Discontinuous Data-Oriented Parsing through Mild Context-Sensitivity

MSc Thesis *(Afstudeerscriptie)*

written by

Andreas van Cranenburgh
(born September 21st, 1984 in Amsterdam, The Netherlands)

under the supervision of Prof. dr. ir. Remko Scha, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	Members of the Thesis Committee:
November 3, 2011	Prof. dr. ir. Remko Scha
	Prof. dr. Benedikt Löwe
	Prof. dr. Rens Bod
	Dr. Khalil Sima'an
	Dr. Willem Zuidema

Institute for Logic, Language and Computation

Es gibt eine und nur eine vollständige Analyse des Satzes. — Wittgenstein (TLP 3.25)

There is one and only one complete analysis of the sentence. (my translation)

[...]
Die Sprache verkleidet den Gedanken. Und zwar so, daß man nach der äußeren Form des Kleides, nicht auf die Form des bekleideten Gedankens schließen kann; weil die äußere Form des Kleides nach ganz anderen Zwecken gebildet ist als danach, die Form des Körpers erkennen zu lassen.
Die stillschweigenden Abmachungen zum Verständnis der Umgangssprache sind enorm kompliziert. — Wittgenstein (TLP 4.002)

Language veils thought. And in such a manner that by the form of the cloak one cannot tell the form of the thought which has been cloaked; because the outer form of the cloak was designed with entirely different purposes than revealing the form of the body.
The implict conventions for understanding common language are enormously complicated. (my translation)

Abstract

It has long been argued that incorporating a notion of discontinuity in phrase-structure is desirable, given phenomena such as topicalization and extraposition, and particular features of languages such as cross-serial dependencies in Dutch and the German *Mittelfeld*. Up until recently this was mainly a theoretical topic, but advances in parsing technology have made treebank parsing with discontinuous constituents possible, with favorable results.

We improve on this by applying Data-Oriented Parsing (DOP) to a mildly context-sensitive grammar formalism which allows for discontinuous trees. Decisions during parsing are conditioned on all possible fragments, resulting in improved performance. Despite the fact that both DOP and discontinuity present formidable challenges in terms of computational complexity, the model is reasonably efficient. Our results emulate and surpass the state of the art in discontinuous parsing.

Acknowledgments: I am grateful to Wolfgang Maier for correspondence and making the code of his parser `rparse` available, which was an invaluable resource. Federico Sangati offered advice and, at a crucial moment, convinced me the results might be good enough for a paper. Sandra Kübler provided the Tepacoc corpus. Tikitu de Jager prompted me to take typography seriously. Mark Dufour improved his compiler specifically to optimize my parser. Henk Zeevat persuaded me to pursue the master of logic. Rens Bod introduced me to discontinuous constituents in one of his lectures. Finally, I am most grateful for the continuous moral support from my supervisor.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

NP	Noun Phrase
PP	Prepositional Phrase
AP	Adjective Phrase
VP	Verb Phrase
POS	Part-of-Speech
WSJ	Wall Street Journal
CFG	Context-Free Grammar
PCFG	Probabilistic Context-Free Grammar
HG	Head Grammar
LFG	Lexical-Functional Grammar
FUG	Functional Unification Grammar
DPSG	Discontinuous Phrase-Structure Grammar
RCG	Range Concatenation Grammar
SRCG	Simple Range Concatenation Grammar
LMG	Literal Movement Grammar
TAG	Tree-Adjoining Grammar
CCG	Combinatory Categorial Grammar
MCFG	Multiple Context-Free Grammar
MG	Minimalist Grammar
LCFRS	Linear Context-Free Rewriting Systems
PLCFRS	Probabilistic Linear Context-Free Rewriting Systems
CKY	Cocke-Kasami-Younger
DOP	Data-Oriented Parsing
TSG	Tree-Substitution Grammar
PTSG	Probabilistic Tree-Substitution Grammar
MPD	Most Probable Derivation
MPP	Most Probable Parse
SL-DOP	Simplicity Likelihood DOP
EWE	Equal Weights Estimate
EX	Exact Match
LP	Labeled Precision
LR	Labeled Recall

Chapter 1

Introduction

In which we introduce the problem of parsing as the challenge of coping with structural ambiguity.

The study of language is broadly divided into two fields: philology & linguistics. Philology (love of words) treats texts in their historical and aesthetic dimensions. Linguistics aims to study language in a disinterested and abstract manner. One aim of the latter is to come up with a grammar for a language, or even a 'meta-grammar' encompassing all possible languages. In modern linguistics a grammar is a clear and precise specification of what constitutes a sentence of the language (and consequently, what does not). There is a tradition of formulating formal rules of grammar going back to Pāṇini's work on Sanskrit, more than two thousand years ago. It is, however, a difficult and tedious task, because "all grammars leak" (Sapir, 1921, p. 39)—i.e., they accept ungrammatical sentences, or they reject certain grammatical sentences (typically both).

grammar

With the advent of the computing age, it has become possible to put intricate grammars and theories to the test—provided that they be specified in sufficient detail—and this has vindicated Sapir's sentiment. Statistics was then introduced into the picture, encouraged by its success in speech recognition. The reason for employing statistics is not just the difficulty of demarcating what is and what is not grammatical, but especially the discovery that finding the right analysis for a grammatical sentence involves discriminating between a huge number of possible analyses—a phenomenon called structural ambiguity. To a human, most of these analyses are not apparent, but a formal grammar by itself has no way of pinning down the right one. Although the introduction of statistics was seen as a compromise to the formal (abstract) nature of linguistics by some (such as Chomskians), statistics is not not merely a means to an end motivated from an engineering perspective. Its use can be motivated by a cognitive agenda, as a key part of the answer to the problem of ambiguity. In a nutshell, the whole field of statistical parsing revolves around coping with ambiguity.

ambiguity

A hand-written grammar can be enriched with statistics from a corpus of sentences manually annotated with structures, from which the occurrence frequency of grammatical rules can be inferred. Such probabilistic grammars proved to be much more effective than those relying on manually specified disambiguation criteria. The next step was to abandon manually written grammars altogether, and infer the grammar directly from the structures in the corpus; this is known as *treebank parsing*.

treebank parsing

The structures that are produced by such a parser are not intended for particular applications; they are theoretical objects specifying in detail the form

of an isolated sentence without treating its content. While they can be employed as a source of information in a speech recognition or machine translation system, it turns out that getting this to work well is rather tricky, most likely due to data sparseness; thus the state of the art in those fields still relies largely on shallow training data.

Informally, the structure of a sentence indicates 'who does what to whom, and how, why' More specifically, the structure of a sentence relates words to other words or parts of the sentence until all words or parts have some relation to each other. A collection of such structures, referred to as a corpus or treebank, encodes information such as that 'the unicorn' and 'the unicorn over there' behave similarly; e.g., they can both combine with a verb to form a sentence.

treebank

phrase-structure trees

A concrete instance of such structures are phrase-structure trees, which group words into labeled constituents.[1] A constituent is defined as a unit which can be substituted for other instances of its kind while maintaining grammaticality. A phrase-structure tree is then made up of constituents which are in turn part of higher-level constituents, until the whole sentence belongs to a single constituent. It is usually assumed that every non-root node has a single parent, and that every node dominates a sequence of adjacent nodes or words in the sentence. It has been noted, however, that both constraints appear to be violated in commonly occurring linguistic phenomena.

The first constraint is arguably violated in complex co-ordinations such as 'Mary sees and John recognizes Jane,' in which 'Jane' is the object of two distinct propositions. Instead of duplicating 'Jane,' it would be more elegant to let words appear in multiple constituents of a sentence. This topic is not treated in this work, so unique parents will be assumed hereafter.[2] The second constraint is also violated in many cases, which gives rise to the phenomenon of discontinuity, which will be discussed at some length in the next section. Both constraints have been upheld for reasons of computational and theoretical complexity. But as formalisms, theories and computers developed, treebank parsing with grammars that produce richer structures has come within reach. This forms the point of departure for the present thesis. The aim of this thesis is as follows:

discontinuity

- We will develop a treebank parser with the specific purpose of taking discontinuity into account.
- Specifically, we will extend the Data-Oriented Parsing model to handle discontinuous phrase structures.
- We will evaluate this model on two German corpora which directly encode discontinuity in their annotations.

1 There exists an alternative representation of sentence structure, namely dependency structures, which encode (labeled) relations between words. However, the present work focuses exclusively on phrase-structure trees.

2 Note, however, that range concatenation grammar, which will be introduced in section 3.1, allows for parsing structures with multiple parents in polynomial time, so there is not a strong argument for avoiding them on computational grounds.

Analytical Index

Being a Sketch of the Main Argument

Chapter 2

Discontinuity

Discontinuity is a pervasive phenomenon in language, which suggests that it should be part of theoretical and computational accounts of language. Corpora are presented in which discontinuity is a central feature of the annotation scheme.

Discontinuity refers to a "want of continuity [...] or disunion of parts" (Webster's Revised Unabridged Dictionary, 1913). In the case of constituency trees it refers to the possibility of discontinuous constituents, which have one or more gaps. The gaps have to be filled by the words of other constituents, such that the constituent containing the whole sentence is by definition continuous, since no words are left to interrupt it.

2.1 Linguistic motivation

There are two motivations for discontinuous constituency. The first is a particular phenomenon displayed by some languages which is outside the class of context-free languages. The second is that certain representations might be desirable on theoretical grounds. The strength of the former lies in its independence of linguistic theory, although it relies on this single phenomenon. In contrast, the latter has lots of supporting examples which are arguably more elegantly expressed with discontinuity, while not strictly necessary to produce a grammar which recognizes and produces precisely the strings in the language.

This section deals with discontinuous constituency, but some of these phenomena apply to dependency structures as well, where the stronger notion of *non-projectivity* is employed. A dependency structure is projective iff[3] the head and all nodes it dominates are a contiguous sequence of words. This notion is stronger because not every discontinuous constituent is non-projective when converted to a dependency structure (cf. figure 1), while every non-projective dependency structure is necessarily discontinuous as a constituent structure (trivial).

non-projectivity

A well-known phenomenon which is beyond the power of context-free grammars is called cross-serial dependencies. It has been attested in two languages, namely Dutch (Bresnan et al., 1982) and Swiss-German (Shieber, 1985). Before we move on to cross-serial dependencies, consider the following clauses in English and German:[4]

cross-serial dependencies

3 'iff' stands for 'if and only if.'

4 These examples are adapted from Kolb et al. (2000).

that Charles lets Mary help Peter teach Hans to swim

daß der Karl die Maria dem Peter den Hans schwimmen lehren helfen lässt

The lines indicate dependencies between verbs and their objects; other dependencies are left out, for simplicity's sake. English shows a flat structure where each verb and its argument is directly adjacent, while German shows a nested, palindromic structure. Both of these structures are easily formalized in a context-free grammar using recursive rules.

In Dutch and Swiss-German, the equivalent clauses display cross-serial dependencies:

dat Karel Marie Peter Hans laat helpen leren zwemmen

daß de Karl d'Maria em Peter de Hans laat hälfe lärne schwümmen

These structures cannot be produced by a context-free grammar, if it is to support an arbitrary number of such crossed dependencies. More strongly, Shieber shows that regardless of particular constituent structures, it is not even possible to devise a context-free grammar which accepts exactly the strings of Swiss-German. This is because Swiss-German requires case markings on all NPs, in contrast to Dutch. Without formally treating the cross-serial dependencies, i.e., by generating each verb together with its complement, a CFG would invariably generate sentences with incorrect case markings.

These sentences may seem contrived, but findings from psycholinguistics report that cross-serial dependencies are actually easier to process than nested dependencies (Bach et al., 1986; Vogel et al., 1996). This suggests that the unrestricted recursion of context-free grammars has no cognitive plausibility, while in cognitive processing certain operations can be employed that are beyond context-free, such as detection of duplication. All of this appears to happen in linear time, on average, while most statistical parsers, especially ones with strong stochastic models, require at least cubic time in theory and in practice. Vogel et al. suggest a meta-grammatical approach in which a context-free back-

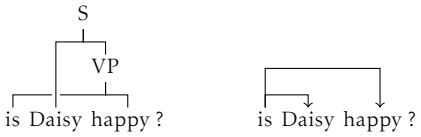

Figure 1: Discontinuity does not imply non-projectivity: a discontinuous constituency structure with a projective dependency structure.

bone is filtered to accept or generate only correct cross-serial dependencies. In essence the approach is for certain rules to expect a duplication, or at least a duplication of a certain feature such as case. The filter is triggered by a special feature on the non-terminals in the CFG rules. In the case of the Swiss-German sentence, the first half is parsed with plain CFG rules, which triggers filtering through features in the VP rules; each of the VPs will expect an NP with a certain case marking in the rest of the sentence.

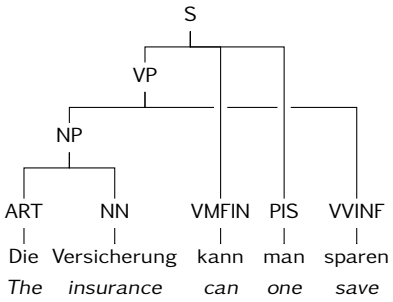

Figure 2: A sentence with topicalization from the Negra corpus.
Translation: *As for the insurance, one can save it.*

extraposition

topicalization

A different class of phenomena deals with constructions that can be readily produced by a context-free grammar, but arguably require richer representations to be treated elegantly. The main example of this is extraposition. Extraposition describes a situation where part of a constituent seems to have moved from where it would be expected.[5] A concrete example of this is the case of topicalization, in which a new topic is introduced by giving it prominence, expressed by moving a constituent to a salient position such as the start of the sentence. Figure 2 shows an example.

extraposed relative clauses

Another case is constituted by extraposed relative clauses. An NP such as 'a man' can be modified by a relative clause such as in 'a man who was PRED'; in principle these are adjacent, to make the relation clear. However, for pragmatic reasons it might be desirable to express the relation of the subject to the verb first:

A man entered the room who was seeking a unicorn

Arguably in this case "A man [...] who was seeking a unicorn" should still be a constituent, because it constitutes a semantic unit—the argument to the verb 'enter' is not just any man, but the one who was seeking a unicorn.

Further examples are given in Bunt (1996) and Dowty (1996), showing that not only NPs but also VPs and APs can be discontinuous, and that discontinuities do not just wrap around other constituents, but readily cross each other, even without cross-serial dependencies:

5 I say 'seems' because in this thesis I assume that discontinuous phenomena are to be described by appropriate representations instead of rich operations on arbitrarily limited representations, such as in the transformational approach.

If the syntactic representation explicitly[6] encodes discontinuities introduced by extraposition and other forms of 'movement,' then it becomes possible to express argument structure directly in the syntactic annotation. This was one of the motivations for the designers of the Negra corpus (Skut et al., 1997) to allow discontinuous constituents. By relaxing the adjacency constraints of traditional phrase-structure trees, it becomes possible to employ constituency to express relationships between heads and *all* of their arguments and adjuncts, instead of arbitrarily attaching them to the nearest possible constituent to satisfy some mathematical stipulation. This means that even if one is not convinced about the reality of discontinuities or movement, there is still a very compelling argument that these representations are better suited for semantic purposes.

argument structure

2.2 Corpora

Most corpora are annotated with traditional (context-free) phrase-structure trees or projective (non-crossing) dependencies. Two German corpora, Negra (Skut et al., 1997) and Tiger (Brants et al., 2002) encode discontinuities directly in their representations. A tree is represented as an arborescence, viz. a directed acyclic graph such that there is exactly one path from its root vertex u to any non-root vertex v. Additionally, there is a total order on its terminals (contrast this with phrase-structure trees, which rely on a partial order among sibling nodes). See figure 5 for an example of a tree encoded with discontinuities.

arborescence

The annotation of both corpora is similar, although Tiger provides some additional information in secondary edges,[7] morphological tags and lemmas. The tree structures in these corpora are flatter (less hierarchical) than those of the Wall Street Journal (wsj) corpus.[8] Because of the discontinuous representations it is possible to employ immediate dominance to express argument structure directly, allowing a tight fit between syntax and semantics. Edge labels encode grammatical functions, including distinctions between complements and adjuncts. A conspicuous difference with the wsj annotation is that there are no finite vps—the main verb and its object are placed directly under the s node. Similarly there is no np node in pps, presumably because it can be inferred. Unary phrasal projections that can be inferred are omitted as well, such that for example a single noun will not get an np node. These annotation decisions are supported with the argument that they eliminate "redundancies" and improve the ergonomics of corpus annotation. This argument strikes me as strange, as

6 There exists a different strategy of encoding discontinuities: the use of special nodel labels such as traces, a method which works with a context-free backbone (Maier and Søgaard, 2008), and employed in corpora such as the Penn treebank (Marcus et al., 1993), and Tüba-D/Z (Telljohann et al., 2004). While this approach can in principle encode exactly the same information, it is arguably more ad-hoc.

7 Secondary edges are used to indicate coordination information not expressible as normal grammatical functions through edges. An example of this is non-constituent coordination when for example a subject np may be shared by multiple verbs.

8 The wsj corpus is part of the Penn treebank (Marcus et al., 1993) and is the most common benchmark in statistical parsing.

this aspect of the annotation can be automated, and the redundancies can be linguistically relevant. As a constituent is often defined in terms of what it can substitute for, it would appear especially relevant for a statistical parser that an NP occurring as subject could also appear in a PP.

Figure 3 shows a graph of the number of discontinuities per tree in the Negra and Tiger corpora. While the majority of trees has no discontinuities at all, approximately thirty percent are annotated with one or more discontinuities per tree. Alternatively the degree of the discontinuities can be compared, which is defined as the maximum number of discontinuities in a single constituent; see table 1. From these figures it can be concluded that discontinuity forms a non-negligible component of the data, and that the discontinuity is not limited to singular gaps. These statistics are based on the original trees from the treebank; later on it will turn out that when binarized (see section 3.4), the number of discontinuities increases. This is because in longer constituents it is possible that there are latent discontinuities which are resolved if all non-terminals are combined at once, but surface when the constituent is built up in multiple steps.

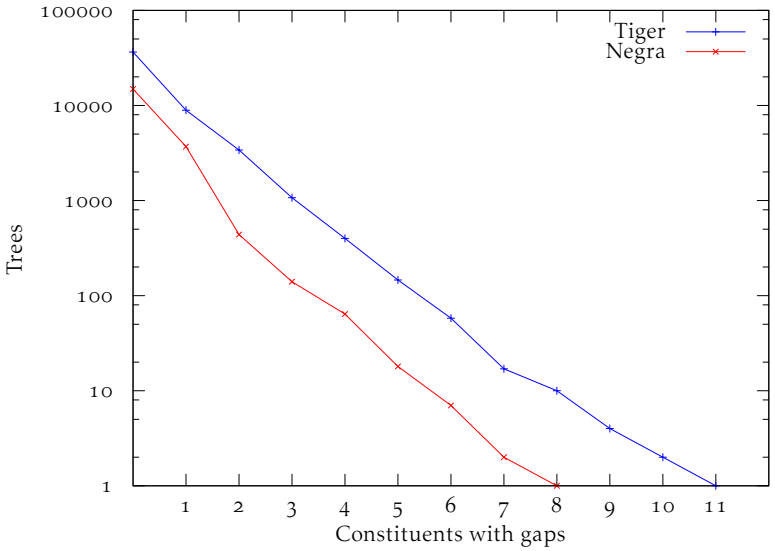

Figure 3: Distribution of trees in the Negra and Tiger corpora with respect to the number of constituents with gaps per tree. The y-axis has a logarithmic scale.

	Negra	Tiger
c = 0	14,924 (72.44%)	36,573 (72.46%)
c = 1	4,991 (24.23%)	12,302 (24.37%)
c = 2	677 (3.30%)	1,585 (3.14%)
c = 3	8 (0.04%)	14 (0.03%)

Table 1: The constituent gap degrees of Tiger/Negra trees, as reported by Maier and Søgaard (2008). The constituent gap degree of a tree is the maximum number of gaps in a constituent.

2.3 Previous work

wrapping

Early work on discontinuity (Yngve, 1960; Harman, 1963) extended the representation of phrase structures with the notion of wrapping a constituent around the next constituent. The idea of wrapping has been further elaborated in the form of Head Grammar (Pollard, 1984), which allows a single gap per constituent to bridge an arbitrary number of constituents; the resulting parser is shown to have a complexity of $\mathcal{O}(n^6)$; compare this to context-free grammar which can be parsed in $\mathcal{O}(n^3)$ time. Formalisms and parsers that allow for arbitrary discontinuity have been reported as well (Johnson, 1985; van Noord, 1991). The first probabilistic discontinuous parser evaluated on a large treebank is reported by Plaehn (2004), who uses a discontinuous phrase-structure grammar (DPSG; Bunt, 1996). A serious problem with these latter formalisms is that they have exponential time complexity.

Fortunately a compromise is possible, which allows arbitrary discontinuity with polynomial time complexity. The crucial difference is that the exponential time grammar formalisms provide a way for derivations to share an unbounded amount of information among its nodes. In the case of DPSG this is because each discontinuous production is constrained to skip particular non-terminals in its gaps, so a production depends on a non-terminal which is not being rewritten, making the formalism context-sensitive. If, however, the derivation trees themselves are context-free, then we can obtain non-context-free parse trees in polynomial time (Vijay-Shanker et al., 1987).

