Electronic Notes in Theoretical Computer Science 53 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume53.html 17 pages

Optimality parsing and local cost functions in Discontinuous Grammar

Matthias Trautner Kromann [1,2]

Department of Computational Linguistics
Copenhagen Business School
Copenhagen, Denmark

Abstract

After a brief survey of Discontinuous Grammar (DG), we propose local cost functions as a generalization of violable constraints in OT, probabilities in PCFG, and processing times in distributed competition models. We demonstrate how local cost functions can be used in DG to encode violable constraints on word order, landing sites, islands, agreement, and selectional restrictions, as well as lexical frequencies and pragmatic preferences.

Rephrasing parsing as an optimization problem in which a large space of partial parses is searched for a minimum-cost solution, we propose a heuristic parsing algorithm for DG based on local search, with worst-case complexity $O(n \log^4 n)$ given linguistically reasonable assumptions on tree depth and island constraints, and $O(n^5)$ without any assumptions. The proposed algorithm works in the presence of local ambiguity, produces the right analysis for a wide range of discontinuous word-order phenomena such as topicalizations, relativizations, extractions, scramblings, and discontinuous APs, and fails on garden-path sentences, just like humans.

1 Discontinuous Grammar

Discontinuous Grammar [10] (or DG) is a syntax formalism that was originally conceived as a synthesis between Head-Driven Phrase Structure Grammar [15], Government & Binding [5], and German dependency grammar [6]. Subsequently, it has borrowed ideas from Optimality Theory [8], Tree-Adjoining Grammar [1],

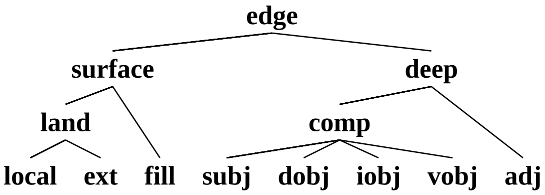
[1] Email: mtkromann@ieee.org. Web: www.id.cbs.dk/∼mtk/dg

[2] I wish to thank my thesis advisor, Professor Carl Vikner, to whom this paper is dedicated. Thanks also to Richard Hudson, Don C. Mitchell, Owen Rambow, Lars Konieczny, Line Mikkelsen, Sten Vikner, Sabine Kirchmeyer-Andersen, and Alex Klinge for fruitful discussions, and to two anonymous reviewers for highly valuable comments. This work was made possible by a grant from the Danish Research Council for the Humanities.

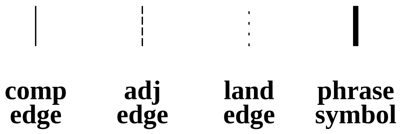
Word Grammar [7], as well as psycholinguistics [4]. Through these common origins, DG also shares some ideas with Lexical-Functional Grammar [3] and newer dependency-based theories [9].

1.1 Syntax graphs in DG

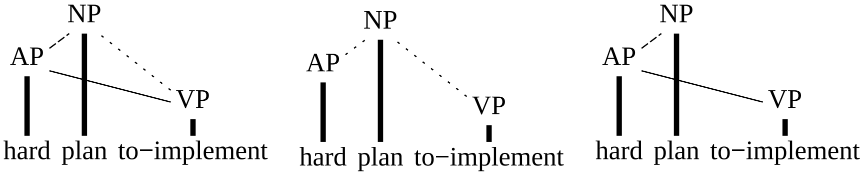
In Discontinuous Grammar, syntax graphs are acyclic and directed, with typed nodes and edges that represent words and grammatical relations. Nodes and edges are organized in an inheritance hierarchy, shown partly below for edges.



When drawing syntax graphs, edge types are usually not specified. Instead, the main edge type is indicated with one of the following arrows:



An example of a DG graph is shown below. For notational convenience, DG graphs are drawn like phrase-structure graphs, with the phrase symbol drawn above each word (although formally, the phrase symbol and the word are just one node). Moreover, phrases like “to implement” will sometimes be drawn as simple units, if the internal structure is irrelevant to the discussion.



DG graph **Surface tree** **Deep tree**

Each node in the graph (except the root) has a *landing site* encoded by a surface edge, and a *governor* encoded by a deep edge. The node must be licensed as a *landed node* of its landing site, and as a *dependent* of its governor. Dependents are either *complements*, which are selected by their governors and act as semantic arguments to them, or *adjuncts*, which select their governors and act as functors to them. For the graph to be well-formed, surface edges must form a continuous *surface tree*, deep edges must form a possibly discontinuous *deep tree*, and the two trees must satisfy an *upwards deep movement principle* that requires a word’s landing site to dominate the word's governor in the deep tree. [3]

1.2 The lexicon in DG

In the lexicon, each type is defined by a lexical entry that specifies an ordered list of its immediate super types, and declares its variables. Variable declarations and values are inherited by default from super types. Variables are marked as either *atomic* or *set-valued*. The value of an atomic variable is either specified locally, or inherited from the first super type in which it is defined (ie, by *default priority inheritance*). The value of a set-valued variable can be locally specified using different operators: $v = A$ is used to set the value of variable v to the set A, whereas $v = +A - B$ sets the value of v to set A plus all values for v in super types minus set B.

In the simplified example below, *verb* is defined as subtype of type *word* with variable *comps* having set value *[subj:noun]* (that is, a verb licenses by default a complement of type *noun* with edge type *subject*). *Eat* is defined as subtype of *verb* and *pres* (present tense), and its value for *comps* is *[subj:noun, dobj:noun]* after inheritance (that is, it also takes a noun as direct object). *Rains* overrides the inheritance by setting *comps* to *[subj:it]*.

```
type(verb, [word], [comps = [subj:noun]]).
type(pres, [flex], []).
type(eat, [verb,pres], [comps = +[dobj:noun]]).
type(rains, [verb,pres], [comps = [subj:it]]).
```

If s and t are types, $s * t$ is called the *join* of s and t, and is defined as the type that has s and t as its super types, with no local variable specifications. *Type specifications* are either type names, or composites of the form $\sigma + \tau$, $\sigma - \tau$, $-\tau$, and $\sigma \mid \tau$ where σ and τ are type specifications. The operator $\mathrm{isa}(t, \sigma)$ is used to test whether a type t satisfies a type specification. It is defined recursively by:

