, when the amount of training data is very limited, training on the WeScience treebank yields the best results. However, when we extend the amount of WikiWoods derivations in the training data, these differences diminish, and from 100,000 training sentences onwards, there seems to be no substantial differences between the training data configurations. At about 150,000 derivations, the curve appears to flatten out, and even starting to decline. For the remainder of the paper, we continue using the model trained on WeScience, augmented with Wikiwoods annotation, totally 150,000 derivations, as this configuration achieved the highest accuracy on the development data set.

In this iteration we have indiscriminately used WikiWoods sentences from Section 2000 and onwards. It could prove beneficial to reject sentences in the training data that are obvious outliers. This could be very short (one word) sentences, very long sentences, sentences partially or completely in a foreign language[7], or sentences that have other properties that may imply that they are unsuited to use as training data. This will be subject for further research.

[7]Certain articles in the English Wikipedia contain large amount of foreign texts, for example articles concerning a specific language.

6 Evaluation

CuteForce is evalutated on Section 13 from WeScience (*ws13*), and the experiments were carried out on an Intel Xenon(R) server with 2 GHz CPU.

6.1 Deterministic HPSG Parsers

To our knowledge, the parser outlined in Ninomiya et al. (2010) is the only effort in incremental deterministic unification-based parsing, along side with CuteForce. However, a complete head-to-head comparison of the parse analyses produced by the parsers is not possible due to formal and technical difference in the underlying grammars. Whereas the HPSG Enju grammar used by Ninomiya et al. (2010) is induced from the Penn Treebank, the ERG is a handcrafted grammar. Where the granularity of an induced grammar like Enju is largely determined by information available in the treebank, the ERG includes more fine-grained and richer analyses, for example with respect to subcategorization patterns (including relational nouns and adjectives), multi-word expressions and other subtle linguistic properties that cannot easily be extracted from a treebank with heuristics alone.

Even if the parser proposed by Ninomiya et al. (2010) had been publicly available, adapting it to ERG would not be straightforward, because the typed feature structure logic assumed in the ERG is formally richer than that of Enju. Alternatively, one could opt for a partial evaluation of the analyses generated by the two parsers, e.g. by extracting dependency relations (Clark and Curran (2007), Cahill et al. (2008)). This would be subject for further investigation.

Instead of directly comparing the two deterministic parsers, we can however see how they relate to traditional non-deterministic parsers. Ninomiya et al. (2010) present an extensive evaluation of their deterministic parser, compared to other non-deterministic Enju HPSG parsers. In the following, we will compare CuteForce to the PET HPSG Parser, which is the de facto HPSG parser for the DELPH-IN consortium.

6.2 CuteForce vs PET HPSG Parser

The PET HPSG Parser is the most widely used parser for ERG. Its design is distinctly different from CuteForce (see Section 2.2). Comparing the two parsers based on available testing data is non-trivial: Whereas Ytrestøl et al. (2009) report a parsing coverage of 86 % for the sentences annotated in WeScience, it is only the sentences that were actually covered that appear in the WeScience Treebank. Hence, the PET Parser will have a 100 % coverage in all the test sentences we are able to evaluate. To accommodate for this, we have evaluated the parsers on test sentences where the coverage overlaps for both parsers, hence we have divided the results according to the parser mode in which CuteForce is operating. Table 4 presents the parser scores on *ws13* when CuteForce is using gold standard supertags, whereas Table 5 presents the same score when CuteForce is applying supertagged input.

6.3 Parsing Results

The oracle mode determines the coverage for CuteForce, hence the subset of the test data that is evaluated for each configuration. *M* in Table 4 and 5 refers to the CuteForce oracle mode, where *N* in unrestricted, *C* is CFG approximation mode, and *U* is unification mode. *Ex* is the proportion of sentences that was parsed identically to the gold standard derivation. *Cov* refers to the corresponding coverage, only relevant for CuteForce[8]. *F1* is the Parseval F1-score when treating the gold standard and the parsed analysis as a CFG tree structure. *TA* is the lexical type (supertag) accuracy. All tree derivation scores are computed by *evalb*[9].

Val is the proportion of the parsed analyses that are valid HPSG derivations, and $\sim$*SL* refers to the average sentence length for the subset. *mps* is the average number of millisecond per sentence used by the parser (this excludes preprocessing time done by the supertagger).