It is instructive to distinguish three orthogonal senses in which a formalism can transcend the different aspects associated with context-free grammars:

1. the form of productions
2. independence assumptions of the probabilistic model
3. objects being rewritten and structure of the resulting derivations

In the strongest sense a formalism can have productions which violate context-freeness in a formal sense. Instead of the original, narrow definition of context-free from the Chomsky-Schützenberger hierarchy (Chomsky and Schützenberger, 1963), Vijay-Shanker et al. (1987) suggest more general criteria to capture the efficiency that comes with context-free rewriting operations. When (a) a formalism's derivation trees can be described by a context-free grammar, and when (b) only a bounded amount of structure can be added or removed by applying productions, recognition will have polynomial time complexity. Examples of formalisms that violate these criteria are DPSG, LFG and FUG (Lexical Functional Grammar and Functional Unification Grammar, respectively; Trautwein, 1995). The weakest sense is when parsing decisions are weighted using non-local statistical dependencies, while the string language remains unchanged; an example of this is Data-Oriented Parsing (DOP, cf. chapter 4). In between these is the possibility of a formalism that produces richer representations, while its productions remain context-free in a more general sense. This is achieved by adhering to conditions (a) and (b). Such a formalism will be introduced in the next section, while in a later section it will be be combined with non-local statistical dependencies as well.

Chapter 3

Linear Context-Free Rewriting Systems

A mildly context-sensitive formalism is able to recognize and parse discontinuity effectively and efficiently. Rules can be read off directly from a treebank and assigned probabilities based on relative frequencies. A chart-parser is presented and methods of binarization are investigated.

mild context-sensitivity

The (somewhat informal) notion of mild context-sensitivity was introduced by Joshi (1985) to capture precisely the amount of generative capacity needed to describe natural languages—as opposed to employing richer frameworks which require ad-hoc constraints to be tractable. Mildly context-sensitive languages are characterized by the following properties:

1. limited crossed dependencies
2. constant growth
3. polynomial time parsing

For a formal description of these properties, refer to e.g., Groenink (1997a). A diverse set of formalisms with these properties has since developed. However, while their structures and operations differ wildly, it has been observed that they share two common properties (Vijay-Shanker et al., 1987; Weir, 1988):

Linear: only a bounded amount of structure can be added or removed by applying productions, i.e., operations are size preserving

Context-free: choices during a derivation are independent of the context in the derivation (where context is anything which is not being rewritten)

Furthermore, it does not matter whether the formalism rewrites strings, tuples, or trees. This led to the introduction of Linear Context-Free Rewriting Systems (lcfrs), which subsumes all formalisms with these properties. Groenink (1997a) states that "[t]he class of mildly context-sensitive languages seems to be most adequately approached by lcfrs." For expository purposes, we will first look at a more powerful formalism.

3.1 The grammar formalism

Range Concatenation Grammar (rcg; Boullier, 1998) is a grammar formalism which aims to cover a wide range of linguistic phenomena while still being efficiently parsable. Rcg started out as a variant of simple Literal Movement Grammar (lmg; Groenink, 1995, 1997b), which has equivalent generative power. A derivation in rcg is presented as a deduction where

non-terminals are treated as predicates holding over ranges of terminals in the input. When all predicates have arity one the formalism is equivalent to a context-free grammar (CFG), but predicates with higher arity can be used to describe an arbitrary number of discontinuities.

As an example, consider the sentence pair "Gatsby is rich" and "is Gatsby rich?". The former will be generated by a typical NP-VP rule, but in a CFG, the latter is usually described by flattening (eliminating) the VP:

$$S \rightarrow VB\ NP\ ADJ$$

In an RCG it can be generated as follows:

$$S(xyz) \rightarrow NP(y)\ VP(x, z)$$
$$VP(x, z) \rightarrow VB(x)\ ADJ(z)$$

ranges

yield function

fan-out

The variables x, y, z denote ranges, which consist of one or more consecutive terminals. Each non-terminal in a rule is a predicate that covers one or more ranges. The left-hand side of the rule defines a yield function specifying how the ranges on the right-hand side compose to form the non-terminal on the left-hand side. The yield function combines the ranges into a new, potentially discontinuous, constituent. Consecutive ranges may be concatenated to form a new range, while non-consecutive ranges must remain discontinuous, resulting in a discontinuous production. Concatenation is expressed on the left-hand side by a sequence of variables not separated by commas. The number of arguments to a predicate will be referred to as its fan-out (in other contexts referred to as arity). Note that the order of non-terminals on the right-hand side is not significant, as everything is determined by the order of the ranges.

A complete grammar for the declarative and polar question can be expressed as a simple recognizer in Prolog:[9]

```
s(ABC)  :- np(B), vp(A,C),    append([A, B, C], ABC).
s(AB)   :- np(A), vp(B),      append([A, B], AB).
vp(AB)  :- v(A),  adj(B),     append([A, B], AB).
vp(A,C) :- v(A),  adj(C).
np(A)   :- n(A).
v([ is ]).
n([ gatsby ]).
adj([ rich ]).
```

Since `ABC` is an arbitrary identifier, we have to append the variables manually. The recognizer can distinguish grammatical and ungrammatical sentences:

```
-? s([gatsby,is,rich]), s([is,gatsby,rich]).
Yes
-? s([rich,gatsby,is]).
No
```

9 It is also possible to encode an RCG in Datalog, a restricted subset of Prolog with efficient bottom-up query evaluation. Datalog does not admit compound structures (such as lists), so rules must be rewritten to employ ranges instead. The interested reader is referred to Kanazawa (2011).

Simple Range Concatenation Grammar (SRCG) is a restricted subset of RCG where each variable must occur at most once on either side (linear), and each variable on the right-hand side must occur on the left-hand side (non-erasing). Together, these two restrictions make for a mildly context-sensitive formalism, which further improves the time complexity, although at the cost of reduced expressiveness. Furthermore, an *ordered* grammar is one where the ranges (equivalently, variables) on each non-terminal on the right-hand side are ordered according to the order in which they appear on the left-hand side. This has the effect that no 'word-order switch rule' can be introduced (i.e., one that converts the order of adjuncts & complements to some canonical order), but it greatly increases efficiency.

linear

non-erasing

ordered

The formalism Linear Context-Free Rewriting Systems is equivalent to Simple Range Concatenation Grammar. This formalism is known to subsume a wide variety of mildly context-sensitive formalisms such as tree-adjoining grammar (TAG), combinatory categorial grammar (CCG), minimalist grammar (MG) and multiple context-free grammar (MCFG), in terms of weak generative equivalence[10] (Vijay-Shanker and Weir, 1994; Kallmeyer, 2010). Even a synchronous context-free grammar can be encoded as an LCFRS, by having one gap in each rule, where the first yield is the source sentence and the second is the target sentence:

$$S(xy, pq) \rightarrow NP(x, p)\ VP(y, q)$$

Additionally, it is possible to parse dependency structures (projective or not) with an LCFRS (Gómez-Rodríguez et al., 2009; Kuhlmann and Satta, 2009). Maier and Kallmeyer (2010) present the first results for grammar-based non-projective dependency parsing.

More strongly, SRCG is a syntactic variant of LCFRS (Kallmeyer, 2010), but arguably the notation of the former is more readable. The notation for LCFRS specifies the production and the composition function separately, which obscures the relation between the non-terminals and their associated variables. See section 3.4 for an example. On the other hand, RCG and SRCG operate on ranges of the sentence, i.e., intervals, while an LCFRS is more general because it can operate on any kind of structure, in our case on tuples of strings. In the rest of this thesis I will employ the SRCG notation for grammar rules while working with the LCFRS formalism.

Formally, an LCFRS is a tuple $G = \langle N, T, V, P, S \rangle$. N is a finite set of non-terminals; every non-terminal symbol has a unique fan-out characterized by a function $dim : N \rightarrow \mathbb{N}$. T and V are disjoint finite sets of terminals and variables. S is the distinguished start symbol with $dim(S) = 1$. P is a finite set of rewrite rules of the form:

$$\begin{aligned} A(\alpha_1, \ldots \alpha_{dim(A)}) \rightarrow & B_1(X_1^1, \ldots, X_{dim(B_1)}^1) \\ & \ldots B_m(X_1^m, \ldots, X_{dim(B_m)}^m) \end{aligned}$$

for $m \geq 0$, where A, B_1, …, $B_m \in N$, each $X_j^i \in V$ for $1 \leq i \leq m$, $1 \leq j \leq dim(A_j)$ and $\alpha_i \in (T \cup V)^*$ for $1 \leq i \leq dim(A_i)$.

10 Weak generative equivalence characterizes the strings a grammar produces; contrast with strong generative equivalence, which considers trees as well.

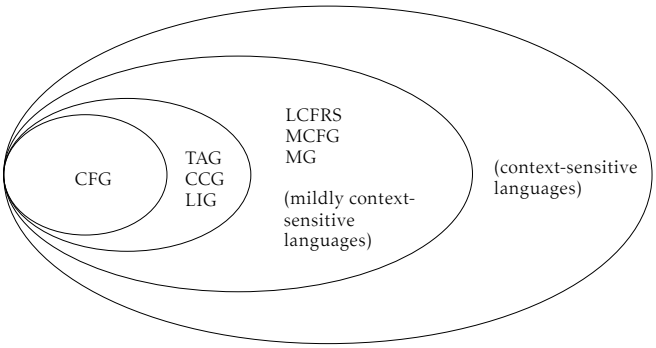

Figure 4: A containment hierarchy of grammar formalisms. Note that while LCFRS, MCFG & MG define an infinite hierarchy of languages, it is not known whether they fully exhaust the class of mildly context-sensitive languages.

instantiated

A rule can be *instantiated* when its variables can be bound to spans such that for each component α_i of the LHS, the concatenation of its terminals and bound variables forms a contiguous span in the input, while the endpoints of each span are non-contiguous.

The parsing complexity of Range Concatenation Grammars is exactly that of PTIME (polynomial time on a deterministic Turing machine). How this relates to context-sensitive grammars of the Chomsky-Schützenberger hierarchy is unknown—although membership in a context-sensitive grammar is known to be PSPACE-complete, it is not known whether an RCG forms a subset of the context-sensitive languages—i.e., the range concatenation languages could include languages outside the class of context-sensitive languages.

LOGCFL

The problem of fixed grammar recognition with a Linear Context-Free Rewriting System is in LOGCFL[11] (Kanazawa, 2007). The parsing complexity for a given grammar is dependent on the maximum fan-out of the grammar, where the fan-out of a grammar is defined as the maximum number of gaps (discontinuities) in the LHS non-terminal of a rule. If φ is the fan-out after binarization, and $|w|$ the number of words, the asymptotic complexity of fixed grammar recognition is $\mathcal{O}(|w|^{3\varphi})$ (Gómez-Rodríguez et al., 2009). A tighter bound for fixed grammar recognition is given by Gildea (2010). Define the

parsing complexity

parsing complexity of a rule as the sum of the fan-outs in the rule (LHS and RHS). Let p be the maximum parsing complexity in the grammar rules; the parsing complexity of the grammar is then $\mathcal{O}(|w|^p)$. Note that in both of these characterizations, the grammar constant $|G|$ has been left out, because it does not dominate asymptotically. In practice, however, it is a very important factor.

Note that binarization can increase the fan-out (cf. section 3.4). If the fan-out is 1, we get a context-free grammar, and hence cubic time, while higher fan-outs define a natural progression of complexities beyond context-free. See figure 4 for a diagram showing the containment hierarchy of complexities of the various formalisms.

11 LOGCFL is the class of problems that can be rewritten in logarithmic space as membership in a context-free language. It is a class of problems which can be efficiently parallelized.

3.2 Grammar extraction

Rules can be read off from (discontinuous) phrase-structure trees in a straightforward manner, similarly to a CFG. The process is slightly more involved, because the yields and their composition functions have to be sorted out. It is spelled out in algorithm 1, which follows Maier and Søgaard (2008). Figure 6 shows the rules extracted from the tree in figure 5.

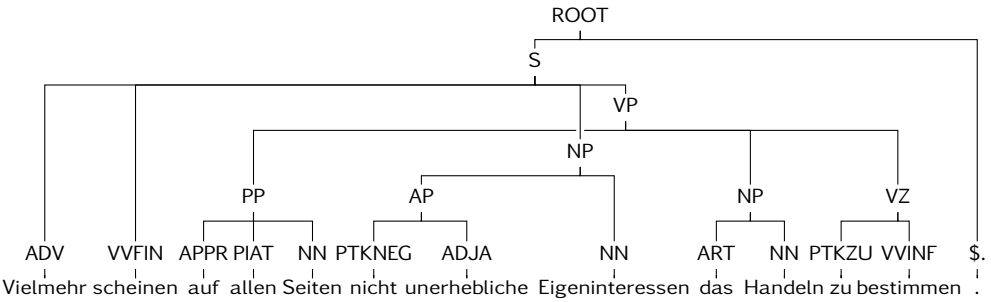

Figure 5: A tree with crossing branches from the Tiger corpus. In this case the subject NP is surrounded by the children of the VP.

To obtain the equivalent of a treebank PCFG for an LCFRS, we can trivially collect the relative frequencies of each production. This maximum likelihood estimate produces a proper probability distribution because just as for a PCFG, there is an isomorphism (one-to-one correspondence) between constituents and productions. And since it is known that relative frequencies form a maximum likelihood estimate for a PCFG, this must also hold for a PLCFRS.

```
1:  for all trees t ∈ T
2:      for all nodes n ∈ t
3:          if lexical?(n)
4:              add pos(word) → ϵ to R, given a tag 'pos' and terminal 'word'
5:          else
6:              k ← number of children of n
7:              for i = 1 to k
8:                  yield ← set of indices of terminals that n_i dominates
9:                  args ← {x | x ∈ yield ∧ x − 1 ∉ yield}
10:                 right-args_i ← args
11:                 B_i ← ⟨ label of n_i, |args| ⟩
12:             left-args ← sorted concatenation of right-args
13:             replace indices in left-args and right-args with new variables
14:             A ← ⟨ label of n, |left-args| ⟩
15:             add to R: A(left-args) → B_1(right-args_1) ... B_k(right-args_k)
```

Algorithm 1: The procedure for extracting LCFRS rules from a treebank. Discontinuous yields are identified by abstracting over terminals and examining ranges of indices instead.

$\text{ROOT}(x_0 x_1) \to \text{S}(x_0)\ \$.(x_1)$
$\text{S}(x_0 x_1 x_2 x_3 x_4) \to \text{ADV}(x_0)\ \text{NP}(x_3)\ \text{VP}_2(x_2, x_4)\ \text{VVFIN}(x_1)$
$\text{NP}(x_0 x_1) \to \text{AP}(x_0)\ \text{NN}(x_1)$
$\text{AP}(x_0 x_1) \to \text{PTKNEG}(x_0)\ \text{ADJA}(x_1)$
$\text{VP}_2(x_0, x_1 x_2) \to \text{PP}(x_0)\ \text{NP}(x_1)\ \text{VZ}(x_2)$
$\text{PP}(x_0 x_1 x_2) \to \text{APPR}(x_0)\ \text{NN}(x_2)\ \text{PIAT}(x_1)$
$\text{NP}(x_0 x_1) \to \text{ART}(x_0)\ \text{NN}(x_1)$
$\text{VZ}(x_0 x_1) \to \text{PTKZU}(x_0)\ \text{VVINF}(x_1)$

$\text{ADV}(\text{Vielmehr}) \to \epsilon$
$\text{VVFIN}(\text{scheinen}) \to \epsilon$
$\text{APPR}(\text{auf}) \to \epsilon$
$\text{PIAT}(\text{allen}) \to \epsilon$
$\text{NN}(\text{Seiten}) \to \epsilon$
$\text{PTKNEG}(\text{nicht}) \to \epsilon$
$\text{ADJA}(\text{unerhebliche}) \to \epsilon$
$\text{NN}(\text{Eigeninteressen}) \to \epsilon$
$\text{ART}(\text{das}) \to \epsilon$
$\text{NN}(\text{Handeln}) \to \epsilon$
$\text{PTKZU}(\text{zu}) \to \epsilon$
$\text{VVINF}(\text{bestimmen}) \to \epsilon$
$\$.(.) \to \epsilon$

Figure 6: Grammar rules as extracted from the tree in figure 5. Note that the VP is marked as having fan-out 2.

3.3 Parsing

Parsing is made difficult by the fact that there is no canonical order in which the search space can be systematically and optimally explored. A typical CKY[12] parser traverses this space using three nested for-loops, enumerating all possible start, end, and split indices over the input string, straightforwardly going from left to right. This is not possible for an LCFRS, because discontinuous constituents skip an arbitrary number of non-terminals. Since the formalism allows an arbitrary number of discontinuities, enumerating all spans exhaustively runs into combinatorial problems.

deductive parsing

Instead an explicit agenda is used. Following the framework of deductive parsing (Shieber et al., 1995; Nederhof, 2003), propositions are taken from the agenda and all possible deductions are put on the agenda, until the goal-proposition, the top node covering the whole input string, is reached or the agenda is empty. The propositions consist of instantiated grammar rules, which are rules in which the variables have been bound to tuples of strings in the input string.

We will employ an extended version of the agenda-based chart-parser for LCFRS by Kallmeyer and Maier (2010). The algorithm is Knuth's generalization of Dijkstra's shortest path algorithm to the case of hypergraphs, where the shortest path is the Viterbi derivation and the hypergraph is the chart defining possible derivations. As with CKY, this parser requires that rules are binarized; for efficiency, we also restrict the parser to ordered rules. The search space of a PCFG can be explored systematically from left to right with constituents of increasing size; this is what makes typical CKY parsers efficient. Unfortunately this approach does not translate to LCFRS because discontinuous constituents

12 The Cocke-Kasami-Younger algorithm (Younger, 1967) is a bottom-up tabular parsing algorithm that relies on productions being in binary form. Although the original formulation is specifically about context-free languages and requires the grammar to be in the more restricted Chomsky Normal Form (CNF), this CNF conversion is not strictly necessary and in fact detrimental to parsing efficiency as it may produce an exponential increase in grammar size (Lange, 2009).

```
initialize agenda 𝒜 with POS tags
WHILE 𝒜 ≠ ∅
    ⟨I, x⟩ ← item with lowest score on agenda
    add ⟨I, x⟩ to 𝒞 and ℱ
    FOR ALL ⟨I′, y⟩ deduced from {⟨I, J⟩, ⟨J, I⟩, ⟨I⟩ | J ∈ 𝒞}
        IF I′ ∉ 𝒜 ∪ 𝒞
            enqueue ⟨I′, y⟩ in 𝒜
        ELSE IF I′ ∈ 𝒜 ∧ y < score for I′ in 𝒜
            add I′ with old score to ℱ
            update weight of I′ in 𝒜 to y
        ELSE
            add ⟨I′, y⟩ to ℱ
```

Algorithm 2: An agenda-based chart-parser for LCFRS.

can cover any subsequence of the input. For this reason an explicit agenda has to be used, ordered by inside probability; alternatively the agenda can employ figures of merit or A* heuristics to order the agenda, but this is not explored in this work.

The pseudo-code for the parser is given in algorithm 2. An item I is defined by a label (e.g., NP) and one or more spans, which can be represented as a bit vector to allow efficient set operations (e.g., 1001 to denote the first and last word in a sentence of four words). Items are associated with weights and backpointers to other items to form edges which make up complete derivations, but when determining whether an item is already on the agenda or in the chart, only the label and the spans are considered. The data structures $\mathcal{A}$ and $\mathcal{C}$ are the agenda and the chart of Viterbi items, as in Kallmeyer and Maier (2010). An additional data structure, $\mathcal{F}$, keeps track of all edges encountered, including suboptimal ones. This structure implicitly defines a parse forest, from which derivations can be sampled or extracted in descending order of probability. Another important difference is that the algorithm does not terminate upon reaching the first goal node, but only when the agenda has been completely exhausted.

item

edges

The algorithm is deceptively simple. Most of the work is in producing all items that can be deduced from a given item and an item in the chart. This involves iterating over all grammar rules with matching labels, and verifying whether the yields are compatible according to the rule. This can be optimized by representing yields as bit vectors and first verifying that the two yields do not overlap. The next step is verifying that each variable in the rule corresponds to a component of one of the yields of the RHS non-terminals, and that these components are adjacent if and only if they are concatenated in the rule. Rules that can be instantiated are given a score that is the sum of the weights of their RHS (assuming log probabilities).

The parser just presented relies on having rules in binarized form, because when an item is taken from the agenda it needs to derive all possible new items from it. If the grammar rules were not binarized and have the maximum rank n (i.e., the maximum number of non-terminals on a right-hand side is n), then obtaining the set of new items involves evaluating the Cartesian products from 1 to n items against all rules. Explicitly enumerating this Cartesian product is clearly

not feasible, so a parser for unbinarized grammars would have to work with incomplete edges. This has been done, using thread automata (Villemonte de la Clergerie, 2002), and in the form of an incremental Earley parser (Kallmeyer and Maier, 2009). However, it appears that it entails a prohibitive amount of bookkeeping in the context of treebank parsing.

When rules are binarized, the current item taken from the agenda only has to be paired with all possible items in the chart for a given rule (unary productions can be allowed as well, without further ado). For efficiency the Viterbi chart is implemented as a mapping from labels to spans with scores. An outer loop treats all rules compatible with the item from the agenda according to its label. An inner loop evaluates all items from the chart matching the rule under consideration, again indexed on the label, this time prescribed by the rule. The final step is then to see whether the spans of the two items combine in the manner prescribed by the yield function of the rule. This is implemented as a loop iterating over the spans and the yield function until either an incompatibility is found, or the end of the spans is reached, indicating success.

This strategy is much less efficient than a CKY parser for a CFG, because there the only items compatible with an item under consideration are the ones whose span is directly adjacent, which translates to a simple array look up. For an LCFRS an arbitrary number of spans have to match in just the way prescribed, such that a brute force search for compatible spans is necessary, aside from optimizing for matching labels.

If discontinuity is suitably limited (preferably to singular gaps), it is feasible to use a geometric data structure to represent chart items as n-dimensional points (Waxmonsky and Melamed, 2006); candidate items can then be retrieved using a range query. However, such a data structure quickly becomes unwieldy with higher dimensions or when the chart is densely packed. This strategy is therefore not pursued in this work.