$\mathbf{isa}(\mathbf{t}, \mathbf{s})$ if type s dominates type t in the type hierarchy
$\mathbf{isa}(\mathbf{t}, \sigma + \tau)$ if $\mathrm{isa}(t, \sigma)$ and $\mathrm{isa}(t, \tau)$
$\mathbf{isa}(\mathbf{t}, \sigma - \tau)$ if $\mathrm{isa}(t, \sigma)$, but not $\mathrm{isa}(t, \tau)$
$\mathbf{isa}(\mathbf{t}, \sigma \mid \tau)$ if $\mathrm{isa}(t, \sigma)$ or $\mathrm{isa}(t, \tau)$

Each word and word class must list the complement structures, adjunct governors, landed nodes, and cost functions it licenses. A simplified example is shown below for the adjective *hard*, which states that it accepts a phrase headed by *for* as prepositional object, a phrase headed by infinitival *to* as verbal object, can attach to a noun as an *amod* adjunct, only accepts its own local dependents as landed nodes, and has a cost function that punishes left landed complements as highly ungrammatical.

[3] In *surface movement*, a word's landing site must dominate the word's governor in the surface tree. This is more restrictive than deep movement, since the surface tree can be seen as a flattened version of the deep tree. Slash-propagation in classical HPSG and movement in GB correspond to surface movement. Interestingly, Bouma, Malouf, and Sag [2, §3.1] have recently proposed changing slash-propagation within HPSG so that it corresponds to deep movement.

```
type(hard, [adjective], [
        comps = [pobj:for, vobj:to_inf],
        adjs = [amod:noun],
        land = [word+local],
        costs = [1000 * left(comp)]]).
```

For simplicity, we ignore how the lexicon encodes ambiguity, idioms, semantics of complement and adjunct structures, and licensed fillers (phonetically empty words used to satisfy double dependencies such as relativized roles or shared subjects).

2 Cost functions

Like Optimality Theory, DG uses violable constraints in order to make parsing more robust to speech errors and unexpected constructions. However, DG constraints differ from OT constraints by having a non-negative *cost* rather than a rank, and by being *localized*, ie, associated with specific nodes in the graph. Another difference is that the weighted constraints in DG are specified in a well-defined constraint language which can express a wide range of linguistic constraints, but ensures computational efficiency by requiring all constraints to be locally computable from a node's nearest neighbours in the graph. The fact that most linguistic constraints can be computed locally is an important consequence of DG's design, where a node's governor, landing site, complements, adjuncts, and landed nodes constitute its neighbours.

2.1 Operator definitions

DG constraints are defined in terms of *cost functions*, which return a cost that measures the number of times a syntactic condition is violated, weighed with its severity. Cost functions are expressed in terms of the real- and set-valued operators shown below. Each operator is stated in the context of a word w in a graph G. DG defines the following set-valued operators (we let n^* denote the type join of the node n with all of its parent edges):

- $\mathbf{this} := \{w\}$ where w is the word defined by the context.
- $\mathbf{dep}(\mathbf{t}) :=$ set of all dependents d of w such that d^* has type t.
- $\mathbf{left}(\mathbf{t}) :=$ set of all left landed nodes n of w such that n^* has type t.
- $\mathbf{right}(\mathbf{t}) :=$ set of all right landed nodes n of w such that n^* has type t.
- $\mathbf{gov}(\mathbf{t}) := \{g\}$ if w has governor g and g^* has type t, $\emptyset$ otherwise.
- $\mathbf{lsite}(\mathbf{t}) := \{l\}$ if w has landing site l and l^* has type t, $\emptyset$ otherwise.
- $\mathbf{island}(\mathbf{t}) :=$ set of all n such that e is a dependent edge of w with type t, and n is a node whose path from its governor to its landing site includes e.
- $\mathbf{sem}(\mathbf{n},\mathbf{t}) := \{n\}$ if the semantics of node n is of type t, $\emptyset$ otherwise.

We extend the definition of these operators to set-valued arguments by defining

$$\Omega(X_1,\ldots,X_n) := \bigcup_{(x_1,\ldots,x_n)\in X_1\times\cdots\times X_n} \Omega(x_1,\ldots,x_n)$$

for each set-valued operator Ω with set-valued arguments $X_1, \ldots, X_n$. Moreover, we define the following integer-valued operators (where 0 corresponds to false, and 1 corresponds to true):

- $\mathbf{n_1} < \mathbf{n_2}$:= 1 if node n_1 precedes node n_2, 0 otherwise.
- $\mathbf{n_1} > \mathbf{n_2}$:= 1 if node n_1 follows node n_2, 0 otherwise.
- $\mathbf{x} = \mathbf{y}$:= 1 if x and y are identical, 0 otherwise.
- $\mathbf{x} \neq \mathbf{y}$:= 1 if x and y are non-identical, 0 otherwise.
- $\mathbf{isa}(\mathbf{n}, \mathbf{t})$:= 1 if node n has type t, 0 otherwise.
- $\mathbf{dist}(\mathbf{n})$:= the distance between w and node n, measured as the number of intervening words.

These operators are extended to set-valued arguments by defining:

$$\omega(X_1, \ldots, X_n) := \sum_{(x_1, \ldots, x_n) \in X_1 \times \ldots X_n} \omega(x_1, \ldots, x_n)$$

for each integer-valued operator ω with set-valued arguments $X_1, \ldots, X_n$. Finally, we define the following integer-valued operators (where $+$ denotes addition of real numbers, and $*$ multiplication):

- $|\mathbf{x}|$:= the absolute value of x, defined as the cardinality of x if x is a set.
- $\mathbf{and}(\mathbf{x_1}, \ldots, \mathbf{x_n}) := |x_1| * \cdots * |x_n|$
- $\mathbf{or}(\mathbf{x_1}, \ldots, \mathbf{x_n}) := |x_1| + \cdots + |x_n|$
- $\mathbf{not}(\mathbf{x})$:= 1 if $|x| = 0$, and 0 otherwise.

In the following, we will demonstrate how the operators above can be used to encode cost functions for a wide range of syntactic constraints, ie, to express what counts as a violation of the constraint. In a real lexicon, the cost functions must be assigned relative weights by multiplying them with a fixed non-negative cost. However, for simplicity, we will pretend that all cost functions have unit cost.