In Table 4 and 5 we see that with gold standard supertags, CuteForce is comparable, and even more accurate than PET in certain validation matrices. When parsing with supertagged input, PET is in overall better. However, when considering the F1-score, it seems clear that the parse derivations produced by CuteForce have high quality. Parsing time is consistently 15 msec/sentence unless we apply unification. This is to our knowledge the fastest parsing time reported by a parser on a major HPSG grammar.

We see that the proportion of sentences that are well-formed HPSG derivations are in the range from 0.47-0.59, depending on the configuration. Although a fairly high share of the derivations will not yield a unifiable HPSG derivation, it is however likely that one could extract a partially well-formed semantic structure through robust unification (Fouvry, 2003). This is also addressed in Zhang and Krieger (2011), and will be evaluated in further studies.

7 Future Work

There are alternative paths that could be pursued to attempt to improve on the current configuration. Instead of using single tags, one could let the supertagger assign multiple supertags. Zhang and Clark (2011) present a shift-reduce parser for CCG where the parsing oracle chooses between a set of candidate supertags assigned by the supertagger. Hence they are able to take syntactic information into account when choosing the supertag. A similar approach could be beneficial for CuteForce.

Allowing for non-determinism would open for a number of strategies. Ytrestøl (2011) evaluated a backtracking algorithm where ranking is applied to locate and redo an incorrect transition done by CuteForce. Alternatively, one could consider a beam-search strategy in line with Ninomiya et al. (2010). This will be subject for further study.

[8]Since the treebanks only consist of sentences that the PET parser was able to parse, PET will have 100 % coverage in all configurations.

[9]`http://nlp.cs.nyu.edu/evalb/`

	M	Cov	Ex	F1	TA	Val	∼SL	mps
CF	N	0.99	0.42	0.86	1	0.51	14.1	15
PET	-	1	0.48	0.87	0.965	1	14.1	2.3k
CF	C	0.81	0.52	0.89	1	0.59	12.0	15
PET	-	1	0.55	0,88	0.966	1	12.0	2.3k
CF	U	0.57	0.77	0.94	1	1	8.6	1.6k
PET	-	1	0.66	0.90	0.972	1	8.6	2.3k

Table 4: Parsing results for CuteForce (gold standard lexical types) and PET HPSG Parser on *ws13*, evaluated on sentences where the coverage overlaps for both parsers.

	M	Cov	Ex	F1	TA	Val	∼SL	mps
CF	N	0.99	0.36	0.82	0.953	0.47	14.0	15
PET	-	1	0.48	0.87	0.965	1	14.0	2.3k
CF	C	0.79	0.45	0.85	0.959	0.59	11.9	15
PET	-	1	0.55	0.88	0.966	1	11.9	2.3k
CF	U	0.52	0.70	0.92	0.974	1	8.1	1.6k
PET	-	1	0.69	0.90	0.976	1	8.1	2.3k

Table 5: Parsing results for CuteForce (supertagged lexical types) and PET HPSG Parser on *ws13*, evaluated on sentences where the coverage overlaps for both parsers.

8 Concluding Remarks

We have presented an efficient deterministic parsing approach to HPSG. Whereas unification-based parsing traditionally has been associated with non-deterministic parsers, we have demonstrated a deterministic system capable of achieving a high level of precision with very fast parsing times. Although it is questionable if a deterministic system could ever reach the same precision-level as state-of-the-art non-deterministic systems, deterministic parsing would be attractive for applications where it is desirable to trade some precision for high-speed incremental parsing.

Acknowledgement

The author would like to thank the anonymous reviewers for their helpful suggestions, and Stephan Oepen (University of Oslo) and Joakim Nivre (Uppsala University) for their valued input and inspiring feedback during the writing of this paper, and in the PhD project in general. Experimentation and engineering was made possible through access to the TITAN high-performance computing facilities at the University of Oslo (UiO), and we are grateful to the Scientific Computation staff at UiO, as well as to the Norwegian Metacenter for Computational Science.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics*, pages 237–265.

Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Meeting of the Association for Computational Linguistics*, pages 31–37.

Thorsten Brants. 2000. TnT - a statistical part-of-speech tagger. In *Proceedings of the 6th ACL Conference on Applied Natural Language Processing*, pages 224–231.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-coverage deep statistical parsing using automatic dependency structure annotation. 34:81–124.