3.4 Binarization

The binarization of LCFRS grammars is a considerable topic in itself, first because it has a large influence on parsing complexity, and second because the freedom of reordering the right-hand side results in $\mathcal{O}(n!)$ binarization strategies.

Binarization has an influence on the parsing complexity because certain binarizations will increase the fan-out (introduce discontinuities). Another reason, which also holds for a CFG, is that the order of binarization affects the number of useless edges that will be produced (edges not part of a complete derivation). For example an adverb can be part of many kinds of constituents, because it can modify verbs, adjectives and other adverbs. It would therefore not be a good idea to binarize productions with adverbs such that the initial trigger for a rule is the presence of an adverb, because this could trigger a needless search for a VP which could have been avoided if the verb had been the trigger. To counter this, binarizations can be made with respect to the head of a constituent, which is the linguistically most important part of a constituent. The first reason, however, affects not just the size of the grammar (the grammar constant) or the resulting chart, but the asymptotic complexity, which could be

a more serious problem.

Consider a production $X(pqrs) \rightarrow A(p,r)\ B(q)\ C(s)$. This is a continuous production (fan-out 1), but one of its non-terminals on the right-hand side has one gap. If we were to apply a right binarization, the non-terminals B and C would be combined first, but the variables q and s are not consecutive, so this binarization introduces a discontinuity in the resulting grammar. If we apply a left binarization, however, the non-terminals A and B will be combined, whose variables concatenate to pqr, after which C can be introduced, maintaining the continuity of the original production. For this reason it is interesting to see whether we can minimize the number of discontinuities introduced through binarization.

Gildea (2010) gives an algorithm to find the minimal binarization of a rule in terms of a given function, which has exponential running time. Parsing complexity can then by minimized by using a function that returns a tuple of the number of the number of variables in a production and the fan-out of the left-hand side; the former minimizes the parsing complexity and the latter breaks ties in a way that minimizes the fan-out. The algorithm proceeds by taking the lowest scoring item from the agenda (a priority queue) and combining it with another, resulting in a new production with an artificial label, until a production covering the whole right-hand side is found. The ordering of the agenda guarantees that the minimal binarization is found first.

```
1:  FOR ALL children c of t
2:      workingset ← workingset ∪ {c}
3:      enqueue c in A
4:  WHILE A ≠ ∅
5:      p ← minimum from A
6:      IF p ∉ workingset
7:          CONTINUE
8:      ELSE IF nonterms(p) = children of t
9:          RETURN p
10:     FOR ALL p1 ∈ workingset such that
                  nonterms(p) ∩ nonterms(p1) = ∅
11:         p2 ← newprod(p, p1)
12:         IF ¬∃p3 ∈ workingset such that
            nonterms(p2) = nonterms(p3) ∧ score(p3) < score(p2)
13:             workingset ← {p2} ∪ {p3 | p3 ∈ workingset ∧
                                        nonterms(p2) ≠ nonterms(p3)}
14:             enqueue p2 in A
```

Algorithm 3: Finding the optimal binarization of a constituent t according to a given scoring function.

See algorithm 3 for a specification. The function 'newprod' takes two non-terminals and returns a new production with an artificial label, covering the union of their yields. The score for a production is the maximum of the scores for the productions leading up to a production and of the production itself.

The algorithm is an adaptation of Gildea (2010). First, when an item is popped from the agenda, it confirms whether the item is still in the working set (it might have been superseded meanwhile; an alternative would be to

remove inferior items from the agenda immediately, but that is a more expensive operation). Second, when producing new candidate productions, only items without overlap should be evaluated. Lastly, when evaluating a new production, it should be added not just when it is better than a previous item covering the same non-terminals, but also when no such item had been found thus far.

Gildea (2010) further gives a production which demonstrates that minimizing fan-out and parsing complexity can be mutually exclusive. Gildea presents a production with an optimal binarization according to parsing complexity that differs from the binarization with minimal fan-out. The fan-out is simply the number of spans on the left-hand side (gaps plus one), while the parsing complexity is defined as the total number of variables in the rule (the sum of the fan-outs of the left and right-hand side). There is a binarization with an optimal parsing complexity of 14 and a fan-out of 6, versus a binarization with parsing complexity of 15 and fan-out of 5. The production in question is as follows in LCFRS notation:

$$\begin{aligned} &A \to g(B1, B2, B3, B4) \\ &g(\langle x_{1,1}, x_{1,2}\rangle, \langle x_{2,1}, x_{2,2}, x_{2,3}\rangle, \langle x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}\rangle, \langle x_{4,1}, x_{4,2}, x_{4,3}\rangle) \\ &\quad = \langle x_{4,1}x_{3,1}, x_{2,1}, x_{4,2}x_{1,1}x_{2,2}x_{4,3}x_{3,2}x_{2,3}x_{3,3}x_{2,3}x_{3,3}, x_{1,2}x_{3,4}, x_{3,5}\rangle \end{aligned}$$

Converted to SRCG notation, this reads:

$$\begin{aligned} &\mathrm{A}(x_0x_1, x_2, x_3x_4x_5x_6x_7x_8x_9, x_{10}x_{11}, x_{12}) \\ &\quad \to B4(x_0, x_3, x_6)\ B1(x_4, x_{10})\ B2(x_2, x_5, x_8)\ B3(x_1, x_7, x_9, x_{11}, x_{12}) \end{aligned}$$

If we optimize first for complexity and then for fan-out, we get these productions (with complexity and fan-out listed in the first two columns):

$$\begin{array}{cc} 14 & 5 \\ 14 & 6 \\ 14 & 6 \end{array} \quad \begin{aligned} &A(x_0x_1, x_2, x_3x_4x_5x_6, x_7, x_8) \\ &\qquad \to B4(x_0, x_3, x_5)\ A_{B1,B2,B3}(x_1, x_2, x_4, x_6, x_7, x_8) \\ &A_{B1,B2,B3}(x_0, x_1, x_2x_3, x_4, x_5x_6, x_7) \\ &\qquad \to A_{B2,B3}(x_0, x_1, x_3, x_4, x_6, x_7)\ B1(x_2, x_5) \\ &A_{B2,B3}(x_0, x_1, x_2, x_3x_4x_5, x_6, x_7) \\ &\qquad \to B3(x_0, x_3, x_5, x_6, x_7)\ B2(x_1, x_2, x_4) \end{aligned}$$

If we optimize first for fan-out and then for complexity, we get these productions:

$$\begin{array}{cc} 15 & 5 \\ 12 & 5 \\ 9 & 4 \end{array} \quad \begin{aligned} &A(x_0x_1, x_2, x_3x_4x_5x_6, x_7x_8, x_9) \\ &\qquad \to A_{B1,B2,B4}(x_0, x_2, x_3, x_5, x_7)\ B3(x_1, x_4, x_6, x_8, x_9) \\ &A_{B1,B2,B4}(x_0, x_1, x_2x_3x_4, x_5, x_6) \\ &\qquad \to A_{B1,B4}(x_0, x_2, x_4, x_6)\ B2(x_1, x_3, x_5) \\ &A_{B1,B4}(x_0, x_1x_2, x_3, x_4) \to B4(x_0, x_1, x_3)\ B1(x_2, x_4) \end{aligned}$$

See figure 7 for the production and its minimal binarizations rendered as tree structures.

These results on optimal parsing complexity are not the whole story, however. There is another motivation for binarization: a treebank grammar contains many unique n-ary productions, and therefore it is likely that it lacks many unseen productions. Breaking them up into re-usable binary productions is a reasonable way to smooth them (Sima'an, 2000).

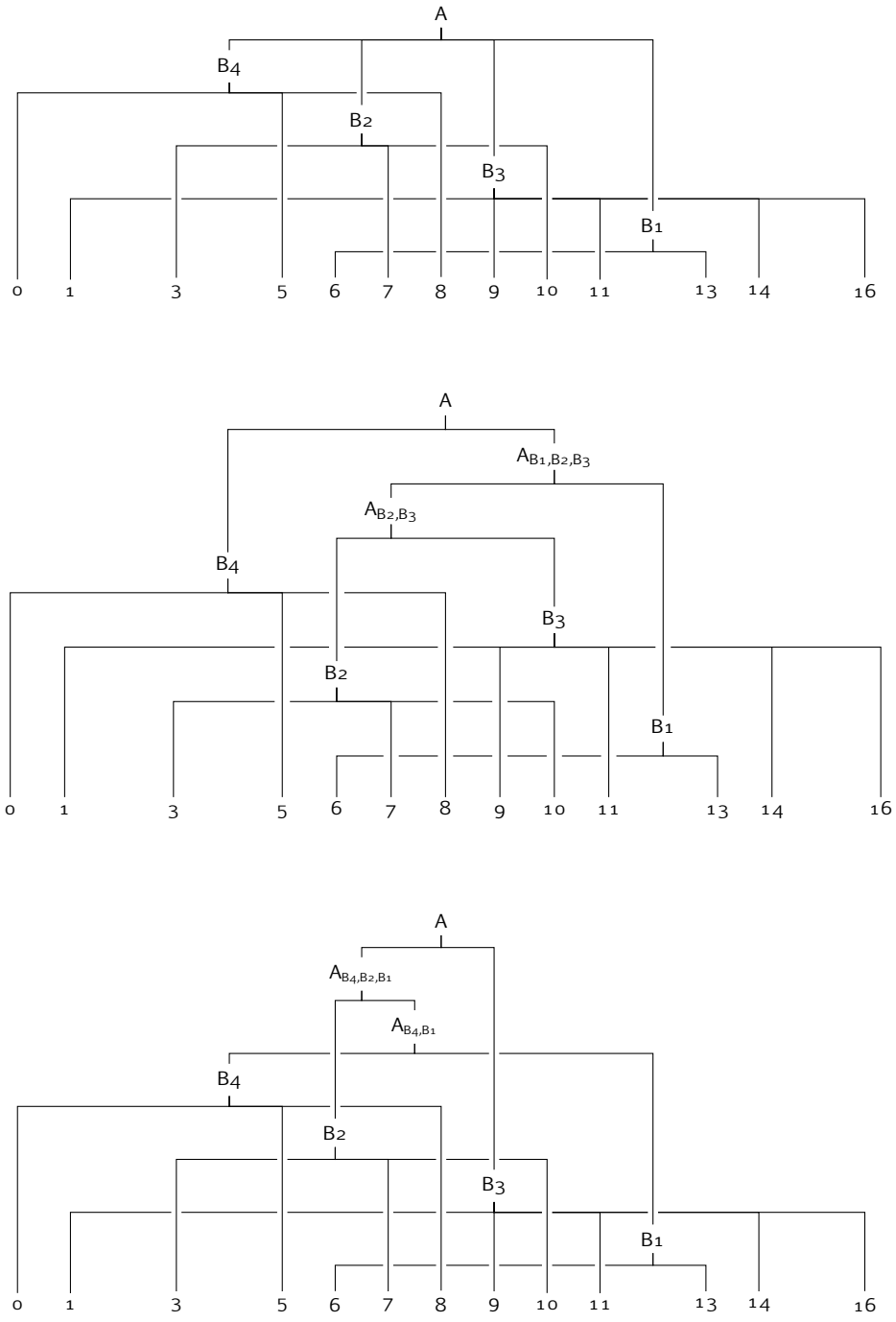

Figure 7: From top to bottom: the production from Gildea (2010) for which minimal complexity and fan-out are exclusive; an optimal binarization that minimizes complexity, which must sacrifice minimal fan-out; an optimal binarization that minimizes fan-out, which must sacrifice minimal complexity.

The artificial labels introduced by the optimal binarization are unique such that they are only applicable when the particular set of non-terminals on the right-hand side of the original production is present. When instead labels are introduced with a limited amount of horizontal context—for example the previous sibling—two binarized productions can be combined to recognize a new sequence of non-terminals (Klein and Manning, 2003a). On the one hand binarizations can be restricted by markovization in the vertical direction by adding parent annotation. A setting of v=1 corresponds to a default binarization equivalent to the original grammar, while v=2 adds a grandparent as well; conversely v=0 gives a grammar that is less restrictive than the original. On the other hand we can smooth in the horizontal direction instead of using the default h=∞. With a PCFG on Negra, the best results are obtained with v=2, h=1 (Rafferty and Manning, 2008).

horizontal context

parent annotation

lexical heads

Additionally, when information about lexical heads is present in the corpus this can be exploited by making sure the head will be parsed first (Collins, 1999). Negra and Tiger contain annotations marking certain pre-terminals as head, but it is useful to ensure that all constituents contain exactly one head. This has been realized by the use of heuristic rules based on those employed by the Stanford parser (Klein and Manning, 2003b).

Unfortunately, incorporating this information in a search for an optimal parsing strategy results in an NP-hard problem[13] (Crescenzi et al., 2011). A simpler approach is to employ a simple head-driven strategy without regard for efficiency, following Maier and Kallmeyer (2010). The right-hand side is ordered by linear precedence of its yields, after which everything after the head is reversed and moved to the front, so that the head will be the final element. This effectively produces a head-outward binarization.[14] The result is that when parsing bottom-up, the head is covered first (incomplete constituents are restricted to ones covering a head). To increase coverage, the first and last element of the right-hand side are binarized to unary productions; this allows stringing together parts of constituents not only when they match somewhere in the middle, but at the boundaries as well. Figure 8 illustrates the binarization for a single constituent, while figure 9 shows the result of binarization on a full tree.

Note that binarization is sometimes presented as a grammar transformation, but considering it as a transformation on trees has important advantages. For one, parent annotation becomes trivial: one simply adds parents to each label in the trees, after which rules are read off. With grammar rules one would have to keep track of the contexts in which a rule was used, and introduce a new copy of the rule for each distinct parent. Another reason is probabilities. When binarizing trees, no special attention has to be given to probabilities or frequencies; reading of a PCFG or PLCFRS automatically gives the right frequencies, also for artificial nodes introduced by the binarization procedure. This distinction has direct ramifications. In the work of Kallmeyer and Maier (2010), binarization operates on a grammar, and as a consequence of the requirement in LCFRS that every non-terminal has a unique fanout, a continuous and discontinuous NP are

13 An NP-hard problem is a problem that is at least as hard as any problem solvable in non-deterministic polynomial (NP) time. In practice, this means it is not efficiently solvable, irrespective of computing hardware, when the problem size is sufficiently large.

14 Reversing the part after the head is equivalent with switching the direction of binarization from right to left in a CFG.

labeled differently, even before binarization. The result is that their markovization is more strict: a discontinuous NP below an S node will introduce the artificial node S_{NP_2}, which selects a discontinuous NP specifically. Furthermore, this means that these fan-out markers are introduced twice: before and after binarization. In our system the trees from the treebank are binarized directly without adding such fan-out markers; only after binarization the grammar is read off and such markers are added at which point they are strictly necessary.

To evaluate the actual effects on efficiency, different binarizations need to be compared empirically on real data. To this end, a grammar was read off from sentences of up to 25 words in the first 7200 sentences of the Tiger treebank, and the next 100 sentences of up to 25 words were parsed using this PLCFRS, using the different binarization schemes. First with a right branching, left-to-right binarization, and second with the minimal binarization according to parsing complexity and fan-out. The last two binarizations are head-driven and markovized—the first straightforwardly from left-to-right, the latter optimized for minimal parsing complexity. With markovization we are forced to add a level of parent annotation to tame the increase in productivity caused by h=1.

Table 2 shows the results. Despite the optimal binarizations being exponential and NP-hard, they can be computed relatively quickly on this data set. There is no improvement on fan-out or parsing complexity, so no asymptotic improvement in parsing. On the other hand, the values of fan-out and parsing complexity only reflect the worst-case production in the grammars, so the optimal binarizations might still have improved the average case for them. It is strange that the optimal non-markovized binarization is slower than the right branching binarization, while it has better accuracy. This has to be due to the fact that they introduce different kinds of artificial nodes. Despite that, the optimal head-driven binarization does give a considerable speedup with respect to the non-optimal head-driven binarization, but this is also paid for in terms of accuracy, so it could be a side-effect of the lower number of labels, instead of optimality with respect to parsing complexity.

I have also been able to apply optimal binarization to all of the 30 word sentences of Negra. This took just 179.91 seconds, and it did not result in a lower fan-out with respect to the default right-factored binarization, while the parsing complexity was 14 compared to 15 for a right-factored binarization. As an optimization, continuous constituents[15] were detected and given a default right-factored binarization instead. The fact that only a modest improvement in parsing complexity could be gained on a sizable amount of real-world data does suggest that the preoccupation with binarization and its effects on complexity is unwarranted. When it is not very difficult to obtain empirical results, as in this case,[16] it should be *de rigueur* to consider these data instead of relying completely on formal proofs of complexity, which can be highly misleading. This is because complexity proofs consider the general case, which may not coincide with the 'interesting' case: constant factors can be either too low or too high for the optimistic or pessimistic conclusions to apply. Despite this, formal proofs tend to be perceived as 'the final word'; empirical evaluations are

15 A constituent is continuous iff it has a fan-out of 1 and in addition its children have a fan-out of 1.

16 End-to-end testing with treebank parsing results is rather involved, of course, but the algorithm for reading off LCFRS rules (Maier and Søgaard, 2008) is very simple and would have demonstrated that the increase in fan-out is modest. Specifically, the observed increase in fan-out is linear.

necessary to correct such perceptions.

	right branching	optimal	head-driven	optimal head-driven
markovization	v=1, h=∞	v=1, h=∞	v=1, h=2	v=1, h=2
fan-out	6	6	6	6
complexity	12	12	12	12
labels	6104	5737	3027	2193
clauses	29373	29732	26513	25491
time to binarize	0.81 s	20.59 s	1.67 s	17.75 s
time to parse	33.25 s	42.21 s	167.12 s	45.61 s
coverage	90.0%	94.0%	97.0%	95.0%
F_1 score	65.89%	68.60%	71.05%	68.98%

Table 2: The effect of binarization strategies on parsing efficiency, with sentences of up to 25 words. For an explanation of the F_1 scores refer to section 7.1.

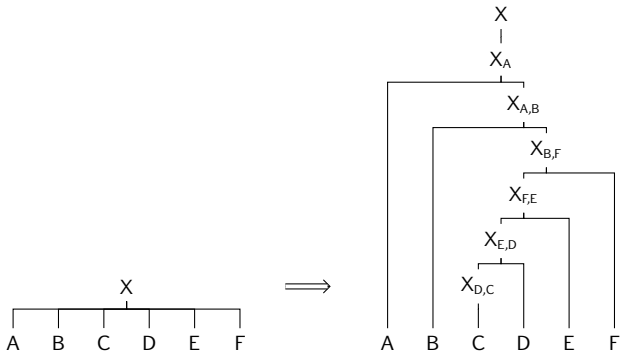

Figure 8: A head-outward binarization with h=2 v=1 markovization; C is the head node.

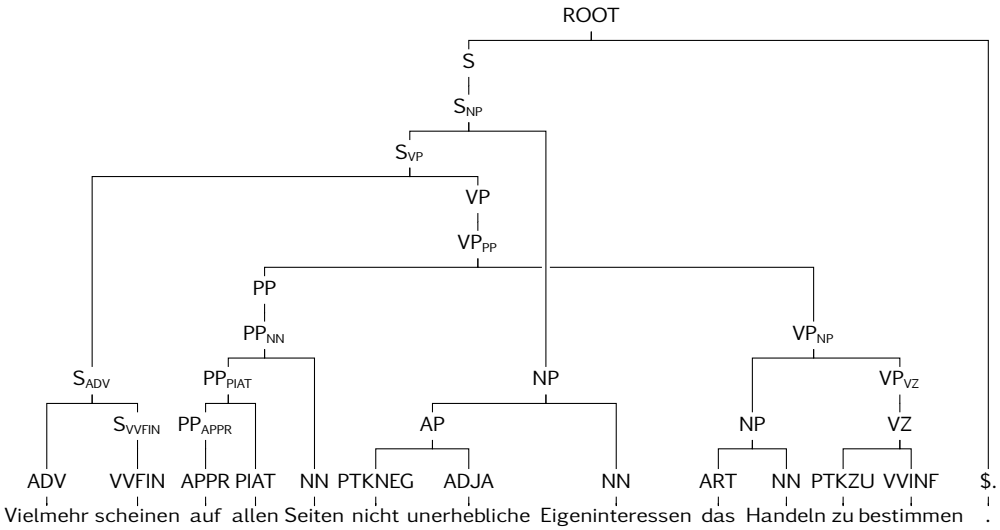

Figure 9: A right-factored head-driven binarization of the tree in figure 5, with markovization of v=1, h=1. Before binarization, the children are reordered so that the head is the last non-terminal. After binarization, the children of each node are sorted according to the index of their first terminal, for aesthetic purposes.

Chapter 4

Data-Oriented Parsing

We turn to data-oriented parsing for a conceptually simple and general account of the interpretation of new utterances given a corpus of previous utterances. Methods for extending existing DOP *implementations with discontinuity are presented.*

Most generative work in statistical parsing is based on Probabilistic Context-Free Grammars (PCFG) and extensions thereof. A PCFG consists of the minimal decomposition of the training data, viz., the individual productions, together with their relative frequencies. This provides a simple, efficient and reasonably accurate model, given simple refinements to overcome the strong independence assumptions made by the PCFG model. These assumptions can be broken down as follows (taken from Manning and Schütze, 1999, ch. 11):

Place invariance: The probability of a subtree does not depend on where in the string the words it dominates are [...]

Context-free: The probability of a subtree does not depend on words not dominated by the subtree. [...]

Ancestor-free: The probability of a subtree does not depend on nodes in the derivation outside the subtree. [...]