2.2 Missing edges: roots and obligatory dependents

Since we want a complete rather than partial analysis, a word should impose a cost for a missing landing site or a governor. This can be achieved with:

- not(lsite(any)): missing landing site
- not(gov(any)): missing governor

In DG, all dependents are optional by default, since no cost is associated with their absence. However, complements and adjuncts can be made "obligatory" by imposing a cost on their absence. Here is an example for the English verb *reside*:

- not(dep(subj)): missing subject
- not(dep(locadv)): missing place adverbial

2.3 *Word order violations*

A landing site has a left field and a right field that contains its landed nodes. Some violations in the right verbal field are listed below. In Danish and English, these violations are always ungrammatical and must be assigned a high cost. In German, they are only marked and must be assigned a lower cost.

- right(subj) $>$ right(land): a subject is preceded by any landed node
- right(iobj) $>$ right(comp$-$subj): an indirect object is preceded by any complement which is not a subject
- right(dobj) $>$ right(comp$-$subj$-$iobj): a direct object is preceded by any complement which is neither a subject, nor an indirect object

In verbs with V2-order in Danish and German, the left field must either contain one complement, or one or more adjuncts. Violations can be tested with:

- not(left(land)): there are no landed nodes
- left(comp) $\neq$ left(land): there is a complement and some other dependent

2.4 *Island constraints*

While particular island constraints are not true of all languages, it must be possible to state them in DG. The main island constraints are listed below, stating a violation condition and the word classes it applies to.

- island(subj) in verb: extraction occurs via a subject edge (Subject Condition)
- island(vobj|rel) in noun: extraction occurs via an edge to a relative clause or a verbal complement in a noun (Complex NP Constraint)
- and(sem(this,wh),island(any)) in verb: extraction occurs via any edge in a wh-marked verb (Wh-Island Condition)
- island(adj) in word: extraction occurs via an adjunct edge in any word (Adjunct Island Constraint)
- and(gov(that),dep(subj+ext)) in verb: a subject is extracted from a verb governed by a complementizer (Complementizer Gap Constraint)

2.5 *Agreement*

Agreement can be tested with the following violation conditions:

- isa(this,subj–nom) in noun: subject has no nominative marking
- isa(dep(subj),neut) $\neq$ isa(dep(adjective+spred),neut) in verb: subject and subject predicative have different neuter markings

2.6 *Selectional restrictions and pragmatics*

In order to implement selectional restrictions, we assume that semantic interpretations are organized in a type hierarchy so that we can test whether they match a semantic type specification.

- sem(dep(subj),–human) in verb: subject does not denote a human
- and(isa(this,locadv), sem(this,–place)) in noun and prep: a noun or preposition that acts as place adverbial does not denote a place

More generally, cost functions give a natural way of taking the semantic and pragmatic plausibility of a syntax graph into account: simply add the cost computed by the semantic or pragmatic module to the other costs.

2.7 Word distance

Evidence in psycholinguistics suggests a preference for low attachment in human parsing [4]. In DG, this can be formulated as a preference for closeness between a node and its landing site and governor:

- dist(gov(any)): number of intervening words between word and governor.
- dist(land(any)): number of intervening words between word and landing site.

2.8 Lexical preferences and probabilities

Words often have alternative readings that occur with different frequencies. In DG, frequent forms can be favored over infrequent forms by assigning a fixed cost to each form. The cost might reasonably be selected as $-\log P(w)$ where $P(w)$ is the prior probability of word form w. Then the total lexical preference cost in graph G would reduce to $-\sum_{w\in G}\log P(w) = -\log\left(\prod_{w\in G} P(w)\right)$, ie, to the probability of the word sequence under a minus-log transform, assuming statistical independence between the words.

Similarly, we can attach fixed costs to complement structures, resulting in the probability of a tree as calculated in a dependency-based PCFG, but under a minus-log transform. More generally, probabilities could be used to estimate the "true" cost of any violation condition: simply weigh a cost function by the cost $-\log p$ where p is the probability of a violation; then p might be estimated from a corpus of parsed sentences, or perhaps from a corpus of intermediate parses.

2.9 Interpretations of cost

Since costs can be interpreted as minus-log probabilities, DG can be viewed as a dependency-based generalization of a PCFG grammar [11]. DG can also be viewed as a generalization of Optimality Theory, since constraint violations in OT have an obvious interpretation in terms of costs, as follows:

Let r denote the *rank function* from OT constraints into the non-negative integers, defined by assigning 0 to the lowest ranked constraints, 1 to the next-lowest ranked constraints, etc. Let $V(n,c)$ denote the number of violations of an OT constraint c at a node n in the graph. Then the *cost polynomial* corresponding to the constraint c at node n can be defined as the function $C(n,c) = V(n,c)X^{r(c)}$ where X is a polynomial variable. The sum of all local cost polynomials encodes the total number of OT violations and their rank. Moreover, OT's candidate ordering reduces to the standard order on polynomials (ie, $f > g$ if the leading coefficient of $f - g$ is positive). Thus, OT constraints can be viewed as cost functions with polynomial-valued costs, which can be reduced to real-valued costs by setting X to a large positive number.

Finally, the heuristic parsing algorithm in DG may have an interpretation in terms of distributed competition models such as [17]. With a slight reformulation of such models, we might interpret human parsing as a sequential race between different parsing operations that are computed in parallel. The first operation that finishes its constraint checking wins, and the race is then continued with a new set of operations. That is, costs can also be interpreted as processing times.

3 Optimality parsing

Parsing can be viewed as a search problem in which a space of potential parses of some input is searched for an optimal solution, given some cost function that measures linguistic plausibility. The problem of finding a guaranteed globally optimal solution — that is, the problem of *exact parsing* — is probably intractable in DG, [4] like in most other formalisms. [5] However, from a purely linguistic point of view, the inability to find an efficient exact parsing algorithm is no cause for concern, since we know from psychological experiments with garden-path sentences that human parsing is not exact [4]. Thus, the linguistic interest lies in finding fast heuristic parsing algorithms that accurately model human parsing and its imperfections, not in solving the problem of exact parsing.

Exact parsing is also undesirable from an engineering point of view, since the inability to model human parsing may lead to applications that misinterpret human input or generate sentences that humans fail to parse correctly.