Ulrich Callmeier. 2000. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99 – 108.

Bob Carpenter. 1993. Skeptical and credulous default unification with applications to templates and inheritance. In *Inheritance, defaults and the lexicon*, pages 13–37. Cambridge University Press.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the*

43th Meeting of the Association for Computational Linguistics, pages 173–180.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 132–139. Morgan Kaufmann Publishers Inc.

Stephen Clark and James Curran. 2007. Formalism-independent parser evaluation with ccg and depbank. In *Proceedings of the 45th Meeting of the Association for Computational Linguistics*, pages 248–255.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the Association for Computational Linguistics and the 7th Conference of the European Chapter of the ACL*, pages 16–23.

Ann Copestake. 1993. Defaults in lexical representation. In *Inheritance, defaults and the lexicon*, pages 223–245. Cambridge University Press.

Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications.

Koby Crammer and Yoram Singer. 2002. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292.

Rebecca Dridan. 2009. *Using Lexical Statistics to Improve HPSG Parsing*. Ph.D. thesis, Saarland University.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874.

Dan Flickinger, Stephan Oepen, and Gisle Ytrestøl. 2010. Wikiwoods: Syntacto-semantic annotation for english wikipedia. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*. European Language Resources Association (ELRA).

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6 (1):15 – 28.

Frederik Fouvry. 2003. *Robust Processing for Constraint-Based Grammar Formalisms*. Ph.D. thesis, Univeristy of Essex.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. 2009. Cutting-plane training of structural SVMs. *Machine Learning*, 77:27–59.

Ronald M. Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell III, and Er Vasserman. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*.

S. Sathiya Keerthi, S. Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. 2008. A sequential dual method for large scale multi-class linear SVMs. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’08, pages 408–416. ACM.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–7. Association for Computational Linguistics.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP workshop Beyond Shallow Analysis*.

Takuya Matsuzaki, Yusuke Miyao, and Jun’ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and cfg-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1671–1676.

Yusuke Miyao and Jun’ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage hpsg parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 83–90. Association for Computational Linguistics.

Yusuke Miyao, Takashi Ninomiya, and Jun ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In

Proceedings of the 1nd International Joint Conference on Natural Language Processing, pages 684–693.

Takashi Ninomiya, Yusuke Miyao, and Jun'ichi Tsujii. 2002. Lenient default unification for robust processing within unification based grammar formalisms. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7.

Takashi Ninomiya, Nobuyuki Shimizu, Takuya Matsuzaki, and Hiroshi Nakagawa. 2009. Deterministic shift-reduce parsing for unification-based grammars by using default unification. In *Proceedings of the 12th Conference of the European Chapter of the ACL*, pages 603–611. Association for Computational Linguistics.

Takashi Ninomiya, Takuya Matsuzaki, Nobuyuki Shimizu, and Hiroshi Nakagawa. 2010. Deterministic shift-reduce parsing for unification-based grammars. *Natural Language Engineering*, pages 1–35.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics*. Association for Computational Linguistics.

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

Stephan Oepen and John Carroll. 2000. Ambiguity packing in constraint-based parsing. Practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 162 – 169.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Chris Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank. Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics*.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, pages 480 – 487.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies*, pages 125–132. Association for Computational Linguistics.

Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. In *Research in Language and Computation*.

Vladimir N. Vapnik. 1995. *The nature of statistical learning theory*. Springer.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206.

Gisle Ytrestøl, Dan Flickinger, and Stephan Oepen. 2009. Extracting and Annotating Wikipedia Sub-Domains — Towards a New eScience Community Resource. In *Proceedings of the 8th Workshop on Treebanks and Linguistic Theories*, pages 185 – 197.

Gisle Ytrestøl. 2011. Optimistic backtracking: a backtracking overlay for deterministic incremental parsing. In *Proceedings of the ACL 2011 Student Session*, HLT-SS '11, pages 58–63.

Yue Zhang and Stephen Clark. 2011. Shift-reduce ccg parsing. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, pages 683–692.

Yi Zhang and Hans-Ulrich Krieger. 2011. Large-scale corpus-driven pcfg approximation of an hpsg. In *Proceedings of the 12th International Workshop on Parsing Technologies*.

Yi Zhang, Stephan Oepen, and John Carroll. 2007. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Workshop on Parsing Technologies*, pages 48–59. Association for Computational Linguistics.