Examples of refinements to weaken these assumptions are including parent categories in the labels and introducing linguistic distinctions not present in the original annotation (Klein and Manning, 2003a). Higher accuracy can be obtained with further refinements, be they manual or automatic, as well as different estimation methods. Examples are latent annotations (Matsuzaki et al., 2005), discriminative re-ranking (Collins and Koo, 2005), and Dirichlet processes (Liang et al., 2007). This leads to models with impressive accuracy and efficiency, but the focus is on cranking up the score in any way possible, which leads to complicated procedures for something which is conceptually very simple: what parts does a sentence have and how do they relate to each other. It is by no means obvious that such methods could ever be given a linguistic or cognitive interpretation. Furthermore, it appears to me that these sophisticated machine learning applications serve to extract the most out of the limited amount of available data, while I think that actual language users face the opposite situation: an abundance of data which would preferably be processed with simple methods—in particular data would be processed extensively rather than intensively.

Data-Oriented Parsing (DOP, Scha 1990) addresses such concerns by departing from the assumption that humans process a sentence by preferring

structures based on previously heard fragments. Two kinds of preferences can be distinguished:

A MEMORY BIAS: "[T]he number of constructions that is used to re-construct the sentence in order to recognize it must be as small as possible." (Scha, 1990).
A PROBABILISTIC BIAS: "More frequent constructions are to be preferred above less frequent ones." (Scha, 1990).

The definition of a DOP model can be broken down into four parts:

FRAGMENTS: what are the units on which the model operates?
OPERATIONS: what operations can be performed to combine or alter fragments?
ESTIMATION: how will the probability of performing operations on particular fragments be determined?
DISAMBIGUATION: how will the most appropriate parse tree be selected among candidates?

This work contributes a new definition of the first part, and combines it with existing answers to the other parts.

4.1 The probabilistic framework

The first instantiation of DOP is DOP1 (Bod, 1992), which is a Probabilistic Tree-Substitution Grammar (PTSG).[17] A tree-substitution grammar can be seen as a context-free grammar which rewrites trees instead of strings. It is defined by a set of elementary trees and a substitution operation which combines these trees until they form a derivation of a complete sentence.

A derivation is defined as a sequence of elementary trees combined through left-most substitution. Left-most substitution is defined for any two trees t_1 and t_2, such that t_1 has a frontier node labeled X and $\text{root}(t_2) = X$; the result of $t_1 \circ t_2$ is a new tree where t_2 is substituted for the first frontier node labeled X in t_1. The probability of a derivation is the product of the weights of its elementary trees.

derivation

substitution

In general, a tree-substitution grammar is not equivalent to a context-free grammar. However, in the case of DOP1, the set of elementary trees is such that their generative powers are in fact identical. Specifically, all fragments are built up out of CFG rules, and all CFG rules are themselves fragments of depth 1, so the generative power must coincide.

Although the generative power of the underlying grammar is identical to a context-free grammar, probabilities are estimated not just on the basis of the frequencies of CFG rules, but by considering all connected fragments of the trees in the training[18] data. More specifically, a fragment of a tree is a tree of depth ≥ 1, such that every node has a corresponding node in the original tree, and has either no children, or the same children as in the original tree. When a node in a fragment has zero children, it is called a frontier node. Frontier nodes are the substitution sites of fragments, and correspond to open slots in

fragment

frontier node

17 Sometimes the name Stochastic Tree Substitution Grammar (STSG) is used, but 'probabilistic' avoids confusion with synchronous grammars.

18 Technically, a DOP model is not trained, because its probabilities are directly derived from data. I will maintain the terminology of training and testing to distinguish the part of the data which the parser has at its disposal and the strictly separated part which the model is evaluated on.

constructions. Figure 10 shows the bag of fragments extracted from a sentence; figure 11 shows a DOP1 derivation with these fragments.

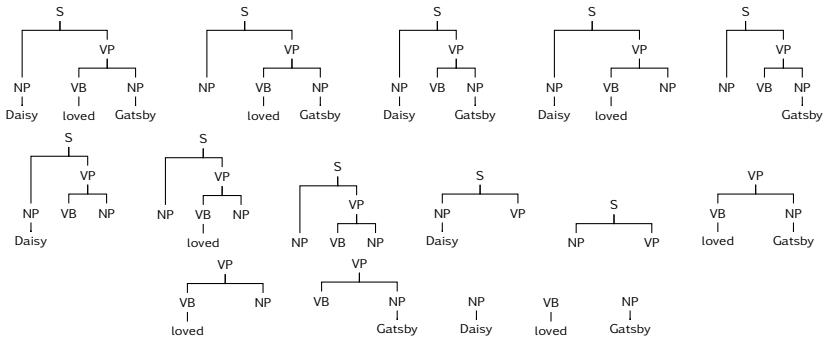

Figure 10: The fragments as extracted from "Daisy loved Gatsby."

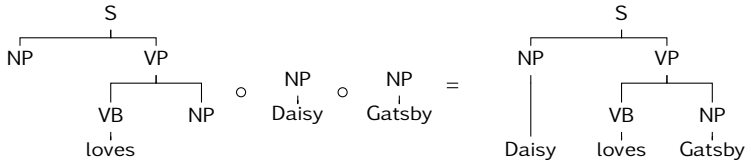

Figure 11: A DOP1 derivation. Note that "Daisy" becomes the subject, because fragments are combined with left-most substitution.

Since these fragments can cover an arbitrary number of terminals & non-terminals, the independence assumptions made in parsing are much weaker, and much more information from the training data is exploited during parsing. It is tempting to conclude that DOP models *all* possible statistical dependencies, because DOP uses *all* fragments. This is not true, however, for several reasons. For one, there could be more general definitions of what constitutes a fragment; e.g., relaxing the assumption that a 'fragment' must be a connected subset. Furthermore, certain statistical regularities cannot be captured using frequencies of fragments, such as Markov processes or phenomena that violate the place-invariance assumption. Lastly, and most importantly, while DOP1 is strong on modeling structural relations, it is not sensitive to lexical dependencies (Sima'an, 2000). The DOP1 model does weaken both the context-free and the ancestor-free assumptions made by PCFG models, through its probabilities of larger fragments. Additionally, there can be multiple sequences of fragments which cover the sentence, because there will be overlap. This so-called spurious ambiguity should be exploited because it allows a more fine-grained comparison of possible analyses for a sentence.

This suggests two fundamental methods of disambiguation based on frequencies: the most probable derivation (MPD), and the most probable parse (MPP). The former maximizes the probability of an individual derivation (one sequence of fragments leading to a complete analysis). The latter maximizes the

sum of derivations leading to the same analysis, i.e., we choose t to maximize

$$P(t) = \sum_{d \in D(t)} \prod_{f \in d} p(f)$$

where $D(t)$ is the set of possible derivations of t. Bod (1995) cites a score of 65% when using the MPD, and 96% with the MPP. Unfortunately, this step of calculating not just the MPD but the MPP is often neglected in DOP-inspired tree-substitution grammars (O'Donnell et al., 2009; Cohn et al., 2009; Post and Gildea, 2009), in pursuit of a more economical or efficient model. I believe that this kind of parsimony has unfortunate theoretical consequences (aside from the obvious decrease in accuracy), given the results on the importance of frequency in grammaticalization and the formation of idioms (Bybee, 2007). Since it appears that arbitrary units can partake in this process, all fragments & frequencies must be available. This leaves the door open to topics such as language change & acquisition, instead of modeling a parsimonious but synchronic snapshot provided by the sample that is the training corpus.

It has been shown that the problem of finding the most probable parse is NP-hard (Sima'an, 2002). Consider that there is an exponential number of fragments for each tree, hence a potentially exponential number of derivations, and it follows that the exact best parse cannot be identified in polynomial time, in the general case. However, this is not a problem in practice, as there are methods to approximate the best parse effectively, using any number of random or best derivations.

There is also a non-probabilistic method of disambiguation, the shortest derivation (Bod, 2000). The objective is to minimize the length of the derivation. In order to break ties of multiple shortest derivations, some additional criterion is necessary. An example of this is the most probable shortest derivation (MPSD), which breaks ties by looking at derivation probabilities.

shortest derivation

4.2 Estimators

In DOP1 the probability of a fragment f from the set of all fragments $\mathcal{F}$ being substituted for a frontier node with label $\text{root}(f)$ in a derivation is given by its relative frequency:

$$\frac{\text{freq}(f)}{\sum_{f' \in \mathcal{F}'} \text{freq}(f')} \quad \text{where } \mathcal{F}' = \{\, f' \in \mathcal{F} \mid \text{root}(f') = \text{root}(f) \,\}$$

Johnson (2002) has shown that this estimator is *biased* and *inconsistent*. The bias of an estimator is the difference between the estimator's expected value[19] and the true value of the parameter being estimated. For a DOP estimator, a sample consists of a sequence of parse trees (a training corpus) sampled from the true distribution; the parameter being estimated is a distribution over parse trees. A DOP estimator is biased iff there is a distribution such that the estimator's expected probability distribution given all training corpora of a certain size has a non-zero difference with the true distribution. Bias can be a good thing: it allows the estimator to make systematic generalizations not licensed by the data, e.g., a preference for re-use. It has been shown that an unbiased DOP

biased

19 The expected value of an estimator is the average estimate given all possible samples.

estimator is worthless: it must assign a weight of zero to any parse tree not part of the training corpus (Prescher et al., 2003). Still, the kind of bias is crucial. An issue with DOP1 is that it has a bias for larger trees Bonnema et al. (1999). There are many more large trees than small trees; analogously, a large tree has many more fragments than a small tree. This is reflected in DOP1's probabilities since these directly reflect frequencies. However, it is rather easy to remedy this particular deficiency by scaling the probabilities appropriately, as suggested by Bonnema et al. (1999) and Goodman (2003). Goodman's strategy, the equal weights estimate (EWE, cf. section 4.4), is employed by Bod (2003) and yields good results.

equal weights estimate

consistent

A more serious challenge is inconsistency. An estimator is consistent iff the predictions of the estimator get arbitrarily close to the real distribution as the amount of training data grows to infinity. Note that this property only guarantees convergence in the limit; an estimator can be consistent without performing well on real-world corpora, and vice versa. A reason for DOP1's inconsistency is the fact that, by design, it reserves probability mass for all fragments, even those for which productivity has not been attested. For example, in the Wall Street Journal, the contraction 'won't' is annotated as two words, but 'wo' does not combine with any other word, so the true distribution simply assigns a probability of zero for any fragment containing 'wo' but not 'n't', while DOP1 will always reserve probability mass for such combinations which may never materialize (Zuidema, 2007).

Observe why bias and consistency are orthogonal: bias is a property of all the estimates taken together (does the mean of the estimates equal the true distribution?), whereas consistency is a property of a sequence of estimators (will it progress towards and reach the true distribution?). If we take as an example throwing darts at a dartboard while aiming for the bulls-eye, then hitting a circular pattern around the bulls-eye is unbiased, no matter the error, while consistency refers to approaching the bulls-eye after practice.

The statistical oddities of DOP1 have instigated a search for more reasonable DOP estimators. Back-off DOP (Sima'an and Buratto, 2003) estimates probabilities of larger fragments by backing off to smaller fragments, similar to Katz smoothing for Markov models. However, Backoff-DOP starts with DOP1 probabilities, so it inherits DOP1's inconsistency problems.

DOP* (Zollmann, 2004; Zollmann and Sima'an, 2005) splits the training data in a held-out corpus and an extraction corpus. Fragments are taken from the extraction corpus, while probabilities are estimated based upon the shortest derivations of trees from the held-out corpus, which results in a consistent estimator. While this results in a considerably smaller number of fragments (fragments not part of any shortest derivation can be given a probability of zero), this efficiency is conditional on having sufficient coverage of the held-out corpus with respect to the extraction corpus. Without such coverage, a smoothing factor will introduce non-zero probabilities for all fragments, just as with DOP1 (ch. 4.3; Zollmann, 2004). Note that the kind of conservative estimation of DOP* is different from the parsimonious (Bayesian) models criticized earlier. Because DOP* is just an estimator, the original fragments and frequencies can be retained for an online model which is continually or periodically re-estimated. One could surmise that in language acquisition & usage, conservative estimation is precisely what separates active from inactive constructions.

The estimator DOPα (Nguyen, 2004) provides a different property, *rank*

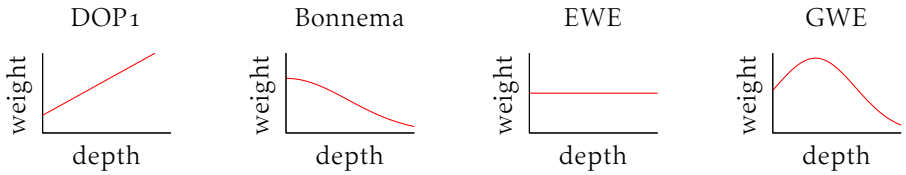

Figure 12: A sketch of the relation between depth and weight in various estimators, showing the contribution of fragments after the type frequencies of nodes have been factored out. 'Bonnema' refers to the estimator introduced in Bonnema et al. (1999). The Gaussian Weights Estimate proposed here is shown with $\mu = 4$ and $\sigma = 8$.

consistency, such that for every two fragments in the training data, the more frequent one will also be assigned a higher probability. Contrary to Back-off DOP, probabilities of fragments are built starting from the smallest fragments. While rank consistency does not imply consistency, the former is arguably more desirable than the latter. This is because rank consistency is stronger in the sense that it affects the estimates for all corpus sizes, while consistency only guarantees eventual convergence.

rank consistency

However, Zuidema (2006) argues convincingly that bias and consistency are simply not useful criteria for judging PTSG estimators: the problem of estimating fragment weights from corpora is *underdetermined*. We can also consider the frequencies of trees produced by a PTSG compared to the expected frequencies of trees in the true distribution. In this case it is possible to achieve consistency, as DOP* does, but only with an estimator that, in the limit, assigns all its weight to full parse trees. Different criteria such as speed of convergence (Zollmann, 2004) or robustness in the face of noise might yield more immediate benefits.

I also wonder whether basing an estimator on the shortest derivation (explicitly or implicitly) is the optimal choice. A 'medium' length derivation could increase re-use of fragments—'basic-level constructions' which are neither too specific (sparse) nor too general (not informative). While the EWE removes all bias regarding fragment size, we could also shift the bias. We can scale weights with a Gaussian, where μ defines the preferred (basic-level) depth, and σ determines the strength of this preference—i.e., a 'Gaussian weights estimate' (GWE). Stock phrases and idioms can still be parsed as one big chunk, but they have to be exceptionally frequent before they trump derivations with the more common, medium-sized fragments. Figure 12 sketches the relation between depth and weight in various estimators.

basic-level constructions

Gaussian weights estimate

Despite their desirable theoretical properties, these estimators have not been applied to large treebanks, because the computational techniques for DOP parsing of large corpora with a fragment-based parser were not available at the time. On the contrary, it is an inconsistent estimator (EWE) which has attained state-of-the-art performance for generative models on the WSJ (Bod, 2003). The non-trivial estimators just discussed were all evaluated on the Dutch OVIS corpus, with utterances collected from a public-transport information system. Its sentences follow common patterns, and have only 4.6 words on average (Zollmann, 2004). This is in stark contrast to newspaper corpora such as WSJ or

Tiger, with five times as many sentences, which are at least three times longer on average, and covering a much more diverse domain. It is likely that on such corpora the results for these estimators would be very different, for better or for worse. An empirical evaluation of these more sophisticated estimators on large corpora remains an important goal.

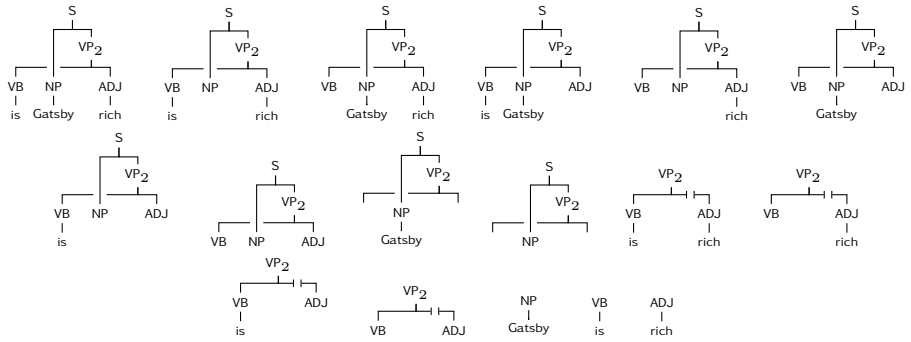

Figure 13: Discontinuous fragments as extracted from "is Gatsby rich?"

4.3 Discontinuous fragments

A DOP1 model includes frequencies for non-contiguous fragments. A fragment is non-contiguous when it has a frontier node that is surrounded by terminals. As such it is possible for DOP1 to model arbitrary constructions such as *from* NP_1 *to* NP_2, even when this construction does not consist of a single CFG production. However, this non-contiguity is only a property of the fragment as a whole, and has to be resolved to attain a full parse tree.

A discontinuous fragment contains at least one constituent whose terminals will still be non-contiguous after all frontier nodes have been substituted. As such, the discontinuity is part of the annotation, whereas non-contiguous fragments only exist in a statistical sense as co-occurrences in surface forms. Discontinuity reflects a richer representation that is independent of DOP's inherent strengths of modeling non-local dependencies.

The discontinuous tree structures in the Negra and Tiger annotations, combined with the definition of fragments as connected subsets of nodes from DOP1, suggest a natural generalization to discontinuous fragments. Note that, just as with DOP1, either all or none of a node's children are present; this is no different for discontinuous nodes. However, a discontinuous frontier node does specify explicitly where its components end up in the yield of the tree; in a DOP1 fragment this is implicitly specified through the order of nodes. Figure 13 shows the fragments extracted from a sentence with discontinuity. Refer to figure 14 for an example of a discontinuous derivation.

4.4 Goodman's reduction of DOP

Goodman (1996; 2003) offers a reduction of DOP to a PCFG where the number of rules is linear in the size of the training corpus. This allows polynomial time parsing, because the exponential number of fragments

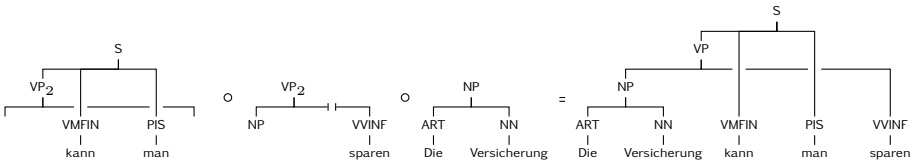

Figure 14: A discontinuous DOP derivation of the tree in figure 2.

is implicitly represented in the reduction. Note that it does not solve the problem of efficient disambiguation. Finding the most probable parse is still NP-hard and must be approximated.

Each node in the training data with label A gets a unique address, A_j. The probabilities reflect the number of subtrees headed by a node. Given a node A_j with children B_k and C_l, the number of subtrees is given by $a_j = (b_k+1)(c_l+1)$, which counts the subtrees of B_k and C_l and the possibility of them being frontier nodes. The total number of subtrees for A, including its occurrences in other trees in the training data, is given by $a = \sum_j a_j$. The normalization factor $\bar{a}$ is the frequency of non-terminals of type A in the training data. Using these frequencies, the reduction for a given node in the training data is as follows:

$A_j(\vec{\alpha}) \rightarrow B(\vec{\alpha_B})\ C(\vec{\alpha_C})$	$(1/a_j)$	$A(\vec{\alpha}) \rightarrow B(\vec{\alpha_B})\ C(\vec{\alpha_C})$	$(1/(a\bar{a}))$
$A_j(\vec{\alpha}) \rightarrow B_k(\vec{\alpha_B})\ C(\vec{\alpha_C})$	(b_k/a_j)	$A(\vec{\alpha}) \rightarrow B_k(\vec{\alpha_B})\ C(\vec{\alpha_C})$	$(b_k/(a\bar{a}))$
$A_j(\vec{\alpha}) \rightarrow B(\vec{\alpha_B})\ C_l(\vec{\alpha_C})$	(c_l/a_j)	$A(\vec{\alpha}) \rightarrow B(\vec{\alpha_B})\ C_l(\vec{\alpha_C})$	$(c_l/(a\bar{a}))$
$A_j(\vec{\alpha}) \rightarrow B_k(\vec{\alpha_B})\ C_l(\vec{\alpha_C})$	$(b_k c_l/a_j)$	$A(\vec{\alpha}) \rightarrow B_k(\vec{\alpha_B})\ C_l(\vec{\alpha_C})$	$(b_k c_l/(a\bar{a}))$

Where $\vec{\alpha}$ refers to the arguments of the LHS non-terminal. Each addressed non-terminal represents an internal node of a fragment, while the unaddressed nodes represent both the root and the frontier nodes of fragments. The latter allow a switch from one fragment to another during parsing, viz. they simulate substitution sites of DOP fragments.

The use of the normalization factor is called the Equal Weights Estimate; this formulation follows Bod (2003). Goodman (2003) first suggested this normalization but his formula appears to contain a mistake, having a_j in the denominator of the last four rules instead of a. The normalization is intended to counter the bias for large subtrees in DOP1—when all fragments are considered, the majority will consist of large fragments, which results in the majority of probability mass being assigned to rare, large fragments.

The difference with the PCFG reduction and this PLCFRS version is that a rule is not just defined by a set of non-terminals and their ordering, but also by a yield function specifying how the spans on the right-hand side compose the span(s) on the left-hand side. This does not affect the reduction because the discontinuities are only expressed in the yield of the tree, whereas the same subtrees can be (implicitly) enumerated by traversing the tree.