3.1 Parsing with local search

Many NP-hard optimization problems, including the Travelling Salesman's Problem, can be solved quite successfully with local search [14, ch.19]. Local search is based on the simple idea that if we can define a small neighbourhood around every

[4] Chart parsing in DG is impractical since a string of n words has $2^n - 1$ discontinuous substrings, compared to the at most $\frac{1}{6}n(n+1)(n+2)$ continuous substrings that must be stored by a chart parser for a continuous formalism like CFG. Exact DG parsing can be shown to be NP-hard if the grammar is considered to be part of the input, since there is an easy reduction from maximal satisfiability problems [13] to exact DG-parsing in this case.

[5] NP-hard formalisms include Optimality Theory with unrestricted constraints [18], Stochastic Tree-Substitution Grammars and Data-Oriented Parsing [16], LFG [12], and constraint-based formalisms such as HPSG. CFG and TAG are exceptions since they can be parsed in $O(n^3)$ and $O(n^6)$ time, respectively, although there are linguistic phenomena that they cannot account for [1]. Moreover, chart parsers for CFG and TAG create a packed chart that may contain an exponential number of parses (e.g., consider the grammar $X \to X\ X$ with terminal rule $X \to x$). Thus, given a linguistically plausible cost measure on top of CFG and TAG (say, a measure that checks semantic and pragmatic plausibility), it is quite conceivable that exact optimality parsing in CFG and TAG will turn out to be NP-hard.

node in a large solution space, then we can find a local optimum by starting at any solution, picking the best solution in its neighbourhood, and repeating the procedure until we have found a local optimum, hoping that the eventual local optimum will be globally optimal or near-optimal. Given the right kind of problem and the right kind of neighbourhoods, this strategy can be extremely successful.

The parsing algorithm in DG is based on local search. The search space is defined as the set of all partial parses of a sentence, and the solution space consists of all partial parses that contain all input words. Thus, even if DG parsing fails, it will return a maximal partial parse of the input. The starting point is the empty parse, and the total cost of a partial parse is defined as the sum of all violation costs imposed by the words in the parse. This gives the algorithm for *optimality parsing*:

1. Set G := empty graph.
2. Set G' := any optimal graph in the neighbourhood of G.
3. Terminate if G has lower cost than G' and contains all input words.
4. Set G := G' and repeat step 2.

The local neighbourhood around each partial parse is defined in terms of a set of basic parsing operations that compute a set of new graphs from the existing graph. If there are any unread words in the input, the operation that reads the next word and adds it to the graph is always a valid parsing operation. The identity mapping is never a valid parsing operation, ie, a graph is never included in its own neighbourhood.

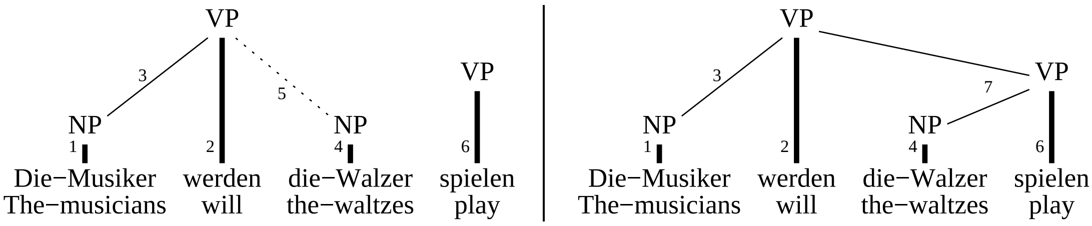
Ideally, the parsing operations must be chosen strong enough to recover from suboptimal analyses that humans recover from, and weak enough to be trapped in suboptimal analyses that humans are trapped in. Preferably, the operations must be computationally efficient as well. There is no reason to suppose that all humans have the same set of parsing operations. On the contrary, humans seem to revise their parsing strategies after exposure to garden-path sentences, so it is reasonable to suppose that parsing operations are learned, and hence that there is some variation between different people and language communities.

When testing an optimality parser, we need to distinguish between grammar failures and parser failures. *Grammar failures* arise if the cost measure defined by the grammar is wrong, either because an undesired analysis is deemed optimal by the grammar, or a desired analysis is deemed non-optimal. *Parser failures* arise when the parser fails to find an optimal analysis. Obviously, one cannot blame the parser for any grammar failures, but parser failures are serious counter-evidence against the parser, unless humans suffer from the same problem.

3.2 Parsing operations

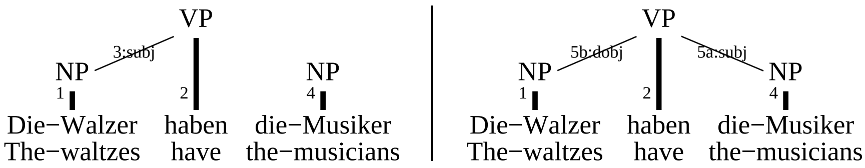
Parsing operations that merely add new nodes or edges to the graph are called *monotonic*, whereas parsing operations that also delete edges from the graph are called *non-monotonic*. Nodes in the graph without a landing site are called *surface roots*, and nodes without a governor are called *deep roots*. DG tentatively proposes six parsing operations:

There are three different kinds of monotonic operations: *lex-operations* add the next unread word to the graph, *land-operations* add a landing site edge to a surface root, and *dep-operations* add a governor edge (and possibly a landing site edge, if none exists already) to a deep root. The land-operation is used to attach a node to a landing site in cases where the governor is not yet available, such as a subject temporarily landing on a complementizer until its governing verb is encountered. In the German example below, steps 1, 2, 4, and 6 are lex-operations, step 3 is a dep-operation, and step 5 is a land-operation.

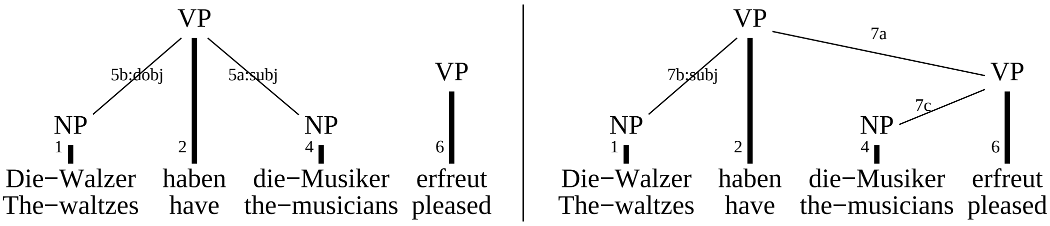

VP
3
5
VP
NP
NP
1
2
4
6
Die−Musiker werden die−Walzer spielen
The−musicians will the−waltzes play
VP
3
7
VP
NP
NP
1
2
4
6
Die−Musiker werden die−Walzer spielen
The−musicians will the−waltzes play

Furthermore, there are three different kinds of non-monotonic repair operations, involving one, two, and three nodes, respectively. *Repair1-operations* assign a governor and a new landing site to a deep root. In the example above, the right-hand side shows the result of a repair1-operation in step 7 where the direct object's temporary landing site from step 5 is replaced with the subordinate verb as landing site and governor.

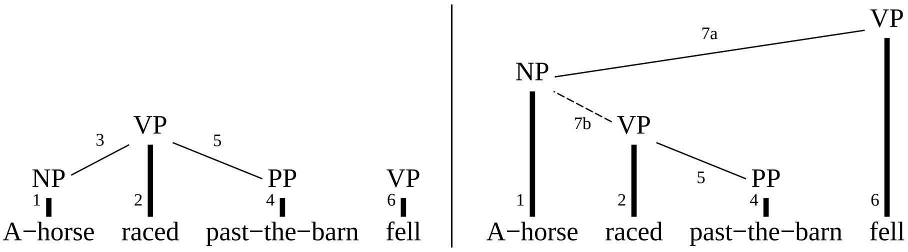
Repair2-operations assign a governor and landing site to a deep root by pushing aside another dependent of the governor which is then given a new governor and landing site. Thus, repair2-operations involve two nodes. An example is given below for the topicalized German sentence corresponding to "The musicians have the waltzes." In the repair2-operation in step 5, "die Musiker" pushes aside "die Walzer" as subject of "haben," reanalyzing "die Walzer" as the direct object of "haben" instead.


VP
3:subj
NP
NP
1
2
4
Die−Walzer haben die−Musiker
The−waltzes have the−musicians
VP
5b:dobj
5a:subj
NP
NP
1
2
4
Die−Walzer haben die−Musiker
The−waltzes have the−musicians

Repair3-operations assign a governor and landing site to a deep root by pushing aside two existing dependents of the governor, giving one of them a new dependent role of the old governor, and the other a new dependent role of a possibly new governor. The example below is the continuation of the parse above: in step 7, a repair3-operation pushes aside "die-Walzer" as direct object of "haben" in order to make room for the VP complement "erfreut" in step 7a; "die Musiker" is then pushed aside and the subject role taken over by "die Walzer" in step 7b; finally, "die Musiker" is analyzed as the object of "erfreuen" in step 7c.


VP
5b:dobj
5a:subj
NP
NP
VP
1
2
4
6
Die−Walzer haben die−Musiker erfreut
The−waltzes have the−musicians pleased
VP
7a
7b:subj
VP
NP
7c
NP
1
2
4
6
Die−Walzer haben die−Musiker erfreut
The−waltzes have the−musicians pleased

Garden-path sentences exemplify the kinds of repair operations that are not freely available to the human parser. For example, in the classical example below, "fell" would have to steal the subject from "raced" in step 7a, and "raced" would have to be reanalyzed as adjunct of "a horse" rather than governor in step 7b.


VP
3
5
NP
PP
VP
1
2
4
6
A−horse raced past−the−barn fell
VP
7a
NP
7b
VP
5
PP
1
2
4
6
A−horse raced past−the−barn fell

Parsing operations that affect at most k nodes in the graph, either by adding a node to the graph or by changing its governor and landing site, are called k-change operations. Clearly, any parsing operation is a k-change operation for large enough k. In particular, the repair operation needed in the garden-path example is a 2-change operation.

A k-change operation has time complexity $O(n^{2k})$, so if we want parsing to be almost linear, k-change is too expensive. However, it is conceivable that humans employ 2-change and 3-change as higher-level repair operations, which are only used as a last, unreliable resort if all normal parsing operations fail. This may also account for how humans learn a set of parsing operations: Initially, they have to resort to 2-change and 3-change operations to find an optimal parsing operation. But after a while, they notice a pattern in the optimal operations which allow them to formulate a more efficient class of parsing operations, such as land, dep, repair1, repair2, and repair3.

3.3 Formal definition of parsing operations

In the following, let $G+E$ denote the graph G plus the edges or words in E, let $G-E$ denote G minus the edges in E, and let $G*E$ denote G plus E minus all edges in G that are incompatible with edges in E (recall that each node has at most one governor or landing site). Moreover, let $n^{\lambda\to l}_{\gamma\to g}$ denote a node n connected with surface edge λ to landing site l, and deep edge γ to governor g. Before defining the different parsing operations, we first need to define the following functions:

- **lands(G)** := set of all $r^{\lambda \to l}$ such that r is a deep root and l is an adjacent landing site that accepts r as a local or external landed node (provided we delete any existing landing site for r in G), with surface edge λ set to local or ext accordingly. *Intuition: return all potential root-landing site configurations for deep roots in the graph.*
- **lgovs(G)** := set of all $r_g^{\lambda \to l}$ such that $r^{\lambda \to l}$ is in lands(G) with $g = l$ if $\lambda = \mathsf{local}$, and g dominated by l in the deep tree if $\lambda = \mathsf{ext}$. *Intuition: return all potential root-landing site-governor configurations for all deep roots in the graph.*
- **deps(G, d, g)** := set of all $d_{\gamma \to g}$ such that g accepts d as a complement with complement edge γ, or is accepted by d as an adjunct governor with adjunct edge γ. *Intuition: return all potential dependency edges for a given dependent and governor.*
- **blocks(G, d, g, B)** := set of all complement edges $d_{\gamma \to g}$ in deps$(G - B, d, g)$ that are not in deps$(G - B', d, g)$ for any proper subset B' of B. *Intuition: return all complement roles for a given dependent and governor that are blocked by a given set of dependency edges.*

Now we can define the local parsing operations that specify the neighbourhood around a graph G as follows:

- **lex(G)** := $\{G'\}$ such that $G' = G + \{w\}$ where w is the next unread word. *Intuition: look up next unread word.*
- **land(G)** := set of all G' such that $G' = G + r^{\lambda \to l}$ where r is a surface root of G, $r^{\lambda \to l}$ is in lands(G), and $\lambda = \mathsf{ext}$. *Intuition: find a valid landing site for a surface root.*
- **dep(G)** := set of all G' such that $G' = G + r_{\gamma \to g}^{\lambda \to l}$ where r is a deep root of G, $r_g^{\lambda \to l}$ is in lgovs(G), and $r_{\gamma \to g}$ is in deps(G, r, g). *Intuition: find a valid landing site and governor for a deep root.*
- **repair1(G)** := set of all valid graphs G' such that $G' = G * r_{\gamma \to g}^{\lambda \to l}$ where r is a deep root in G with an existing landing site edge different from $\lambda \to l$, $r_g^{\lambda \to l}$ is in lgovs(G), and $r_{\gamma \to g}$ is in deps(G, r, g). *Intuition: find a valid landing site and governor for a deep root that already has another landing site edge.*
- **repair2(G)** := set of all valid graphs G''' such that

$$G'' = G - d_{\gamma' \to g} * r_{\gamma \to g}^{\lambda \to l}, \quad G''' = G'' * d_{\gamma'' \to g''}^{\lambda'' \to l''}$$

  where $r_g^{\lambda \to l}$ is in lgovs(G), $d_{\gamma' \to g}$ is in G, $r_{\gamma \to g}$ is in blocks$(G, r, g, \{d_{\gamma' \to g}\})$, $d_{g''}^{\lambda'' \to l''}$ is in lgovs(G''), and $d_{\gamma'' \to g''}$ is in deps(G'', d, g''). *Intuition: find a governor and landing site for a deep root by pushing aside another dependent of the governor, finding a new governor and landing site for it.*
- **repair3(G)** := set of all valid graphs G''' such that

$$G' = G - d_{\gamma' \to g} - c_{\rho \to g} * r_{\gamma \to g}^{\lambda \to l}, \quad G'' = G' + c_{\rho' \to g}, \quad G''' = G'' + d_{\gamma'' \to g''}^{\lambda'' \to l''}$$

  where $r_g^{\lambda \to l}$ is in lgovs(G), $d_{\gamma' \to g}$ and $c_{\rho \to g}$ are in G, $r_{\gamma \to g}$ is in blocks$(G, r, g,$

$\{d_{\gamma'\to g}, c_{\rho\to g}\})$, $c_{\rho'\to g}$ is in $\text{deps}(G', c, g)$, $d_{g''}^{\lambda''\to l''}$ is in $\text{lgovs}(G'')$, and $d_{\gamma''\to g''}$ is in $\text{deps}(G'', d, g'')$. *Intuition: find governor and landing site for a deep root by pushing aside two other dependents of the governor, giving the first a new dependency role in the same governor, and the second a new governor and landing site.*

Note that parsing requires at most $3n$ operations since each operation increases the number of words plus edges in the graph, which is bounded by $3n$.

3.4 Parsing complexity

To estimate the worst-case complexity of the parser, let r denote the maximal number of roots in the graph, h the maximal height, l the maximal number of possible root-landing site pairs, s the maximal number of governors searched below a landing site, and d the maximal number of potential dependency edges between a dependent and its governor specified in the lexicon.

The table below shows worst-case estimates for these variables and the local parsing operations in four different scenarios: (a) no assumptions; (b) we assume syntax trees have a bounded number of roots and are *approximately balanced*, ie, their height is at most $O(\log n)$; (c) we assume (b) holds and that island constraints ensure that there is at most one channel for extraction from unbounded depth in each node; (d) we assume (b) holds and that governors and landing sites always coincide, ie, that there are no discontinuities in the graph. The four scenarios are used to analyze the sources of computational inefficiency in the parser.

	max. ops.	**(a) none**	**(b) balanced**	**(c) islands**	**(d) land=gov**
	r	$O(n)$	$O(1)$	$O(1)$	$O(1)$
	h	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
	l	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
	s	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$
	d	$O(1)$	$O(1)$	$O(1)$	$O(1)$
lex	1	$O(1)$	$O(1)$	$O(1)$	$O(1)$
land	l	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
dep	lsd	$O(n^2)$	$O(n\log n)$	$O(\log^2 n)$	$O(\log n)$
repair1	lsd	$O(n^2)$	$O(n\log n)$	$O(\log^2 n)$	$O(\log n)$
repair2	lhs^2d^3	$O(n^4)$	$O(n^2\log^2 n)$	$O(\log^4 n)$	$O(\log^2 n)$
repair3	lhs^2d^4	$O(n^4)$	$O(n^2\log^2 n)$	$O(\log^4 n)$	$O(\log^2 n)$
parse	$3nlhs^2d^4$	$O(n^5)$	$O(n^3\log^2 n)$	$O(n\log^4 n)$	$O(n\log^2 n)$

The table suggests that approximately balanced trees (b) and restrictive island constraints (c) greatly increase the effectiveness of the parser. Highly nested sentences probably cause problems for humans, especially in spoken language, so assumption (b) is not unreasonable. Since island constraints such as the Adjunct Island Constraint, Complex NP Constraint, and the Complementizer Gap Constraint have the consequence that only verbs and adjectives do not behave as "dead ends" with respect to extraction from an unbounded depth, and verbs and adjectives rarely occur together as complements, assumption (c) is not unreasonable either.[6] Assumption (d) corresponds to having no discontinuities, so it is obviously too strong, but may provide a reasonable lower bound on the efficiency of the DG parser.

3.5 Evaluation and examples

The DG parser has been implemented in Prolog with a small grammar encoding valency patterns, case, and selectional restrictions. The implementation is a prototype, and no attempt has been made to use efficient algorithms, so it is quite slow, with time complexity $O(n^5)$. The parser and grammar implementation, as well as the test results for a wide range of discontinuous examples (topicalizations, extractions, scramblings, and discontinuous APs and coordinations, and sentences with garden paths and local ambiguities), can be downloaded from http://www.id.cbs.dk/~mtk/dg.