Figure 15 shows a concrete example of the reduction, using the trees on which figure 10 and figure 13 are based. The eight productions per node of the reduction can be considered as the pairwise Cartesian product of the original production and the one with addressed nodes. In other words, the productions $A \rightarrow B\ C$ and $A_1 \rightarrow B_2\ C_3$ give rise to the eight productions given above by evaluating $\{A, A_1\} \times \{B, B_2\} \times \{C, C_3\}$ and interpreting the resulting tuples as

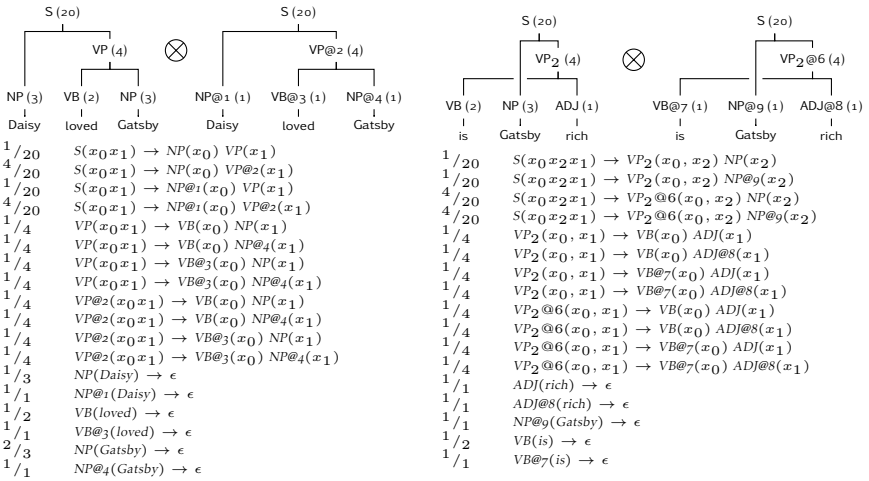

Figure 15: The pairwise Cartesian product of the original and the addressed tree gives the productions in the reduction.

productions. To get the reduction of a complete tree, this operation is applied to all productions of both trees in parallel. The probabilities are derived from the number of subtrees, shown in brackets after the node labels. In this case with only one tree, applying the normalization has no effect (i.e., $\bar{a} = 1$ for all $\bar{a}$). Productions without addressed nodes, i.e., the original productions, will recur, and their probabilities must be summed. In our case productions are considered equivalent when both the non-terminals and their arguments match.

state-splitting in the limit

Intuitively, the reduction can be seen as state-splitting in the limit. A state-split partitions a non-terminal into two or more new non-terminals to cover more specific and fine-grained contexts. Taken to the extreme, we can keep splitting non-terminals until each resulting non-terminal refers to one specific occurrence of that non-terminal in a single sentence, which greatly increases the amount of hierarchical information that can be extracted and exploited from the training corpus. This is exactly what happens in Goodman's reduction. Compared to other automatic state-splitting approaches such as latent variable grammars, this approach has the advantage of being conceptually much simpler.

The clear advantage of this reduction is that it makes it possible to parse DOP using existing, off-the-shelf tools. This comes at a price, however. The DOP model in this reduction is reduced to a mere probabilistic 'upgrade' over a PCFG or PLCFRS. This is because the actual fragments of DOP are only represented implicitly in the reduction. The parser is not actually working with fragments, it just ends up with homomorphic derivations and probabilities, which yield equivalent parse trees after addresses are stripped away and probabilities summed.

4.5 Double-dop

Double-dop (Sangati and Zuidema, 2011) is a new instantiation of dop which represents fragments explicitly. The number of fragments is exponential, so this requires a method of pruning the full set of fragments. Previous approaches sampled fragments at random, or introduced restrictions such as the number of frontier or terminal nodes, or the depth of fragments (Sima'an, 1999; Bod, 2001). A downside of all these methods is that they are rather arbitrary. There is no linguistic or cognitive reason why fragments should be restricted to such criteria. To put it succinctly, the problem is that these criteria are not data-oriented.

In Double-dop the number of fragments is made manageable by extracting only the largest fragments that occur at least twice in the training corpus. This has obvious advantages over random sampling, because the majority of fragments are made of large subtrees occurring only once, while fragments occurring multiple times are all more or less productive. It also has advantages over Goodman's reduction, because the probability model is explicitly defined over fragments, and each step of a derivation is available.

To apply Double-dop to discontinuous trees, we need only generalize the procedure for extracting fragments (Sangati et al., 2010) to trees that are described by lcfrs rules. Where normally the extraction traverses trees by comparing each non-terminal label to ones in other trees, here we must compare the yield function as well.

For two given nodes in trees t_1 and t_2 to match, it suffices that they have the same label (and, consequently, the same fan-out). If, additionally they should become an internal node of a fragment, their children need to match as well. In this case the lcfrs productions need to be equivalent.[20] After a collection of matching nodes has been obtained, the fragments can be collected by looking for connected subsets of nodes.

A working proof-of-concept implementation of discontinuous Double-dop has been made, but no large scale experiments have been performed as this demands careful optimization. An explicit fragment grammar like Double-dop is a prerequisite to implementing non-trivial estimators, so further developing discontinuous Double-dop is a worthwhile endeavor.

20 Formally this implies that their rules should be unifiable, but with the given algorithm for rule extraction, variables can be numbered deterministically, so that no unification is necessary.

Chapter 5

Disco-DOP

A synthesis of LCFRS and DOP realizes the goal of discontinuous data-oriented parsing. To approximate the most probable parse, we employ a general method for k-best derivation enumeration. A new method for parsing DOP efficiently using coarse-to-fine techniques is introduced.

We can now formulate a synthesis between LCFRS and DOP, to achieve discontinuous data-oriented parsing, or Disco-DOP. For practical reasons we will work with Goodman's reduction, albeit reluctantly. Since optimal binarization did not offer a clear advantage, we will disregard this aspect during binarization. We will use information on heads of constituents and binarize constituents head-outward; binarized constituents will be markovized to increase coverage of unseen productions.

However, before this model can be implemented, a number of technical issues need to be worked out. We first present the method of disambiguation that will be adopted. Next we discuss how to obtain a list of derivations so that the most probable parse can be approximated. This still leaves us with tractability issues, due to the large number of rules and non-terminal symbols in the DOP reduction. To address this, we present a novel method to prune the search space of the DOP grammar.

5.1 Simplicity Likelihood DOP

Most DOP models rely only on probabilities to identify the best analysis. This corresponds to the bias for frequently occurring fragments, but the original formulation of DOP included a preference for shorter derivations as well. Simplicity Likelihood DOP (Bod, 2003) realizes this by selecting the n most probable parses, approximated with a k-best list of derivations, and choosing the parse tree with the shortest derivation among these n parse trees (ties are resolved with probabilities again). This method gives better results than either method on its own. The probabilistic component makes use of the equal weights estimate (EWE).

Using Goodman's reduction the length of a derivation can be read off by counting the number of nodes and subtracting the number of addressed nodes, which reflect interior nodes of fragments, leaving only the unaddressed nodes which are the substitution sites of fragments. While approximating the most probable parse trees, the minimum derivation length for each parse tree can be tracked, so that the best parse tree can be extracted by sorting on both the probabilities and the minimal derivation lengths.

	derivations	F_1 %	EX %	CPU time (s)
PLCFRS	1	82.03	46.0	17.39
MPP DOP1	10,000	83.37	46.0	238.30
MPP EWE	10,000	83.99	47.0	234.70
SL-DOP $n = 7$	10,000	84.02	46.0	251.62

Table 3: A comparison of parse selection methods for DOP. The methods were tested on 100 sentences of 15 words, with a grammar based on 15 word sentences in the first 7200 sentences of Tiger. For an explanation of the F_1 and EX scores refer to section 7.1.

To select a suitable value for the parameter n, we could try a variety of values, bisecting until we find the best score. Bod (2003) reports $12 \leq n \leq 14$ to be the optimal values for the WSJ. On the one hand it is of course important to know the behavior of the model for different values of this parameter, but on the other hand it is crucial that the parameter has been chosen independently (i.e., without any knowledge) of the test set. To satisfy this restriction we could use a held-out part of the corpus (a development set), and determine the optimal value of the parameter given our grammar on the development set before touching the test set.

Instead of taking this route with methodological pitfalls,[21] I have opted early on to settle for a fixed value for n, namely 7. This was based on the observation that with a training corpus of 7200 sentences, most of the time there were less than 14 parse trees, especially with longer sentences. This means that the probabilistic bias would actually play no role if we were to consider up to 14 parse trees. On the other hand n should not be too low, because this gives less opportunity for simpler derivations to surface. See table 3 for a performance breakdown of the different methods compared to SL-DOP with $n = 7$. The gain of SL-DOP is very small in this comparison, but it is reasonable to expect it to grow with a larger training corpus, which offers more potential for short derivations.

5.2 Finding the k-best derivations

In order to approximate the most probable parse, the probabilities of a suitably large number of equivalent derivations need to be combined. In early DOP instantiations this was done by taking a random sample of derivations (Monte Carlo parsing), but this relies on sampling the DOP fragments, while these are not explicitly represented in Goodman's reduction. An alternative, employed in this work, is to use the k-best derivations, i.e., the top k items from the list of all derivations sorted in descending order of probability.

Extracting the 1-best derivation from the chart as produced by the parser described previously is trivial: start at the root node and follow the backpointers, each time picking the edge with the best inside score. But what about the second best? It has to be a derivation where at least one edge is suboptimal, such that the difference in score compared to the best derivation is minimized. We

21 Although there is a commonly used split of training, testing and development sets for Negra, this is not used by Kallmeyer and Maier (2010), whose experiments I set out to replicate. Therefore using a proper development set was not an option.

could enumerate derivations systematically by recursively taking the Cartesian product of edges and the children to which they point; but while the space of edges as traversed by the parser is quadratic (in a binarized grammar), the space of derivations (complete and partial) is exponential (i.e., intractable in general).

An algorithm for efficiently computing k-best lists given an exhaustive chart is given by Huang and Chiang (2005).[22] They cast the problem as enumerating hypergraph traversals. A k-best list is an enumeration of paths from the root node to terminals in a hypergraph, in descending order of probability. The hypergraph is simply the chart, the vertices are chart items (non-terminals and spans), and hyperedges are directed and (positively) weighted links between a chart item and one or more other items.

hypergraph traversals

Their algorithm initializes a priority queue[23] of edges for each chart item[24] and lazily[25] builds up derivations by taking a minimal number of items from these queues and incorporating the best one in the next derivation, until k derivations have been produced. So to find the second best derivation, the second best edges for all of the edges in the best derivation are explored and enqueued, after which it is guaranteed that the second best derivation is at the top of the queue. The insight which makes this efficient is that derivations do not have to be represented explicitly, but can be formed from ranked edges. A ranked edge not only has backpointers to chart items, but also specifies a derivation by including a rank for them, pointing to an item in the k-best list for that chart item. For example, given an NP with an NP and a PP as children, a derivation with backpointers could be NP-(2,3), meaning that its children are the 2nd best and 3rd best derivations, respectively. Note that this is different from the 2nd and 3rd best edges for that label and span, which could be trivially obtained by sorting the list of edges; instead, the ranked edge in turn points to other ranked edges.

ranked edge

Although the algorithm itself is very efficient and performs only the work that is absolutely necessary to make sure it arrives at the k-best items, the requirement of an exhaustive chart with Viterbi probabilities[26] is problematic with large grammars and long sentences. In a large grammar, there may be plenty of derivations which are far from the k-best derivations in terms of probability, which ideally should not have to be explored. Long sentences

22 Caveat lector: the published version of this paper contains mistakes; be sure to refer to the corrected version. Even so, I have found it necessary to further adapt their algorithm to fully rule out the possibility of generating duplicate derivations. I have replaced the part where the algorithm checks whether a new derivation is not yet in the queue of candidates by a check whether a derivation was ever seen before, because otherwise a situation could arise where a derivation has already left the queue and is enqueued once more.

23 Another caveat: the priority queue should resolve ties in priorities (probabilities) by falling back to insertion order, something which a typical heap-based priority queue does not keep track of, because heap sort is not a stable sort. This can be remedied by adding a monotonically increasing sequence number to each item as it is enqueued.

24 As noted by Huang and Chiang, an early optimization is to only add the k-best edges to this priority queue. This can be done optimally by employing Quicksort k-best selection, a variant of the Quicksort algorithm. In contrast to typical k-best selection methods, this algorithm returns its results in the original, unsorted order. Indeed, a commonly used k-best selection method actually inserts all elements into a heap and extracts k items, which is especially wasteful if the same items are then immediately added to a priority queue which duplicates the sorting effort.

25 Lazy computation is a technical term which implies that computations are deferred until it is sure that they are needed to obtain the next result.

26 Viterbi probabilities are the optimal (highest) probabilities to reach a certain non-terminal with a certain subset of terminals.

further exacerbate the situation because they afford a much higher number of possible analyses.

A better algorithm exists which incorporates the search for k-best derivations in the parsing algorithm given a heuristic: KA* (Pauls and Klein, 2009). With a suitable heuristic, the parser can construct the requested number of derivations long before all edges have been explored. A KA* parser for PLCFRS has not been implemented yet. It is unclear whether it would work well without a consistent heuristic which is admissible and monotone, a property which is assumed by Pauls and Klein (2009). Monotonicity guarantees that the Viterbi parse will be found first. Generalizing the (consistent) A* heuristics of Klein and Manning (2003b) to LCFRS has proven to be infeasible (Kallmeyer and Maier, 2010). Therefore in this thesis we will employ the algorithm of Huang and Chiang (2005).

5.3 Coarse-to-fine PLCFRS parsing

The grammars produced by Goodman's reduction are still rather large—Sangati and Zuidema (2011) cite a figure of 2.5 million rules for the commonly used training section of the Wallstreet Journal corpus (section 2–21). Therefore it is crucial to tame the search space somehow.

Bansal and Klein (2010) use the technique of coarse-to-fine parsing (Charniak et al., 2006) to prune parsing with the Goodman reduction. Given a grammar, one or more coarser levels are used, each defined as a projection of the previous level. The simplest coarse grammar projects all non-terminals to a single label, say X, except for the pre-terminals and the distinguished root label. Alternatively, a treebank PCFG can also be seen as a coarse grammar for a PCFG reduction of DOP, where all the addressed nodes project to their original labels.

Sentences are first parsed with the coarsest grammar, and using information from the resulting chart, items in the fine grammar can be blocked. At the very least items not contributing to a complete derivation will be eliminated (e.g., a derivation which fails to cover all of the terminals in the sentence). Additionally, the chart can be used to compute outside scores which can be used as estimates of the outside scores in the fine grammar; items whose score does not pass a threshold will then be pruned. However, these outside scores are computed with the inside-outside algorithm, which typically relies on enumerating bracketings. This is of course not possible for discontinuous constituents, as there are exponentially many.

The present work employs a middle road where items not occurring in the k-best derivations of the coarse grammar are blocked when parsing with the fine grammar; see figure 16. I have used $k = 50$ for all reported experiments. This method can be seen as an indirect implementation of a re-ranker, because the fine grammar will only consider derivations composed of the items in the derivations found by the coarse grammar.[27]

This strategy appears to work rather well, such that the exhaustive phase using the coarse grammar is the bottleneck. For short sentences (≤ 15 words)

27 Although it could well be the case that other derivations than the 50-best can be composed of the items in those derivations.

parsing with the DOP model takes less than a second. For longer sentences the coarse grammar requires much more time than the fine grammar, which suggests that parsing is slowed down by a large amount of improbable derivations, and not for example frequent ties between high probability chart items. This is supported by the observation of Levy (2005) that most of the observed complexity of LCFRS is caused by the high number of possible discontinuous items that can be derived. When parsing for the MPP with DOP1 as in table 3, turning coarse-to-fine off results in a parsing time of 780 seconds, such that the speedup is roughly threefold. This factor will increase for larger grammars and longer sentences, which allow for more ambiguity. There is also an accuracy gain of almost 1 percent with coarse-to-fine in this case, but this may well be fortuitous. A similar finding is reported by Bansal and Klein (2010).

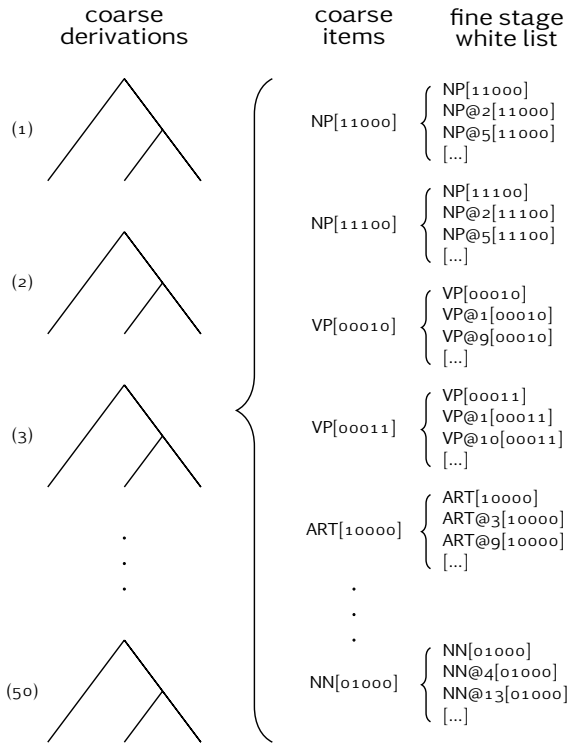

Figure 16: k-best coarse-to-fine inference. On the left are the k-best derivations from the coarse chart. In the middle the chart items (category and spans) occurring in those derivations. On the right is the result of projecting these items to the categories in the fine grammar. This will form the white list when parsing with the fine grammar.

5.4 Even coarser-to-fine parsing: from PCFG to PLCFRS

The coarse-to-fine approach just sketched helps a great deal, but as will be shown in section 7.2, the coarse stage presents a bottleneck which makes parsing long sentences impossible.

Instead of only making the categories coarser, we can also resort to a coarser formalism. Following Barthélemy et al. (2001), we can extract a grammar that defines a superset of the language we want to parse, but with a fan-out of 1. Concretely, a context-free grammar can be read off from discontinuous trees that have been transformed to context-free trees by the procedure introduced by Boyd (2007). Each discontinuous node is split into a set of new nodes, one for each component; for example a node NP_2 will be split into two nodes labeled NP*1 and NP*2 (like Barthélemy et al., we can mark components with an index to reduce overgeneration). Because Boyd's transformation is reversible, derivations from this grammar can be converted back to discontinuous trees, and can guide parsing with an LCFRS. The latter approach will be referred to as CFG-CTF. So far I have only implemented PCFG $\Rightarrow$ Disco-DOP, but more elaborate multi-level schemes such as PCFG $\Rightarrow$ PLCFRS $\Rightarrow$ Disco-DOP, as well as the previously mentioned collapsing of categories by Charniak et al. (2006), could conceivably increase both efficiency and accuracy, since extra levels of coarse PCFG grammars can be readily optimized with off-the-shelf tools.

However, markovized binarization presents an issue. It may me tempting to split the discontinuous nodes before binarization, because binarization might introduce spurious discontinuity, but this conflicts with markovization. The new constituents that can be covered by binarized productions depend on the number and order of non-terminals in productions, but these will differ when discontinuous nodes have been split into a set of new nodes—to obtain a context-free production, these new nodes have to be re-ordered.

Therefore we should binarize twice. The first time is before splitting discontinuous nodes, and this is where we introduce markovization. This same binarization will be used for the fine grammar as well, which ensures the models make the same kind of generalizations. The second binarization is after splitting nodes, this time with Chomsky normal form (modulo unary productions). Figure 17 illustrates this approach.

Parsing with this approach proceeds as follows. First, the sentence is parsed using the split-PCFG; using the resulting chart the k-best derivations are extracted. From these derivations, the second level of binarization is undone, after which the split labels are merged back into discontinuous nodes. From these k-best discontinuous derivations the chart items for the white list are read off. Using this white list parsing with the fine grammar can be pruned.

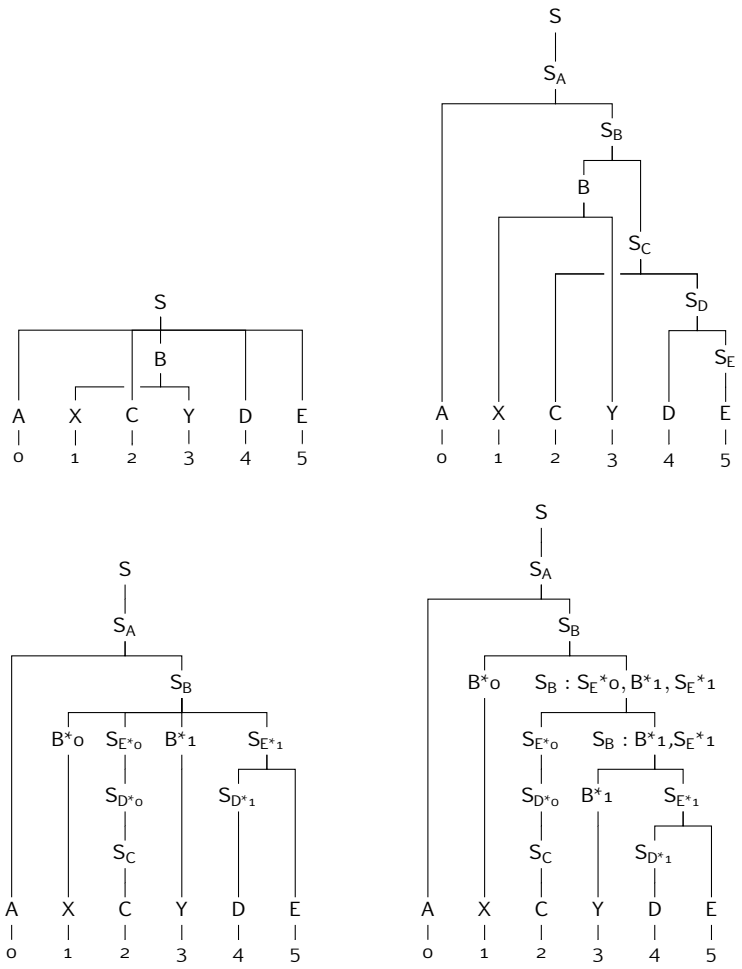

Figure 17: Transformations for a context-free coarse grammar. From left to right: the original constituent, markovized with h=1 v=1, discontinuities resolved, normal form (second binarization).