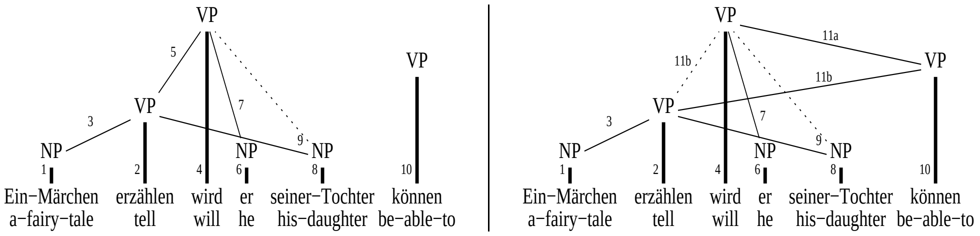
The purpose of the implementation was to examine the precision of the algorithm. When the parser returned a suboptimal parse, it was either a garden path sentence where humans also failed, or the parse was only suboptimal because a landing site was chosen too high in the structure, and optimal in all other respects. Since landing sites play no role in the semantic interpretation, this slight suboptimality is unimportant. Apart from landing site suboptimality, the test showed that the parser returned an optimal analysis in all other cases than strong garden paths. Thus, even though the parser is heuristic, it does not seem to be more significantly more error-prone than an exact parser in practice.

“Ein-Märchen$_1$ erzählen$_2$ wird$_3$ er$_4$ seiner-Tochter$_5$ können$_6$” *A-fairy-tale$_1$ tell$_2$ will$_3$ he$_4$ his-daughter$_5$ be-able-to$_6$*	
1. $\mathbf{lex(1)}_{7.96}$	
2. $\mathbf{lex(2)}_{15.92}$	
3. $\mathbf{dep(1^{land\text{-}2}_{dobj\text{-}2})}_{7.96}$	$\text{lex}(3)_{24.18}$ $\text{dep}(1^{\text{land-2}}_{\text{iobj-2}})_{10.96}$
4. $\mathbf{lex(3)}_{16.22}$	
5. $\mathbf{dep(2^{land\text{-}3}_{vobj\text{-}3})}_{8.26}$	$\text{lex}(4)_{24.18}$ $\text{land}(2^{\text{land-3}})_{12.22}$
6. $\mathbf{lex(4)}_{16.22}$	
7. $\mathbf{dep(4^{land\text{-}3}_{subj\text{-}3})}_{7.96}$	$\text{lex}(5)_{24.18}$ $\text{dep}(4^{\text{land-3}}_{\text{iobj-2}})_{10.26}$ $\text{land}(4^{\text{land-3}})_{12.22}$ $\text{repair2}(4^{\text{land-3}}_{\text{dobj-2}},1^{\text{land-2}}_{\text{iobj-2}})_{14.26}$ $\text{repair2}(4^{\text{land-3}}_{\text{dobj-2}},1^{\text{land-3}}_{\text{subj-3}})_{10.96}$ $\text{repair2}(4^{\text{land-3}}_{\text{dobj-2}},1^{\text{land-3}}_{\text{iobj-2}})_{14.26}$
8. $\mathbf{lex(5)}_{15.92}$	
9. $\mathbf{dep(5^{land\text{-}3}_{iobj\text{-}2})}_{7.96}$	$\text{lex}(6)_{23.88}$ $\text{land}(5^{\text{land-3}})_{11.92}$ $\text{repair2}(5^{\text{land-3}}_{\text{subj-3}},4^{\text{land-3}}_{\text{iobj-2}})_{14.96}$ $\text{repair2}(5^{\text{land-3}}_{\text{dobj-2}},1^{\text{land-2}}_{\text{iobj-2}})_{13.96}$ $\text{repair2}(5^{\text{land-3}}_{\text{dobj-2}},1^{\text{land-3}}_{\text{iobj-2}})_{13.96}$
10. $\mathbf{lex(6)}_{15.92}$	
11. $\mathbf{repair2(6^{land\text{-}3}_{vobj\text{-}3},2^{land\text{-}3}_{vobj\text{-}6})}_{7.96}$	$\text{land}(3^{\text{land-6}})_{11.92}$ $\text{land}(6^{\text{land-3}})_{11.92}$

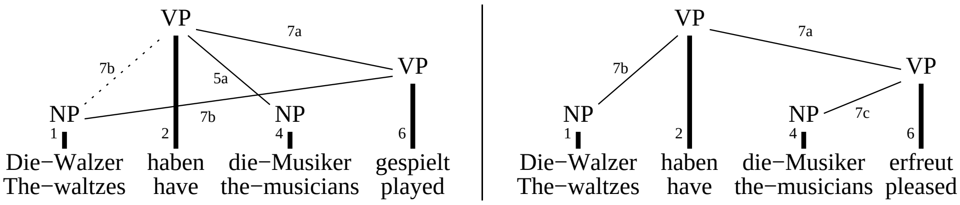
[6] The observation that there is usually only one “spine” of extraction was made independently by Christian Wartena, in a talk given in the Mathematics of Language workshop at ESSLLI 2001.

The parse above illustrates how the DG parser works on the discontinuous German topicalization “Ein Märchen erzählen wird er seiner-Tochter können.” The left column shows the parsing operation that was selected as the minimal-cost operation in each step, the right column all the alternative operations. The associated cost is shown in subscript after each operation. At each step, the selected operation is superior to the alternatives because of the number of roots, case markings, or selectional restrictions. The notation $\pi(E)$ indicates a parsing operation π with new edges E turning the graph G into $G * E$.

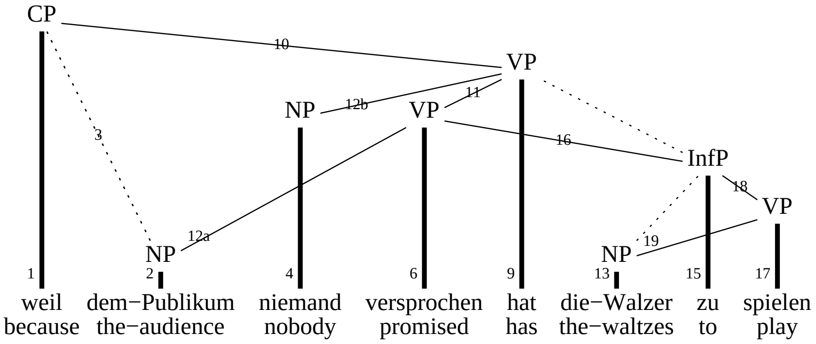
The resulting graph, before and after the repair in step 11, is shown below, with an indication of the step in which each edge was created (if governor and landing site coincide, only the deep edge is shown):


VP
5
VP
7
9
VP
3
NP
NP
NP
1
2
4
6
8
10
Ein−Märchen erzählen wird er seiner−Tochter können
a−fairy−tale tell will he his−daughter be−able−to
VP
11a
11b
VP
11b
VP
7
9
3
NP
NP
NP
1
2
4
6
8
10
Ein−Märchen erzählen wird er seiner−Tochter können
a−fairy−tale tell will he his−daughter be−able−to

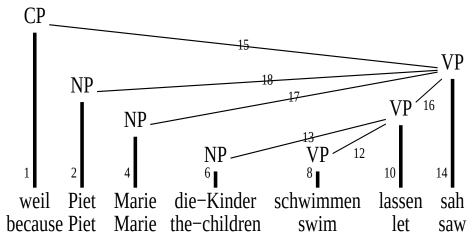
The example below demonstrates local ambiguity, where the NPs can be both subjects and objects, depending on the final verb, and where the verb *have* is ambiguous between having a nominal and a verbal object.

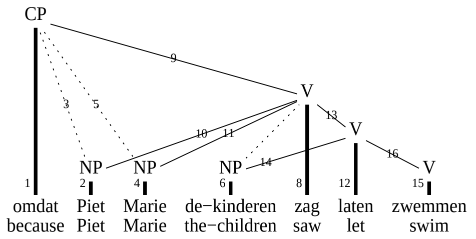

VP
7a
7b
VP
5a
NP
7b
NP
1
2
4
6
Die−Walzer haben die−Musiker gespielt
The−waltzes have the−musicians played
VP
7a
7b
VP
NP
NP
7c
1
2
4
6
Die−Walzer haben die−Musiker erfreut
The−waltzes have the−musicians pleased

The example below illustrates scrambling and extraposition in German:

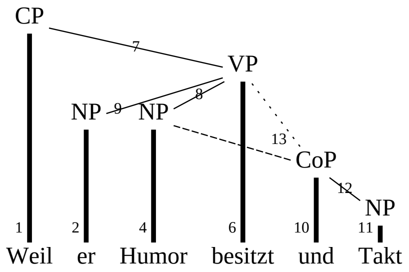

CP
10
VP
NP
12b
VP
11
3
16
InfP
18
VP
12a
NP
19
NP
1
2
4
6
9
13
15
17
weil dem−Publikum niemand versprochen hat die−Walzer zu spielen
because the−audience nobody promised has the−waltzes to play

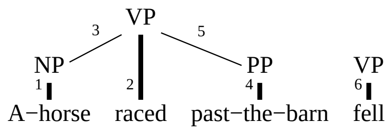
The examples below illustrate center-embedding in German and Dutch:





The example below left illustrates discontinuous coordinations in German. The right example illustrates how DG fails to parse a garden path sentence because its repair operations are too weak, just like in humans.





4 Conclusion

Local cost functions in Discontinuous Grammar provide a flexible and efficient means to measure the syntactic, semantic, pragmatic, and probabilistic plausibility of a parse. Moreover, the optimality parser is very fast, with theoretical time complexity $O(n \log^4 n)$ under linguistically reasonable assumptions, and tests with a simple implementation of an optimality parser and a grammar shows that the proposed parsing operations are strong enough to provide the correct analyses for a wide range of discontinuous sentences, although it fails on garden path sentences where humans fail. Thus, optimality parsing seems to provide a fast and accurate model of human parsing and its imperfections.

Some issues have not been addressed in this article, in particular, the question of how to deal with ambiguity, and the question of how to improve the average-case complexity of the DG parser by using sophisticated search techniques. New parsing operations may have to be added to the set of admissible parsing operations in order to account for some phenomena, such as adverbial reanalysis. Finally, a large-scale grammar and evaluation of DG would be highly desirable as well.

References

[1] Abeillé, A. and O. Rambow, editors, “Tree Adjoining Grammars. Formalisms, Linguistic Analysis, and Processing,” CSLI Publications, 2000.

[2] Bouma, G., R. Malouf and I. Sag, *Satisfying constraints on extraction and adjunction*, Natural Language and Linguistic Theory (2001).

[3] Dalrymple, M., R. Kaplan, J. M. III and A. Zaenen, editors, “Formal issues in Lexical-Functional Grammar,” CSLI Lecture Notes, no. 47, 1994.

[4] Gernsbacher, M. A., editor, “Handbook of psycholinguistics,” Academic Press, 1994.

[5] Haegeman, L., “Introduction to Government and Binding Theory,” Blackwell, 1994/1991, 2nd edition.

[6] Helbig, G. and W. Schenkel, “Wörterbuch zur Valenz und Distribution deutscher Verben,” Max Niemeyer Verlag, 1971/1969.

[7] Hudson, R., *An encyclopedia of English grammar and Word Grammar*, http://www.phon.ucl.ac.uk/home/dick/enc-gen.htm (2001).

[8] Kager, R., “Optimality Theory,” Cambridge University Press, 1999.

[9] Kahane, S., editor, “Dependency grammars,” Traitement Automatique des Langues **41 (1)**, 2000.

[10] Kromann, M. T., *Towards Discontinuous Grammar*, in: *Proceedings of the ESSLLI 1999 Student Session*, 1999, pp. 145–154.

[11] Manning, C. D. and H. Schütze, “Foundations of Statistical Natural Language Processing,” MIT Press, 1999.

[12] Maxwell, J. T. and R. M. Kaplan, *The interface between phrasal and functional constraints*, Computational Intelligence **19** (1993), pp. 571–590.

[13] Papadimitriou, C. H., “Computational Complexity,” Addison-Wesley, 1994.

[14] Papadimitriou, C. H. and K. Steiglitz, “Combinatorial Optimization. Algorithms and Complexity,” Dover Press, 1982.

[15] Pollard, C. and I. A. Sag, “Head-driven Phrase Structure Grammar,” University of Chicago Press, 1994.

[16] Sima’an, K., *Computational complexity of probabilistic disambiguation by means of tree grammars*, in: *Proceedings of COLING 96*, COLING **2**, 1996, pp. 1175–1180.

[17] Vosse, T. and G. Kempen, *Syntactic structure assembly in human parsing: a computational model based on competitive inhibition and a lexicalist grammar*, Cognition **75** (2000), pp. 105–143.

[18] Wareham, H. T., “Systematic parameterized complexity analysis in computational phonology,” Ph.D. thesis, University of Victoria (1998), Rutgers Optimality Archive 318-0599.