Chapter 6

Implementation

A high-level prototype can be incrementally adapted with low-level optimizations. This is achieved by starting from an implementation in a high-level, dynamic language, and adding type annotations to performance-critical parts, without sacrificing integration with the rest of the code.

Matters of implementation are usually not reported in computational linguistics, or at best in a cursory fashion. Implementation is, like methodology, considered a boring aspect. If at all possible, off-the-shelf components are employed. I believe, however, that in order to gain a deep understanding of algorithms and the problems they solve, one has to implement them. Especially in the case of discontinuous constituency parsing, the right implementation is crucial to getting things off the ground at all, because of the challenge in terms of efficiency.

6.1 Python

Implementation[28] started with the tools provided by the Natural Language Toolkit (nltk; Bird et al., 2009), which is a comprehensive Python library written with didactic purposes in mind. Python is a high-level, dynamic language which is very suitable for prototyping, and appreciated for its simple syntax that, in the best case, reads like pseudo-code. At first, unification appeared to be a natural way to instantiate rules. Nltk provides feature structure objects which can contain variables and strings, as in the following example:

```
rule = FeatStruct("[[VP_2, [?X], [?Y]], [PROAV, ?X], [VVPP, ?Y]]")
```

The next step was to devise a representation for discontinuous parse trees. The `Tree` object of nltk provides an implementation of traditional phrase-structure trees, where the order of leaves is implicitly specified through the order of non-terminals. Initially it seemed a good idea to create a subclass of `Tree`, but a far simpler idea suggested itself: separate the representations of the sentence and its tree structure. The leaves on the tree can refer to positions in the sentence with integer indices, which means all of the methods on tree structures, such as binarization, post-order traversal, and enumeration of leaves, could be re-used. This separation neatly fits the conceptual picture of the

28 The source code of the parser and all other components is publicly available from `http://github.com/andreasvc/disco-dop`

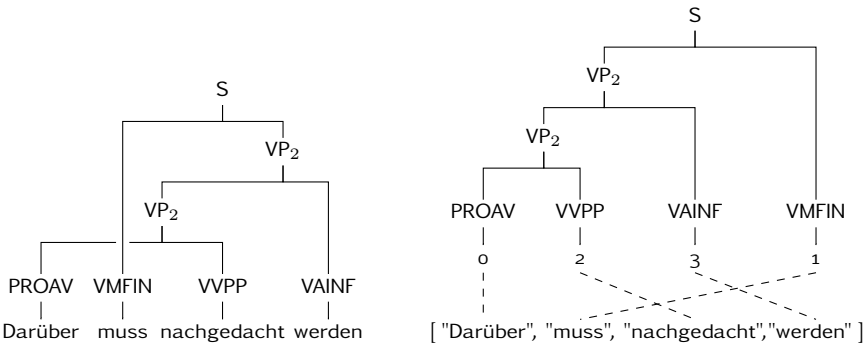

Figure 18: An illustration of the representation. Shown on the left is the original discontinuous tree from the Negra corpus. On the right is a version in which the phrase structure has been decoupled from its surface form.

trees in the Negra format—horizontally there is a total order on terminals, while vertically every non-root node has exactly one parent. This allows for the representation shown in figure 18. Internally, this is represented in two separate objects (note that indexing is zero-based):

```
tree = Tree("(S (VP_2 (VP_2 (PROAV 0) (VVPP 2)) (VAINF 3)) (VMFIN 1))")
sent = ["Darüber", "muss", "nachgedacht", "werden"]
```

A working prototype, based on the pseudo-code given in Kallmeyer and Maier (2010) coupled with the idea of rules as feature structures was ready in a few hours, in about 130 lines of code. Using a manually derived Goodman reduction, there was an immediate proof of concept for discontinuous DOP parsing.

However, due to the didactic focus of NLTK, unification turned out to be a very slow operation. Even when rewritten to use only the built-in data structures of Python (sets, tuples, lists, dictionaries), the parser was much too slow.

At this point most people would conclude that Python is somehow an inherently slow language and not suited for the task. There has historically been little attention given to the speed of the main Python implementation (called CPython, because it is written in C). The general sentiment is that the currently popular dynamic languages (such as Python, Ruby, Perl) are inherently unsuited for efficient implementation. This is because of far-reaching dynamicity; for example a function call may end up executing arbitrary code, through late binding (whereas in a static language, a function such as `printf` is de-referenced at compile time, and is executed immediately). These and other dynamic features, are part of the language design—not just particular implementations of them.

However, a great deal of code does not exploit these dynamic features, or only to a limited extent. This makes it possible to take a program written in such a dynamic language, but treat it as a statically typed program, enabling compile-time optimizations. One method for doing this is Just-In-Time compilation. This means that while the program is running, the code is analyzed in search for performance-critical parts, which can then be optimized using potentially

costly methods. The PyPy project (e.g., Bolz et al., 2009) is a good example of this. The advantage is that unmodified Python code can be accommodated—the project already supports the complete Python language. The downside is that everything has to run in PyPy's specialized virtual machine, and there is no good way to interface with native C code—which is actually one of CPython's strong points.

Another approach is taken by ShedSkin (Dufour, 2006), which translates a restricted subset of Python to static C++ code. To achieve this it performs a whole program analysis with the Cartesian Product Algorithm in order to infer the most specific types that can be assigned to variables and arguments of functions. All of the supported Python built-ins and modules are replaced with C++ equivalents.

Both of these projects are theoretically very appealing, but their approach did not suit the needs of the parser under discussion. PyPy did not seem to offer significant speedup, and with its virtual machine magic there seemed to be little transparency and control over performance. ShedSkin was tricky to get off the ground at all: if the program is not written in a very particular style, i.e., implicitly typed, ShedSkin's attempt at translation simply will not terminate. It did work in the end with a tweaked version of the parser, but there is a lack of transparency—getting things to work well requires tweaking and guesswork. Both projects are challenged with respect to integration with other code; PyPy lacks proper support for C code, while ShedSkin does not support arbitrary Python code.

6.2 Cython

Fortunately, Python is a very active area of research, so a third option presented itself: Cython (Behnel et al., 2011). The idea of Cython is also to translate Python code, this time to C, but the difference is that static typing is completely optional and added manually through type annotations. With a few exceptions, Cython can translate most Python code completely to C code. This will typically not improve performance, however. It does mean that there is now a bridge between the dynamic Python semantics, and the static close-to-the-metal semantics of C. The basic idea of optimized Cython code is to add strategically placed type annotations. Figure 19 compares the translation to C of plain Python and a type-annotated version of incrementing a variable (some error-checking boilerplate has been elided from the part without type annotations). This demonstrates the overhead that is involved with Python code.

type annotations

The next step is to analyze the code to identify 'hot spots' which should be optimized. This is typically done through profiling—measuring how long each function call takes, and how often it is called. Cython also provides another source of information: it can create an annotated source file, showing for each line of Python code the resulting C code. As a heuristic of candidates for optimization, lines which involve more Python calls are marked with darker shades of yellow. This is only a heuristic, however, because replacing Python's heavily optimized dictionary with some other implementation is probably not going to give an improvement, although it will remove all the yellow lines caused by dictionary operations. Other simple changes will give significant

```
/* "test.pyx":1    a = 0 */
if (PyObject_SetAttr(__pyx_m, __pyx_n_s__a, __pyx_int_0) < 0) {
    __pyx_filename = __pyx_f[0]; __pyx_lineno = 1;
    __pyx_clineno = __LINE__; goto __pyx_L1_error; }
/* "test.pyx":2    a += 2 */
__pyx_t_1 = __Pyx_GetName(__pyx_m, __pyx_n_s__a);
__Pyx_GOTREF(__pyx_t_1);
__pyx_t_2 = PyNumber_InPlaceAdd(__pyx_t_1, __pyx_int_2);
__Pyx_GOTREF(__pyx_t_2);
__Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
if (PyObject_SetAttr(__pyx_m, __pyx_n_s__a, __pyx_t_2) < 0) {
    __pyx_filename = __pyx_f[0]; __pyx_lineno = 2;
    __pyx_clineno = __LINE__; goto __pyx_L1_error; }
__Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;

/* "test.pyx":3    cdef int b = 0 */
__pyx_v_4test_b = 0;
/* "test.pyx":4    b += 2 */
__pyx_v_4test_b = (__pyx_v_4test_b + 2);
```

Figure 19: The effect of type annotations on incrementing a variable in Cython.

speedups, for example replacing the Python function `max()` with a C function that only operates on primitive data types; this is because in C comparisons are compiled to dedicated CPU instructions, while Python allows any object to define its own comparison operators, and resolves this at run-time.

If the code is primarily numerical, then adding some primitive type declarations can provide a sufficient speedup. But arguably Python's strength is in more high-level and abstract code. General purpose optimization also involves complex data structures such as lists, dictionaries, and user-defined classes. Properly optimizing data structures requires parametrized polymorphism, i.e., being able to specify that l is of type list whose elements will be of type t. Such a feature is not available yet in Cython, but if we sacrifice type safety, we can have the same performance gain using type casts. Whenever we take an element from a list or a dictionary, we cast it to the right type:

```
rule = <Rule>rules[i]
```

When `rule` is declared to be of type `Rule`, this cast avoids a type check to see whether the assignment is valid. On the other hand, if some other kind of object made its way into `rules`, the first operation on `rule` will result in undefined behavior, such as reading an illegal memory location, which will abruptly halt the program. Usually this can be ruled out, though. When some line of code is executed, say, a million times per sentence, the benefits can be worthwhile.

So far the improvements only avoid calling the Python libraries and hence provide little in the way of drastic improvements. The biggest improvement came from replacing container data structures, in our case rules, chart items and edges, with so-called extension types (also known as `cdef classes`, after their keywords). Extension types appear in Python code as if they were normal

extension types

objects, but internally they are completely implemented in C. In the case of Cython this means that on the one hand an object can inter-operate with plain Python code, while on the other hand being handled efficiently in Cython code because of static type annotations. Conceptually, an extension type can be thought of as a C `struct`—that is, a complex data type where the type of each element is known at compile time—augmented with glue code to interface with plain Python code when needed. Like other classes, an extension type can contain methods as well. Take the example of getting the inside probability from an edge:

```
prob = edge.inside
```

If `edge` is a normal Python object this would involve a dictionary lookup of the key `'inside'` in the `edge` object; if this Python object would be accessed from Cython code where `prob` is a C variable of `double` precision, the Python `float` object would also have to be converted to C format (unboxed). If, on the other hand, `edge` is of type `Edge`, implemented as a `cdef class`, then this statement translates to a direct lookup using pointer arithmetic, i.e., `edge->inside` in C parlance. This is because the structure of an `Edge` object is fully known at compile time, such that getting the value of the member `inside` is as simple as adding an offset to the memory location of the `edge` object.

As a case study, we will now look at the most performance-critical part of the parser: verifying whether two chart items can be combined according to a grammar rule. The rule specifies two non-terminal labels, and all matching candidates can be easily found with a lookup table. And since each non-terminal label is required to have a unique fan-out, their yields will always have the number of components prescribed by the rule. Checking whether their yields actually follow the pattern prescribed by the rule is a more involved process. The yield of a chart item is represented with a bit vector. For example the number `0b110010` specifies a constituent covering the second, fifth, and sixth word in a six word sentence (indices are read from right to left, because the leftmost bit is the most significant bit). With this representation, it is already trivial to check whether the two chart items overlap—given two bit vectors, their bitwise `and` should be zero. This still leaves some other cases to check for. Assume a rule where the second non-terminal is to be wrapped around the first, resulting in a continuous yield:

$$S(xyz) \rightarrow NP(y)\ VP(x, z)$$

These are some example cases:

```
>>> concat(rule, 0b001100, 0b100011)
False # last two components not contiguous
>>> concat(rule, 0b11001100, 0b00110011)
False # unexpected second component in first vector
>>> concat(rule, 0b001100, 0b110011)
True # success; new yield will be 0b111111
```

Clearly, to verify compatibility we have to check each component (i.e., contiguous sequences of bits), of each bit vector. A naive strategy is to simply check

each bit of both vectors, giving an $\mathcal{O}(n)$ algorithm, where n is the length of the sentence. Fortunately, we can do much better, by exploiting well-known methods on bit vectors. Specifically, we can jump right to the boundaries of each component by asking for the next set or unset bit of a bit vector. By shifting the vector we can get the next set or unset bit from a specific position.

The code is given in algorithm 4; the algorithm is from Maier's `rparse`.[29] What is special about this implementation is that it translates to 100% native C code. The operations on bit vectors, `testbit`, `nextset`, `nextunset`, are implemented with built-in functions of the C compiler (which exploit specialized CPU instructions when available). A rule is composed of non-terminal labels `lhs`, `rhs1`, and `rhs2` (not used here), and two arrays `args` and `lengths`. The former, `args` contains a bit vector for each component of the left-hand side. With the previously mentioned rule this would be just xyz. Since rules are binary and ordered, we can represent variables originating from the left corner of the rule (`rhs1`) with a `0`, and the right corner variables with a `1`. The first occurrence of a `0` refers to the first variable of the corresponding non-terminal, and similarly for the other variables and for the right corner variables. So the arguments xyz from the example rule would come out as the value `0b010` in `args[0]`. The only thing that's missing is to know where it stops, because in a bit vector represented as an integer, there is of course an infinity of zeroes after the most significant set bit. The second array, `lengths`, takes care of this, by simply giving the number of variables for each argument position, i.e., `lengths[0]` equals 3.

This is all very neat, but what is the advantage, the reader may now wonder? Most parsers, e.g., `bitpar` (Schmid, 2004) and Mark Johnson's `cky`, are simply implemented in static languages from the start, to avoid these performance issues. In the past the most commonly used languages were C and C++. Nowadays Java is getting more popular—`rparse`, Double-DOP, the Stanford parser, and the Berkeley parser were all written in Java, for example. C is closest to the machine, and one little mistake in memory handling will lead to inscrutable error messages. C++ adds abstraction to this, but resolves all abstraction at compile time so that it does not affect the performance of the final program. Unfortunately this leads to very cryptic error messages, because the underlying mechanism is a templating (macro) language, such that the source of an error can be hard to track down. Java is not as 'close to the metal,' (e.g., it provides garbage collection so that no memory management is needed) but is able to provide good performance through its highly optimized virtual machine (which uses a tracing JIT like PyPy). These languages do solve performance issues adequately, but, arguably, at the cost of increased development time and unnecessary complexity. They require that the *whole* program be written in a verbose, error-prone, statically-typed style. Of course, static typing also *prevents* errors by detecting type errors at compile time, but it does so by forcing the user to 'draw within the lines,' rather than supporting free expression of ideas. Unless the typing system is truly baroque (i.e., Turing-complete), static typing can only give a modest lower bound on program correctness.

With Cython, one can rapidly complete a prototype, and optimize only its performance-critical parts. Corpus pre-processing, reading off grammar rules, evaluation—these have little to gain from optimization, but implementing them in a different language and interfacing with text files is also an error-prone

29 Cf. `http://www.wolfgang-maier.net/rparse`

```
cdef bint concat(Rule rule, unsigned long lvec, unsigned long rvec):
    if lvec & rvec: return False
    cdef int lpos = nextset(lvec, 0)
    cdef int rpos = nextset(rvec, 0)
    cdef unsigned int n, x
    cdef unsigned short m
    # this algorithm was adapted from rparse, FastYFComposer.
    for x in range(rule.args.length):
        m = rule._lengths[x] - 1
        for n in range(m + 1):
            if testbitint(rule._args[x], n):
                # check if there are any bits left, and
                # if any bits on the right should have gone before
                # ones on this side
                if rpos == -1 or (lpos != -1 and lpos <= rpos):
                    return False
                # jump to next gap
                rpos = nextunset(rvec, rpos)
                if lpos != -1 and lpos < rpos:
                    return False
                # there should be a gap if and only if
                # this is the last element of this argument
                if n == m:
                    if testbit(lvec, rpos):
                        return False
                elif not testbit(lvec, rpos):
                    return False
                # jump to next argument
                rpos = nextset(rvec, rpos)
            else:
                [...]
    # success if we've reached the end of both left and right vector
    return lpos == rpos == -1
```

Algorithm 4: The function that verifies whether two bit vectors can be combined according to a rule. The elided part is analogical to the first part, except that left and right are swapped.

process. With Cython one can implement a `Rule` object, for example, that can be instantiated from plain Python code, while type annotations allow direct access to its underlying C array from the Cython-optimized parser.

6.3 Data structures

The right algorithms are the first pre-requisite of an efficient program. The second is choosing the right data structures. There are three important data structures in the parser: the agenda, the Viterbi chart, & the full chart. The agenda uses a heap-based priority queue with the decrease-key operation, whose operations have logarithmic time complexity; additionally the agenda maintains a dictionary so that the probabilities of items can be looked up. The two charts are also dictionaries. The grammar is a minor data structure in terms of efficiency because it is read-only. It is implemented with lookup tables indexed on the label of the first and second part of the right-hand side.

The decrease-key operation can be implemented in logarithmic time (Cormen et al., 2001), by locating the item and sifting it up until it reaches the right place in the heap.[30] There is a much simpler algorithm, though. In the Python documentation[31] it is suggested to remove and change the priority of items by marking the original one as invalid. By maintaining a mapping of keys and values in the heap, the old item can be efficiently located, and marked invalid by assigning it a sentinel value. When popping items from the heap, items marked as invalid are simply ignored. This simplifies the algorithms, and appears to work efficiently as well. Marking items as invalid has minimal computational overhead, at the expense of memory efficiency. However, the number of items in the agenda, valid or invalid, is bounded by the number of items which have to be stored in the full chart anyway, so memory is of no concern. Compared to a complex priority queue such as the Fibonacci heap, this implementation has unremarkable asymptotic performance, but the advantage is that it does not have prohibitive constant factors or amortized costs.

Python dictionaries are implemented as open addressing hash tables, which have constant amortized time complexity. However, the performance of hash tables is strongly dependent on the choice of hash function. The objects which are used as keys in the charts and the agenda are chart items, defined by a label and bit vector. Initially a generic hash function was used, which is equal to Python's internal hash function for tuple objects:

```
#item._hash = hash((label, vec))
item._hash = (1000003UL * ((1000003UL * 0x345678UL)
            ^ label)) ^ (vec & ((1 << 15) - 1) + (vec >> 15))
```

The suffix `UL` indicates an `unsigned long` value, `^` is the bitwise exclusive or, and `a << b` shifts `a` to the left by adding `b` zero bits to its right; `>>` is the

30 The terminology 'to sift up' in the context of heaps refers to performing rotations on a heap until the heap property is restored.

31 Python documentation, 8.4.2. Priority Queue Implementation Notes, cf. `http://docs.python.org/library/heapq.html`

corresponding right shift.[32] This formula pseudo-randomly shuffles the bits in `label` and `vec`.

Maier's `rparse` uses a famous hash function due to Daniel J. Bernstein:[33]

```
def DJBHash(key):
    hash = 5381
    for i in key:
       hash = ((hash << 5) + hash) + ord(i)
    return hash
```

The variable `key` is a sequence of items to be hashed, and the function `ord` gives an ordinal (bitwise) representation of its argument. This hash function is widely used, but like the previous function, there is no theoretical justification for why it should lead to good results. Pseudo-random number generators (of which hash functions are an instance) are only studied stochastically, not deterministically. It is possible to analyze the collision rate of a hash function under a given distribution of data with statistical techniques. However, it is not possible to say what makes a good hash function; or for instance what the effect of certain constants will be. Why prime numbers are good candidates for such constants is probably a deep mathematical question, but for all we know it is just superstition to insist on them.

Generic hash functions are designed to summarize an arbitrary amount of information into a hash value. The size of this hash value typically reflects the word size[34] of the machine, which is 64 bits on modern machines. In our case, however, there is a much better solution than shuffling bits around. Recall that the bit vector represents the presence of a word in a constituent as a single bit. The category of a non-terminal is represented by an integer, derived from a mapping of the labels in the grammar to the natural numbers. We can optimize for the common case where sentences have much less than 64 words, and assume that the category of an item can fit in the rest of the 64 bits.This gives us a hash function which, in most cases, exactly represents the original information:

```
item._hash = label ^ (vec << 31UL) ^ (vec >> 31UL)
```

This function gave a significant performance improvement. One particular sentence was processed in 283 seconds with the first hash function, and in only 164 seconds with the last hash function. The reason for this is probably locality of reference. Python's dictionary implementation is designed to approximate the behavior of an array in case of densely packed hash values. Since the bit vector is shifted to the left, the least significant bits are taken by the category. The consequence is that, with high probability, all possible categories for one

32 Caveat lector: when doing right bit shifts, use unsigned values to ensure a logical shift instead of an arithmetical shift. An arithmetical shift will extend the sign bit which only makes sense for numerical data in two's complement format. Regrettably, The Python language does not include a logical right shift operator, but in Cython its use can be forced by declaring (or casting) values to be unsigned.

33 I have not been able to locate the original source for this algorithm, but cf. `http://www.cse.yorku.ca/~oz/hash.html` and `http://www.partow.net/programming/hashfunctions/index.html`. These links are also the source of the observations in the following paragraph.

34 In computer architecture, the word size refers to the number of bits that a CPU operates on. Many operations on word sized data execute in a single cycle, and memory addresses are word sized.

particular span now form a contiguous part of the array implementing the hash table.[35] while with the generic hash functions they would be randomly distributed, by design.

Note that this discussion tacitly assumed that there will be no sentences longer than 64 words, as this is not feasible with the current algorithms and techniques anyway. However, it is not difficult to replace the word sized integers with generic bit vectors represented in an array of integers. It will come with some performance penalty because operations like equality, intersection (bitwise and) and union (bitwise or) will require more than a single CPU cycle and memory access. However, the idea of optimizing for the common case of shorter sentences will still hold.

6.4 Code overview

Python code:

`grammar` reads off PLCFRS or Goodman reduction from trees
`treetransforms` markovized & optimal binarization methods, splitting discontinuous nodes
`negra` corpus reader for the Negra format; also assigns heads using head rules
`eval` PARSEVAL evaluation
`gen` experiment in PLCFRS generation
`runexp` script to run an experiment based on parameters such as the corpus, number of training sentences, and length restrictions
`tikzdtree` semi-automatic discontinuous tree drawing

Cython code:

`plcfrs` the parser
`kbest` enumerate the k-best derivations
`coarsetofine` produces a list of items which should not be pruned
`disambiguation` various DOP disambiguation methods based on a list of derivations
`agenda` a heap-based priority queue
`bit` operations on bit vectors such as 'find next set bit after position x'
`containers` defines objects such as chart items, edges & grammar rules
`fragmentseeker` experimental implementation of Sangati et al. (2010)
`estimates` experimental implementation of the SXLR context-summary estimates described in Kallmeyer and Maier (2010)

Together these modules make for 3,375 source lines of code (SLOC).[36]

35 The actual results will depend on the implementation of the hash table, specifically how it deals with collisions and resizing—before the hash table reaches the high watermark, collisions will occur, spreading items randomly.

36 This figure was generated using David A. Wheeler's '`SLOCCount`'.

Chapter 7

Evaluation

We now review evaluation metrics for parser performance and present our results, which improve on previous work.

One of the defining features of computational linguistics is the way work in the field is evaluated against common benchmarks. This practice has downsides—the competitive nature can cause one to lose sight of the original goal, which is not to attain the best scores on a particular newspaper, but to parse well. On the other hand it has also brought the field together and fostered progress.

7.1 Metrics

Judging the quality of a parser can be automated by defining a metric for a sequence of parses relative to the so-called gold standard trees from the treebank. The most objective and stringent metric is the exact match criterion which is the number of parses which completely match their corresponding gold trees. The downside is that it is coarse-grained; i.e., it does not discriminate between almost-correct and not-even-close parses. Inconsistency or spurious ambiguity in the annotations will cause the score to plummet.

gold standard

exact match

A different strategy is taken by the Parseval measures (Black et al., 1992), which is the most commonly used evaluation for phrase-structure trees. It compares individual constituents of the parse trees to the annotated trees from the treebank. Specifically, a constituent is considered as a labeled bracketing that defines which terminals the constituent dominates. However, it is not enough to simply consider the percentage of correct constituents. This is because the parser might well produce a different number of constituents than the annotators of the gold corpus. This mean that evaluation could be based on the ratio of correct constituents to either the constituents in the parses or those in the gold corpus. But in either case it is trivial to maximize the score. In the former case (called precision), the parser can aim to produce the smallest number of constituents, which minimizes the number of potential mistakes. Conversely, in the latter case (called recall) the parser can maximize the number of constituents to catch more of the gold constituents. To counter this, the average of the precision and recall can be used, but this still affords the possibility of artificially raising the score at the expense of either precision or recall (just as scoring 2 out of 10 on a midterm and 9 out of 10 on the final does not make a passing grade in most courses). A solution is the f-score (technically, the F_1 measure), which is the harmonic mean of the precision and recall. The harmonic mean is more strongly affected by (gravitates to) the lowest score.

Parseval

labeled bracketing

precision

recall

f-score

A constituent for a sentence is considered correct if a constituent with the same label and covering the same terminals exists in the tree in the treebank. This means that to evaluate a set of parses, we read off the set of constituents in the parses and in the corresponding trees in the treebank, of which the intersection represent the correct constituents. Precision, recall and f-score are then calculated as follows, given sets P and G with the constituents from the parser and the treebank, respectively:[37]

$$\begin{aligned} \text{precision} &= \frac{|P \cap G|}{|P|} \\ \text{recall} &= \frac{|P \cap G|}{|G|} \\ \text{f-score} &= \frac{2}{\text{precision}^{-1} + \text{recall}^{-1}} \end{aligned}$$

For continuous phrase-structure trees, a constituent is defined by a category and the indices of the beginning and end of its yield, e.g., $\langle NP, \langle 2, 5\rangle\rangle$. To generalize this to discontinuous trees, we can straightforwardly consider the yield as a subset of indices, for example $\langle NP_2, \{2, 3, 5, 6\}\rangle$.

Note that comparing individual constituents in this way conveniently follows the independence assumptions made by context-free grammars (and to a lesser extent DOP). As such, the scores reflect the quality of individual parsing decisions, but not how appropriate these decisions are in combination.

It has been observed that the PARSEVAL measures have a bias for larger trees; i.e., the score for trees annotated with more constituents will necessarily be higher when the number of mistakes is constant (e.g., $^9/_{10} > {}^4/_5$). This can also be expressed as the node-to-terminal ratio in the annotation. An important consequence is that this makes comparisons of scores between different treebanks problematic (Rehbein and van Genabith, 2007). A score of 80% is state-of-the-art for Negra, while on WSJ it is just above the PCFG baseline. It should not be concluded from this that parsing German or Negra specifically is harder than English or the WSJ.

Another issue with PARSEVAL is that when a constituent is attached to the wrong part of the tree in the parser output, there will be multiple constituents missing that part of their yield, such that a single attachment error is actually treated as a series of errors. One could reasonably conclude that attachment errors are in fact punished too severely.

A particular issue with the generalization to discontinuous trees of PARSEVAL is that discontinuous constituents are harder to get right for the parser. With the commonly used method to transform the Negra and Tiger treebanks to continuous constituency trees, all components except the one with the head are re-attached to higher nodes in the tree, until no discontinuity remains. This transformation throws away the information that these components are related, so the task of parsing is made easier. Conversely, when the parser not only has to produce discontinuous constituents, but is also evaluated on them, a single missing terminal in one component makes the whole constituent incorrect, because a discontinuous constituent is evaluated as a single unit.

37 Note that the f-score is usually defined with a different but equivalent formula: $\frac{2pr}{p+r}$. This formula, however, obscures the reason for the name 'harmonic mean' which is due to the use of reciprocals.

Alternative metrics have been suggested. The leaf-ancestor metric (Sampson, 2000) looks at the path of each terminal to the root node, and scores these paths by computing the Levenshtein distance[38] against the tree from the gold corpus. leaf-ancestor metric

The idea of using the Levenshtein distance is interesting because it judges the parse tree according to the minimal cost to transform it to the gold tree. However, the choice to evaluate on paths to the root node is still somewhat arbitrary: why single out yet another aspect of the tree structure? There exists a generalization of the string edit distance to trees, giving the tree edit distance (Zhang and Shasha, 1989). This has been adapted to the case of evaluating parse trees (Emms, 2008). The concept of tree edit distance as a metric appears to me to be the most neutral and objective, aside from the exact match criterion, because it does not focus on a single aspect of parse trees. However, the weights and specific operations of the edit distance still have to be defined and justified—it is not obvious that the current definition of operations on individual nodes with unit costs is optimal. Emms defines two similarity metrics, Dice and Jaccard (Dice coincides with the F_1-measure in the case of comparing two sets). We will use the macro-averaged versions of these; i.e., as with PARSEVAL, the average is computed over all trees lumped together, rather than the average of scores for each individual tree. Given the set of gold nodes G and nodes produced by the parser P, the tree-distance consists of the minimal sets $\mathcal{D}$, $\mathcal{I}$, $\mathcal{S}$, and $\mathcal{M}$ containing nodes that are deleted, inserted, swapped, or matched with the gold trees, respectively. The tree-distance metric is then computed as follows: tree edit distance

$$\text{dice} = 1 - \frac{|\mathcal{D}| + |\mathcal{I}| + |\mathcal{S}|}{|G| + |P|} \qquad \text{jaccard} = 1 - \frac{|\mathcal{D}| + |\mathcal{I}| + |\mathcal{S}|}{|\mathcal{D}| + |\mathcal{I}| + |\mathcal{S}| + |\mathcal{M}|}$$

However, this still assumes that it is reasonable for a parser to reproduce the full parse tree of the gold corpus in all its detail. An alternative is to look only at a pre-defined set of relations in a sentence. This approach is taken in dependency evaluation (Lin, 1995). Using a transformation of constituency trees to dependency structures, parse trees can be evaluated against the usual metrics for dependency structures (labeled and unlabeled attachment scores). dependency evaluation

Yet another evaluation strategy is to parse only difficult constructions. This avoids 'noise' caused by annotation schemes and characteristics of different languages, and avoids inflating the score with easy sentences. The Tepacoc[39] test set is one such approach (Kübler et al., 2008, 2009). Previous work on the German treebanks appeared to indicate that the Tüba-D/Z treebank, with its WSJ-inspired hierarchical annotation, resulted in better accuracies. With the Tepacoc test set, however, it is shown that Tiger and Tüba-D/Z are comparable in performance when evaluated against the linguistically most interesting & complex constructions of Tepacoc as selected from the two treebanks. Tepacoc

Despite these alternatives, this work still employs the PARSEVAL measures, because it is important to be able to compare results with previous work, and because it is a well-established metric. Future work should address the problem

38 The Levenshtein distance is one form of an edit distance. It reflects the minimal cost required to transform one string into another, where deletions, insertions and substitutions of individual characters have pre-defined costs.

39 Tepacoc stands for Testing Parsers on Complex Constructions.

of evaluating discontinuous constituents specifically (as opposed to generalizing metrics intended for continuous constituents), since it comes with its own set of issues.

7.2 Results

Results are for models based on splits of 90% training data and 10% test data. Following previous work, the parser is presented with part of speech (pos) tags from the test corpus (gold tags). The dop model, however, exploits its knowledge of lexical dependencies by using subtrees with terminals, as long as the pre-terminals match the given gold tags. In a preprocessing step, function labels are discarded and all punctuation is lowered to the best matching constituent,[40] following Kallmeyer and Maier (2010). Heads are marked using the head finding rules for the Negra corpus used by the Stanford parser. The markovization setting is v=1 (i.e., no parent annotation), and $h \in \{1, 2, \infty\}$, dictated by efficiency concerns. Lower values for h give better performance because they allow more flat structures to be covered through re-combinations of parts of different constituents. However, this also greatly increases the number of possible edges which have to be explored. For this reason the value of h had to be increased for parsing longer sentences, at the cost of decreased performance and coverage. Table 4 lists the size of the training & test corpora and their grammars for the respective length restrictions.[41] Unparsed sentences are assigned a baseline parse with all tags directly under the root node. All scores were obtained with Maier's publicly available implementation, `rparse`.[42]

The model performs consistently better than previous results on discontinuous parsing; see table 5 for the results, including comparisons to previous work, and figure 20 for a graph plotting the number of words against f-score. Figure 21 plots the time required to parse sentences of different lengths with v=1 h=2, showing a strikingly steep curve, which makes clear why parsing sentences longer than 25 words was not feasible with these settings. The coarse-to-fine inference appears to work rather well, apparently displaying a linear observed time complexity on the dop grammar; unfortunately exhaustive parsing with the coarse grammar forms a bottleneck. The total time to parse was 1, 16 and 52 hours for 15, 25, and 30 words respectively, using about 4 gb of memory.

Table 6 shows results with tree-distance evaluation, comparing dop against our baseline plcfrs results. Although Maier (2010) also reports results with tree-distance evaluation, his results were obtained by leaving out unparsed sentences. This makes the results incomparable across parsers. A parser that gets only a single sentence right and leaves the rest unparsed would get a perfect score, under this methodology. Assigning a default tree, as we do, is necessary to obtain a fair evaluation.

Note also that the baseline plcfrs results improve on those of Kallmeyer

40 This is necessary because punctuation is not part of the trees in the Negra annotation, and is instead attached directly to the root node, which would imply spurious discontinuities if the punctuation is part of the grammar. Another solution would be to ignore punctuation altogether, which has some linguistic plausibility and would reduce data sparsity.

41 Note that Kallmeyer and Maier (2010) apply the length restriction before the 90-10 split, but the difference is not more than 12 sentences.

42 Cf. `http://www.wolfgang-maier.net/rparse`

and Maier—except for the result up to 15 words, but it is not clear whether they also restricted the training corpus to 15 words as I did. I hypothesize this striking difference is caused by any or all of the following reasons:

1. The version of `rparse` used in Maier (2010); Kallmeyer and Maier (2010) did not correctly perform the DECREASE-KEY operation;[43]
2. the detrimental effect of context-summary estimates (i.e., not finding the Viterbi derivation);
3. their binarized grammars include two levels of fan-out markers (cf. section 3.4);
4. other, more incidental differences in binarization, particularly markovization.

Furthermore, table 6 also contains results with the CFG-CTF method. These are the first results of LCFRS or discontinuous constituency parsing for sentences up to 40 words. Since the CFG-CTF method allows the use of the most optimal Markovization setting (h=1), the results are quite good.

Table 7 shows results on part of the Tepacoc test set. The categories are extraposed relative clauses (ERC), forward conjunction reduction (FCR), noun PP attachment (PPN), verb PP attachment (PPV), coordination of unlike constructions (CUC) and subject gap with finite/fronted verbs (SGF). While the PLCFRS results mostly improve on those of Maier (2010), the results for DOP are not impressive. Kübler et al. use the first 25,005 sentences in Tiger as training set (because their goal is to compare Tiger against Tüba-D/Z). Maier (2010) apparently uses 90% of Tiger sentences up to 30 words (31,568 sentences) in his evaluation on Tepacoc. I have opted to stick to the set-up of Kübler et al., using the first 25,005 sentences for training, without length restrictions. With v=1, h=∞ binarization, it took 36 days[44] to parse these 76 sentences. It is likely that the combination of complex constructions with h=∞ markovization is especially unfortunate, as these constructions may require rare productions that could have been composed of others with markovization.

words	train	test	PLCFRS rules	Disco-DOP rules	fan-out	parsing complexity
$\leq$ 15	9025	1015	24020	678659	5	11
$\leq$ 25	14870	1639	53773	1769507	7	15
$\leq$ 30	16490	1845	50381	1799797	7	15
words	train	dev	PCFG	Disco-DOP	fan-out	p.c.
$\leq$ 40	17988	968	59862	2659940	1 / 9	3 / 19

Table 4: Number of sentences and rules for the grammars extracted from the Negra corpus. The number of rules and fan-outs are for binarized grammars.

43 This shortcoming has since been addressed (Maier, personal communication).
44 Turning off the length restriction for the training corpus turned out to be a costly decision.

NEGRA	words	LP	LR	F_1	EX
Plaehn (2004): DPSG	≤ 15	73.61	72.72	73.16	39.0
Kallmeyer and Maier (2010): PLCFRS	≤ 15	-	-	81.27	-
This work: Disco-DOP, v=1 h=1	≤ 15	84.26	85.03	84.65	55.17
Kallmeyer and Maier (2010): PLCFRS	≤ 25	73.03	73.46	73.25	-
This work: Disco-DOP, v=1 h=2	≤ 25	77.63	79.12	78.37	39.11
Maier (2010): PLCFRS	≤ 30	72.39	70.68	71.52	31.65
This work: Disco-DOP, v=1 h=∞	≤ 30	72.69	74.42	73.54	32.57
TIGER	words	LP	LR	F_1	EX
This work: PLCFRS	≤ 15	81.91	78.87	80.36	45.77
This work: Disco-DOP	≤ 15	84.09	84.03	84.06	50.07
This work: PLCFRS	≤ 20	76.84	75.02	75.92	36.12
This work: Disco-DOP	≤ 20	82.08	81.54	81.81	43.38

Table 5: Results for discontinuous parsing on two different corpora.

NEGRA	words	Dice	Jaccard	F_1	EX
PLCFRS, v=1 h=1	≤ 15	92.11	85.30	81.10	49.06
Disco-DOP, v=1 h=1	≤ 15	93.53	87.79	84.56	54.68
PLCFRS, v=1 h=2	≤ 25	89.74	81.21	75.98	36.79
Disco-DOP, v=1 h=2	≤ 25	90.99	83.33	78.81	39.60
PLCFRS, v=1 h=∞	≤ 30	87.23	76.93	72.34	31.27
Disco-DOP, v=1 h=∞	≤ 30	87.62	77.57	73.98	34.96
Split-PCFG,† v=1 h=1	≤ 40	87.24	77.21	68.29	27.69
Disco-DOP,† CFG-CTF, v=1 h=1	≤ 40	89.24	80.47	74.21	32.54

Table 6: Tree-distance evaluation. The DOP results were obtained with EWE.
† This result was obtained on the development set according to the common training-test split introduced in Dubey and Keller (2003).

		PLCFRS		Disco-DOP	
Category	≤ 30	F_1	EX	F_1	EX
ERC	11 of 20	67.57	9.09	67.84	0
FCR	18 of 20	76.32	16.67	80.33	44.44
PPN	9 of 10	72.96	22.22	71.08	0
PPV	8 of 10	78.91	50.00	72.85	25.00
CUC	14 of 20	59.35	0	57.92	0
SGF	16 of 20	77.96	25.00	73.85	6.25
total	76 of 100	72.34	17.11	71.35	17.11

Table 7: Results on the TePaCoC test set. These results were obtained with v=1 h=∞ binarization.

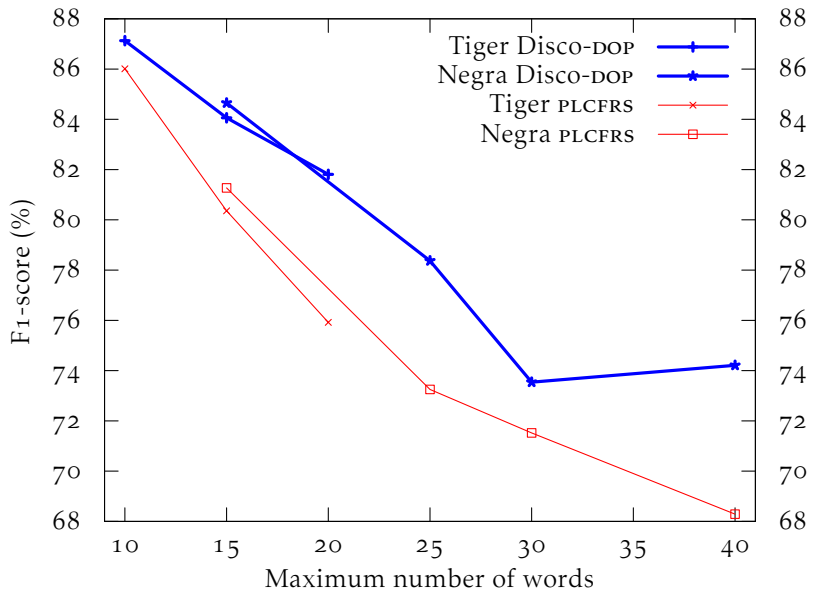

Figure 20: F-score as a function of the number of words

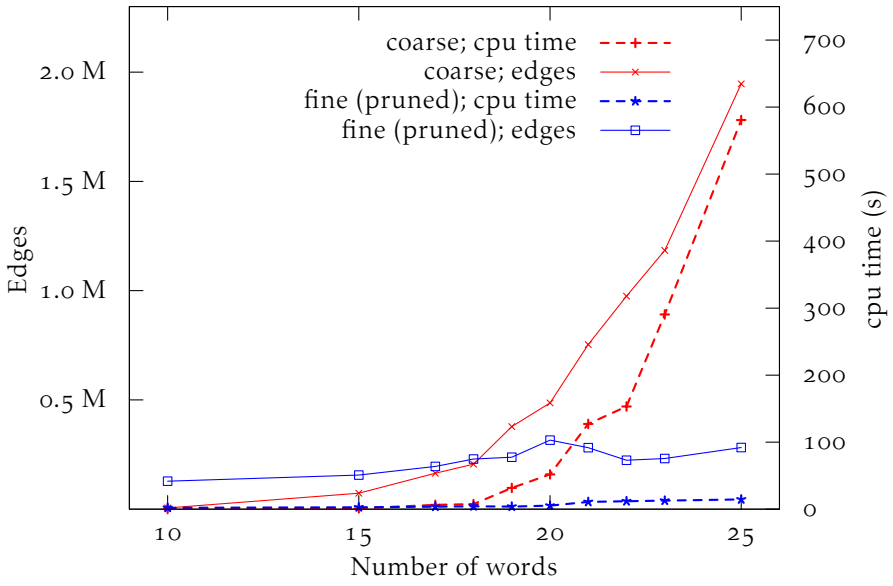

Figure 21: Efficiency as a function of the number of words in the coarse (PLCFRS) and the fine stage (Disco-DOP). The data are from parsing 10 Negra sentences, hand-picked to illustrate the worst case.

Chapter 8

Remaining challenges

The current work has limitations: it fails to model grammatical functions, morphology, and word-order variation.

For reasons of efficiency the work in this thesis is limited to a mildly context-sensitive formalism. This provides a strong improvement in terms of the representations that are produced by the parser, because these representations can directly express argument structure. On the other hand there are two ways in which generalizations are blocked in this formalism. The first is that every non-terminal has a fixed fan-out; categories that occur with different fan-outs in the corpus must be split up, e.g., NP_2 and NP_3 are distinct to the formalism, both in terms of rules and probabilities, while in the corpus they have the same category. The second limitation is that the order of components of each non-terminal in each particular rule is fixed—the rule represents the precise configuration of its original constituent. There are two reasons to work with these limitations: parsing efficiency and to keep the process of reading off rules straightforward, viz., to maintain an isomorphism between treebank and grammar rules.

However, this problem could be alleviated if we could separate the probabilistic model from the formalism. Levy (2005, ch. 4.8) suggests a factorization of probabilities into components for immediate dominance and linear precedence, following ideas from generalized phrase-structure grammar (GPSG). Immediate dominance expresses constraints (or in this case, probabilities) on the parent-child relationship, while linear precedence expresses the possible orderings of terminals in the sentence, which potentially introduces discontinuity. Levy suggests conditioning the probabilities for linear precedence on the sizes of other constituents, because this makes it possible to model distance sensitivity; e.g., long distance discontinuities can receive lower probabilities. Unfortunately this means that probabilities can only be assigned to completed constituents (i.e., when gaps are filled), such that all manner of optimizations based on local probabilities are ruled out, and without that we would be forced to enumerate an exponential space of derivations. Levy admits that "this may seem an ill-formed idea" at first glance; I tentatively conclude that it is—at least for (generative) parsing. It is a perfectly sensible idea for generation, where a particular constituency tree can be assumed to have been fixed independently from its surface realization.

Another problem with the approach is that there is no longer a straightforward way to read off the rules from the treebank. It is straightforward to estimate the immediate dominance probabilities, by turning the right-hand side of each rule into a multiset and ignoring the yield functions. The estimation

of linear precedence is difficult because it must assume an implicit, canonical tree of which the tree in the treebank is a permutation. Concretely, given a VP with fan-out 2, we can either assume its second half was moved to the right, or conversely that the first half was moved to the left, with respect to the canonical tree. It is conceivable that, given a simple model of canonical structures, perhaps based on all trees without discontinuities in the treebank, probabilities for certain types of movement can be estimated. Without distance sensitivity, it would not affect the complexity of parsing while it may increase its stochastic power, because categories that only differ in fan-out can share probability mass. Once the approach is in place it can be extended to DOP by factoring the probability of each fragment into immediate dominance and linear precedence, and redistributing the probability mass for immediate dominance among all fragments with a certain dominance structure.

However, there is a limit to what can be done with linear context-free rewriting systems (and consequently, mildly context-sensitive formalisms). It turns out that scrambling, a form of word-order freedom, cannot be expressed in an LCFRS (Becker et al., 1992). Boullier (1999) shows that in a full Range Concatenation Grammar, scrambling can be expressed and parsed efficiently. Compared to the grammars in this work, the yield functions in the given grammar are not ordered, not linear, and the grammar includes negative clauses (these implement the concept of negation-as-failure from logic programming). Boullier even shows that scrambling can be parsed in quadratic time. However, the given grammar is carefully crafted, including auxiliary predicates to match only ranges of length one. Inducing a scrambling grammar from a treebank is a different matter, let alone maintaining quadratic time complexity.

An issue with the current system is that the markovization is rather ad-hoc. Markovization of an optimal head-driven binarization (Crescenzi et al., 2011) is an even more ill-formed idea, because it covers non-terminals in an arbitrary order, such that whether X follows Y no longer has any meaning in the resulting statistics. The idea of including a certain amount of local context in node labels conflicts with the non-local nature of discontinuous nodes. A much more elegant alternative would be to apply a tree-insertion grammar. Such a grammar has been implemented for DOP (Hoogweg, 2000). Concretely, this model extends DOP with an insertion operation. The model has two kinds of fragments. Initial fragments contain heads whose arguments can be added through substitution. Auxiliary fragments contain adjuncts (roughly, optional items) which can be inserted into derivations; e.g., new adjectives can be added to an NP fragment. In other words, instead of incorporating ad-hoc generalizations through non-terminal labels, adjunction is a first-class citizen of the model, and can be modeled probabilistically. Hoogweg had to induce the set of base fragments and those which can be added as adjunctions heuristically, but the Tiger corpus includes edge labels which indicate whether something is a complement or an adjunction.

Aside from these technical issues, many linguistic features have been glossed over in this work to limit its scope. Assigning part-of-speech tags to words is normally part of the parsing task and integrated with the probabilistic model, while in this work the correct tags from the corpus are given to the parser. Existing approaches can be employed to address this. Known words can be parsed directly by the grammar, and unknown words receive all possible tags for open class words, where their probability is based on the frequency of words

occurring once (in the spirit of the Good-Turing smoothing method). Instead of choosing one tag per word before starting to parse, each possible tag will be treated as any other constituent, i.e., as an element competing to be part of the globally optimal derivation.

A proper parser and evaluation should work with grammatical functions as well, however simply adding functions to phrasal labels introduces sparsity issues (Rafferty and Manning, 2008). Parsing languages with less strict word-order implies that morphology provides important information about constituents that have been moved or extraposed. Movement and extraposition could also be modeled statistically, which can reduce data sparsity.

One answer to this is Relational-Realizational parsing (Tsarfaty, 2010). Although the model is formally represented in a PCFG, it integrates grammatical functions, morphology and word order in a generative, probabilistic model. This is achieved by parsing in two stages. The first stage is relational, where a clause projects a number of grammatical functions to its words. The second is realizational, which realizes word order and morphological markings in a surface form. The model improves on a plain PCFG, for parsing Modern Hebrew. What is most interesting, however, is that it integrates several important linguistic phenomena in a single, statistical model. This opens the door to the prospect of parsing a variety of typologically diverse languages with success to match that for English. Extending this model to discontinuous representations and to a data-oriented model is an interesting topic for future research.

Chapter 9

Conclusion

To recapitulate, discontinuity and data-oriented parsing can be combined fruitfully, but much remains to be done.

A DATA-ORIENTED model of discontinuous phrase structure has been presented which outperforms all previously published results. This has been achieved by combining a variety of techniques. To wit, a linear context-free rewriting system as the symbolic grammar, data-oriented parsing as the probabilistic framework, a general method for enumerating k-best derivations from a chart, and a coarse-to-fine optimization to tame the computational complexity of DOP.

It turns out that using a grammar formalism with a parsing complexity that is well beyond cubic is not an impediment for making a DOP model with considerably better performance. Most technical issues with discontinuous DOP parsing have been addressed in this thesis—the next step will be to introduce linguistic improvements.

∞

Bibliography

Bach, Emmon, Colin Brown, and William Marslen-wilson (1986). Crossed and nested dependencies in German and Dutch: A psycholinguistic study. *Language and Cognitive Processes*, 1(4):249–262.

Bansal, Mohit and Dan Klein (2010). Simple, accurate parsing with an all-fragments grammar. In *Proc. of ACL*, pages 1098–1107.

Barthélemy, François, Pierre Boullier, Philippe Deschamp, and Éric de la Clergerie (2001). Guided parsing of range concatenation languages. In *Proc. of ACL*, pages 42–49.

Becker, Tilman, Owen Rambow, and Michael Niv (1992). The derivational generative power of formal systems or scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for research in cognitive science. Available from: `ftp://ftp.cis.upenn.edu/pub/ircs/tr/92-38.ps.Z`.

Behnel, Stefan, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith (2011). Cython: The best of both worlds. *Computing in Science and Engineering*, 13:31–39.

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python*. O'Reilly Media.

Black, Ezra, John Lafferty, and Salim Roukos (1992). Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proc. of ACL*, pages 185–192.

Bod, Rens (1992). A computational model of language performance: Data-oriented parsing. In *Proceedings COLING*, pages 855–859.

——— (1995). The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of EACL*, pages 104–111. Available from: `http://aclweb.org/anthology/E/E95/E95-1015.pdf`.

——— (2000). Parsing with the shortest derivation. In *Proceedings of COLING*, pages 69–75.

——— (2001). What is the minimal set of fragments that achieves maximal parse accuracy? In *Proc. of ACL*, pages 69–76.

——— (2003). An efficient implementation of a new DOP model. In *Proceedings of EACL*, volume 1, pages 19–26. Available from: `http://www.ldc.upenn.edu/acl/E/E03/E03-1005.pdf`.

Bolz, Carl F., Antonio Cuni, Maciej Fijalkowski, and Armin Rigo (2009). Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pages 18–25. ACM.

Bonnema, Remko, Paul Buying, and Remko Scha (1999). A new probability model for data oriented parsing. In *Proceedings of the 12th Amsterdam Colloquium*, pages 85–90.

Boullier, Pierre (1998). Proposal for a natural language processing syntactic backbone. Technical Report RR-3342, INRIA-Rocquencourt, Le Chesnay, France. Available from: `http://www.inria.fr/RRRT/RR-3342.html`.

——— (1999). Chinese numbers, mix, scrambling, and range concatenation grammars. In *Proceedings of EACL*, pages 53–60.

Boyd, Adriane (2007). Discontinuity revisited: An improved conversion to context-free representations. In *Proceedings of the Linguistic Annotation Workshop*, pages 41–44.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith (2002). The Tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, pages 24–41.

Bresnan, Joan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen (1982). Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13(4):613–635.

Bunt, Harry (1996). Formal tools for describing and processing discontinuous constituency structure. In Bunt, Harry C. and Arthur van Horck, editors, *Discontinuous Constituency*, pages 63–83. Walter de Gruyter.

Bybee, Joan L. (2007). From usage to grammar: The mind's response to repetition. *Language*, 82(4):711–733.

Charniak, Eugene, Mark Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, et al. (2006). Multilevel coarse-to-fine PCFG parsing. In *Proceedings of NAACL-HLT*, pages 168–175.

Chomsky, Noam and Marcel P. Schützenberger (1963). The algebraic theory of context-free languages. In Braffort, P. and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier. Available from: `http://www.sciencedirect.com/science/article/pii/S0049237X08720238`.

Cohn, Trevor, Sharon Goldwater, and Phil Blunsom (2009). Inducing compact but accurate tree-substitution grammars. In *Proceedings of NAACL-HLT*, pages 548–556.

Collins, Michael (1999). *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania.

Collins, Michael and T. Koo (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Cormen, Thomas H., Charles E. Leiserson, Ron L. Rivest, and Clifford Stein (2001). *Introduction to algorithms*. MIT press, second edition.

Crescenzi, P., Daniel Gildea, A. Marino, G. Rossi, and Giorgio Satta (2011). Optimal head-driven parsing complexity for linear context-free rewriting systems. In *Proc. of ACL*.

Dowty, David R. (1996). Toward a minimalist theory of syntactic structure. In Bunt, Harry C. and Arthur van Horck, editors, *Discontinuous Constituency*, pages 10–62. Walter de Gruyter.

Dubey, Amit and Frank Keller (2003). Parsing german with sister-head dependencies. In *Proc. of ACL*, pages 96–103.

Dufour, Mark (2006). Shed skin: An optimizing Python-to-C++ compiler. Master's thesis, Delft University of Technology. Available from: `http://mark.dufour.googlepages.com/shedskin.pdf`.

Emms, Martin (2008). Tree-distance and some other variants of evalb. In *Proc. of LREC*, pages 1373–1379.

Gildea, Daniel (2010). Optimal parsing strategies for linear context-free rewriting systems. In *Proceedings of NAACL HLT 2010.*, pages 769–776.

Gómez-Rodríguez, Carlos, Marco Kuhlmann, Giorgio Satta, and David Weir (2009). Optimal reduction of rule length in linear context-free rewriting systems. In *Proceedings of NAACL HLT 2009*, pages 539–547.

Gómez-Rodríguez, Carlos, David Weir, and John Carroll (2009). Parsing mildly non-projective dependency structures. In *Proceedings of EACL*, pages 291–299.

Goodman, Joshua (1996). Efficient algorithms for parsing the DOP model. In *Proceedings of EMNLP*, pages 143–152.

Goodman, Joshua (2003). Efficient parsing of DOP with PCFG-reductions. In Bod, Rens, Remko Scha, and Khalil Sima'an, editors, *Data-Oriented Parsing*. The University of Chicago Press.

Groenink, Annius V. (1995). Literal movement grammars. In *Proceedings of EACL*, pages 90–97. Available from: `http://aclweb.org/anthology/E/E95/E95-1013.pdf`.

——— (1997a). Mild context-sensitivity and tuple-based generalizations of context-grammar. *Linguistics and Philosophy*, 20(6):607–636. Available from: `http://www.jstor.org/stable/25001685`.

——— (1997b). *Surface Without Structure*. PhD thesis, University of Utrecht.

Harman, Gilbert H. (1963). Generative grammars without transformation rules: a defense of phrase structure. *Language*, 39(4):597–616. Available from: `http://www.jstor.org/stable/411954`.

Hoogweg, Lars (2000). Extending DOP1 with the insertion operation. Master's thesis, University of Amsterdam.

Huang, Liang and David Chiang (2005). Better k-best parsing. In *Proceedings of IWPT 2009*, pages 53–64.

Johnson, Mark (1985). Parsing with discontinuous constituents. In *Proc. of ACL*, pages 127–132.

Johnson, Mark (2002). The DOP estimation method is biased and inconsistent. *Computational Linguistics*, 28(1):71–76.

Joshi, Aravind K. (1985). How much context sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars. In Dowty, David R., Lauri Karttunen, and Arnold M. Zwicky, editors, *Natural language parsing: Psychological, computational and theoretical perspectives*, pages 206–250. Cambridge University Press, New York.

Kallmeyer, Laura (2010). Multiple context-free grammars and linear context-free rewriting systems. In *Parsing Beyond Context-Free Grammars*, Cognitive Technologies, pages 109–129. Springer Berlin Heidelberg. Available from: `http://dx.doi.org/10.1007/978-3-642-14846-0_6`.

Kallmeyer, Laura and Wolfgang Maier (2009). An incremental earley parser for simple range concatenation grammar. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 61–64.

Kallmeyer, Laura and Wolfgang Maier (2010). Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 537–545.

Kanazawa, Makoto (2007). Parsing and generation as datalog queries. In *Proc. of ACL*, pages 176–183.

_____ (2011). Parsing and generation as datalog query evaluation. Unpublished manuscript. Available from: `http://research.nii.ac.jp/~kanazawa/publications/pagadqe.pdf`.

Klein, Dan and Christopher D. Manning (2003a). Accurate unlexicalized parsing. In *Proc. of ACL*, volume 1, pages 423–430.

Klein, Dan and Christopher D. Manning (2003b). Fast exact inference with a factored model for natural language parsing. *Advances in neural information processing systems*, pages 3–10.

Kolb, Hans-Peter, Uwe Mönnich, and Frank Morawietz (2000). Descriptions of cross-serial dependencies. *Grammars*, 3(2):189–216.

Kübler, Sandra, Wolfgang Maier, Ines Rehbein, and Y. Versley (2008). How to compare treebanks. In *Proceedings of LREC*, volume 8.

Kübler, Sandra, Ines Rehbein, and Josef van Genabith (2009). TePaCoC: a corpus for testing parser performance on complex German grammatical constructions. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories. Groningen, The Netherlands*.

Kuhlmann, Marco and Giorgio Satta (2009). Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of EACL*, pages 478–486.

Lange, Martin; Leiß, Hans (2009). To cnf or not to cnf? an efficient yet presentable version of the cyk algorithm. *Informatica Didactica*, 8.

Levy, Roger (2005). *Probabilistic models of word order and syntactic discontinuity*. PhD thesis, Stanford University.

Liang, Percy, Slav Petrov, M. Jordan, and Dan Klein (2007). The infinite PCFG using hierarchical Dirichlet processes. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 688–697.

Lin, Dekang (1995). A dependency-based method for evaluating broad-coverage parsers. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1420–1427.

Maier, Wolfgang (2010). Direct parsing of discontinuous constituents in German. In *Proceedings of the SPMRL workshop at NAACL HLT 2010*, pages 58–66.

Maier, Wolfgang and Laura Kallmeyer (2010). Discontinuity and non-projectivity: Using mildly contextsensitive formalisms for data-driven parsing. In *Proceedings of TAG*, volume 10.

Maier, Wolfgang and Anders Søgaard (2008). Treebanks and mild context-sensitivity. In *Proceedings of Formal Grammar 2008*, page 61.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of statistical natural language processing*. MIT Press.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii (2005). Probabilistic CFG with latent annotations. In *Proc. of ACL*, pages 75–82.

Nederhof, Mark-Jan (2003). Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

Nguyen, Thuy Linh (2004). Rank consistent estimation: The DOP case. Master's thesis, University of Amsterdam. Available from: `http://www.science.uva.nl/pub/theory/illc/researchreports/MoL-2004-06.text.pdf`.

O'Donnell, Timothy J., Noah D. Goodman, Jesse Snedeker, and Joshua B. Tenenbaum (2009). Computation and reuse in language. In *Proceedings of the Cognitive Science Society*, Amsterdam, The Netherlands.

Pauls, Adam and Dan Klein (2009). K-best A* parsing. In *Proc. of ACL*, volume 2, pages 958–966.

Plaehn, Oliver (2004). Computing the most probable parse for a discontinuous phrase structure grammar. In Bunt, Harry, John Carroll, and Giorgio Satta, editors, *New developments in parsing technology*, pages 91–106. Kluwer Academic Publishers, Norwell, MA, USA.

Pollard, Carl (1984). *Generalized Phrase Structure Grammars, Head Grammars, and Natural Languages*. PhD thesis, Stanford.

Post, Matt and Daniel Gildea (2009). Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48.

Prescher, Detlef, Remko Scha, Khalil Sima'an, and Andreas Zollmann (2003). On the statistical consistency of DOP estimators. In *Proceedings CLIN 2003*. Available from: `http://www.cnts.ua.ac.be/clin2003/proc/07Prescher.pdf`.

Rafferty, Anna N. and Christopher D. Manning (2008). Parsing three German treebanks: Lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, pages 40–46.

Rehbein, Ines and Josef van Genabith (2007). Evaluating evaluation measures. In *NODALIDA 2007 Conference Proceedings*, pages 372–379. Available from: `http://doras.dcu.ie/15237`.

Sampson, Geoffrey (2000). A proposal for improving the measurement of parse accuracy. *International Journal of Corpus Linguistics*, 5(1):53–68.

Sangati, Federico and Willem Zuidema (2011). Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of EMNLP*, pages 84–95. Available from: `http://www.aclweb.org/anthology/D11-1008`.

Sangati, Federico, Willem Zuidema, and Rens Bod (2010). Efficiently extract recurring tree fragments from large treebanks. In *Proceedings of the 7th international conference on Language Resources and Evaluation (LREC'10)*, pages 219–226. European Language Resources Association (ELRA). Available from: `http://dare.uva.nl/record/371504`.

Sapir, Edward (1921). *Language: An introduction to the study of speech*. Harcourt, Brace and company, New York.

Scha, Remko (1990). Language theory and language technology; competence and performance. In de Kort, Q.A.M. and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere, the Netherlands. Original title: *Taaltheorie en taaltechnologie; competence en performance*. Translation available at `http://iaaa.nl/rs/LeerdamE.html`.

Schmid, Helmut (2004). Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of COLING '04*. Available from: `http://aclweb.org/anthology/C/C04/C04-1024`.

Shieber, Stuart M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Shieber, Stuart M., Yves Schabes, and Fernando C.N. Pereira (1995). Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36.

Sima'an, Khalil (1999). *Learning efficient disambiguation*. PhD thesis, Utrecht University and University of Amsterdam.

_____ (2000). Tree-gram parsing lexical dependencies and structural relations. In *Proc. of ACL*, pages 37–44.

_____ (2002). Computational complexity of probabilistic disambiguation. *Grammars*, 5(2):125–151.

Sima'an, Khalil and Luciano Buratto (2003). Backoff parameter estimation for the DOP model. *Machine Learning: ECML 2003*, pages 373–384. Available from: http://dare.uva.nl/record/126078.

Skut, Wojciech, Brigitte Krenn, Thorten Brants, and Hans Uszkoreit (1997). An annotation scheme for free word order languages. In *Proceedings of ANLP*, pages 88–95.

Telljohann, Heike, Erhard Hinrichs, and Sandra Kübler (2004). The tüba-d/z treebank: Annotating german with a context-free backbone. In *In Proceedings of LREC*, pages 2229–2235.

Trautwein, Marten H. (1995). *Computational pitfalls in tractable grammar formalisms*. PhD thesis, University of Amsterdam.

Tsarfaty, Reut (2010). *Relational-realizational parsing*. PhD thesis, University of Amsterdam. Available from: http://dare.uva.nl/record/335795.

van Noord, Gertjan (1991). Head corner parsing for discontinuous constituency. In *Proc. of ACL*, pages 114–121.

Vijay-Shanker, K. and David J. Weir (1994). The equivalence of four extensions of context-free grammars. *Theory of Computing Systems*, 27(6):511–546.

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of ACL*, pages 104–111.

Villemonte de la Clergerie, Éric (2002). Parsing mildly context-sensitive languages with thread automata. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7.

Vogel, Carl, Ulrike Hahn, and Holly Branigan (1996). Cross-serial dependencies are not hard to process. In *Proceedings of the 16th conference on Computational linguistics*, volume 1 of *COLING '96*, pages 157–162, Stroudsburg, PA, USA. Association for Computational Linguistics. Available from: http://dx.doi.org/10.3115/992628.992658.

Waxmonsky, Sonjia and I. Dan Melamed (2006). A dynamic data structure for parsing with discontinuous constituents. Proteus Project Technical Report 06–001, New York University.

Weir, David J. (1988). *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis, University of Pennsylvania. Available from: http://repository.upenn.edu/dissertations/AAI8908403/.

Yngve, Victor (1960). A model and an hypothesis for language structure. *Proceedings of the American Philosophical Society*, 104(5):444–466.

Younger, Daniel H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208.

Zhang, Kaizhong and Dennis Shasha (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262.

Zollmann, Andreas (2004). A consistent and efficient DOP estimator. Master's thesis, University of Amsterdam. Available from: `http://www.science.uva.nl/pub/theory/illc/researchreports/MoL-2004-02.text.pdf`.

Zollmann, Andreas and Khalil Sima'an (2005). A consistent and efficient estimator for Data-Oriented Parsing. *Journal of Automata Languages and Combinatorics*, 10(2/3):367. Available from: `http://staff.science.uva.nl/~simaan/D-Papers/JALCsubmit.pdf`.

Zuidema, Willem (2006). Theoretical evaluation of estimation methods for data-oriented parsing. In *Proceedings of EACL*, pages 183–186.

_____ (2007). Parsimonious data-oriented parsing. In *Proceedings of EMNLP-CoNLL*, pages 551–560.