COMPUTING THE MOST PROBABLE PARSE FOR A DISCONTINUOUS PHRASE STRUCTURE GRAMMAR

Oliver Plaehn*
Department of Computational Linguistics
University of the Saarland
66041 Saarbrücken, Germany
plaehn@coli.uni-sb.de

Abstract

This paper presents a probabilistic extension of Discontinuous Phrase Structure Grammar (DPSG), a formalism designed to describe discontinuous constituency phenomena adequately and perspicuously by means of trees with crossing branches. We outline an implementation of an agenda-based chart parsing algorithm that is capable of computing the Most Probable Parse for a given input sentence for probabilistic versions of both DPSG and Context-Free Grammar. Experiments were conducted with both types of grammars extracted from the NEGRA corpus. In spite of the much greater complexity of DPSG parsing in terms of the number of (partial) analyses that can be constructed for an input sentence, accuracy results from both experiments are comparable. We also briefly hint at future lines of research aimed at more efficient ways of probabilistic parsing with discontinuous constituents.

1 Introduction

Natural languages exhibit a good deal of discontinuous constituency phenomena, especially with regard to languages with a relatively free word order like German or Dutch, that cannot be described adequately by context-free phrase structure trees alone. Arguments from linguistics motivate the representation of structures containing discontinuous constituents by means of trees with crossing branches (McCawley, 1982; Blevins, 1990; Bunt, 1996). Bunt (1991, 1996) proposed a formalism called *Discontinuous Phrase Structure Grammar* (DPSG) for use in generation of and parsing with discontinuous trees and outlined an active chart parsing algorithm for DPSGs.

DPSG is quite a natural and straightforward extension of common Context-Free Grammar (CFG). Hence, it is reasonable to assume that *probabilistic* approaches to parsing with CFGs might also be extended with similar ease to deal with crossing branches. In order to test this hypothesis, we adapt the algorithm for computing the Most Probable Parse (MPP) for a sentence given a probabilistic CFG (PCFG) to perform the same task for a DPSG enhanced with rule probabilities (PDPSG) and report on experiments conducted with this algorithm based on grammars extracted from the NEGRA corpus (Skut, Krenn, Brants, & Uszkoreit, 1997). It turns out that accuracy results from PDPSG experiments are comparable to those obtained in PCFG experiments on the same corpus.

In Sections 2 and 3, we define precisely what we mean by a discontinuous tree and a DPSG and present our chart parser for DPSGs. The theoretical foundations of our MPP algorithm for DPSG are discussed in Section 4, in which we also explain how the MPP is computed by a slightly modified version of our parser. Section 5 reports on experiments conducted with grammars extracted from a

*The author is grateful to Thorsten Brants and Alexander Koller for valuable comments and discussion, and also to Nicole Siekmann for proof-reading an earlier version of this paper.

corpus of discontinuous trees and discusses the results we obtained. The paper[1] concludes with a brief summary and an outline of possible future lines of research.

2 Discontinuous Phrase Structure Grammar

The formalism of Discontinuous Phrase Structure Grammar (DPSG) is originally due to Bunt (1991, 1996). Below, we slightly deviate from Bunt's account in order to have suitable formal definitions on which we can base our algorithm for computing the MPP. However, the motivation and ideas underlying DPSG are preserved.

We start with a recursive definition of discontinuous trees (or discotrees for short). Discotrees consist of substructures which themselves are not necessarily valid discotrees. Consider, for instance, the discotree (a) in Figure 1 that contains the two substructures in (b) and (c).

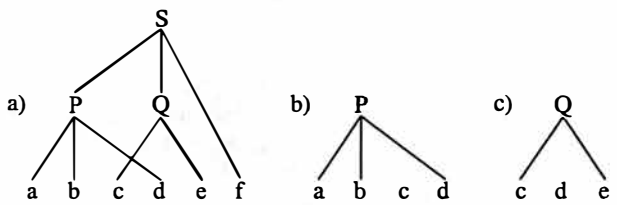


Figure 1: The discotree in (a) contains substructures (b) and (c), which are not valid discotrees

We thus need a definition of these substructures first, which we provide below.

Definition 1 (Subdiscotree)
Let N be a non-empty set of nodes. A subdiscotree *is recursively defined as follows.*

1. *If $x \in N$, then the pair $\langle x, [\,] \rangle$ is a subdiscotree; such a subdiscotree is called* atomic.

2. *If $x \in N$ and $X_1, \ldots, X_n$ ($n \geq 1$) are subdiscotrees that do not share any subdiscotrees[2], then the pair $\langle x, [X_1, X_2^{c_2}, \ldots, X_{n-1}^{c_{n-1}}, X_n] \rangle$ is a subdiscotree, where $c_i \in \{0,1\}$ for $1<i<n$.*

3. *No other structures than those defined by (1) and (2) are subdiscotrees.* ∎

In a subdiscotree $\langle x, [X_1, X_2^{c_2}, \ldots, X_{n-1}^{c_{n-1}}, X_n] \rangle$, the node x *immediately dominates* the top nodes of X_1, X_n, and of every $X_i^{c_i}$ such that $c_i = 0$. In other words, subdiscotrees consist of a top node x and a sequence of daughter constituents whose top nodes are immediately dominated by x, possibly interrupted by constituents whose top nodes are not dominated by x. These latter nodes are called *internal context* of x or *context daughters* of x. As shorthand notation for $\langle x, [X_1, \ldots, X_n] \rangle$, we use $x(X_1, \ldots, X_n)$ and enclose context daughters in brackets instead of using superscripts for this purpose. For example, the structure in Figure 1a is a graphical representation of the subdiscotree S(P(a, b, [c], d), Q(c, [d], e), f).

Let I denote the immediate dominance relation. *Dominance* (D) is defined as the reflexive and transitive closure of I, as usual. A *discotree* is now simply a subdiscotree T in which every node except the root is immediately dominated by some other node in T.

[1] Due to limitations of space, parts of our exposition are rather brief; a more detailed account can be found in (Plaehn, 1999).

[2] Two subdiscotrees X and Y share a subdiscotree Z, if both X and Y contain a node dominating the top node of Z. The inclusion of this condition ensures that subdiscotrees do not contain multidominated nodes.

Next, we provide a suitable definition of linear precedence in discotrees. To this end, we first need two auxiliary definitions. The *leftmost daughter* of a subdiscotree $T = x(X_1, \ldots, X_n)$, $Lm(T)$, is the top node of X_1; if $T = \langle x, [\,] \rangle$ is atomic, then $Lm(T) = x$. The *leaf sequence* of a discotree T, *LeafSeq*(T), is the sequence of terminal nodes in T's formal representation in left to right order; if a leaf appears more than once, only its leftmost appearance is included in the sequence. The leaf sequence of S(P(a, b, [c], d), Q(c, [d], e), f) is thus $\langle$a, b, c, d, e, f$\rangle$.

Definition 2 (Linear precedence (<))
Linear precedence *(<) in a discotree A with LeafSeq*$(A) = \langle x_1, \ldots, x_n \rangle$ *is defined as follows.*

1. *For two leaves* x_i *and* x_j *in LeafSeq*(A), $x_i < x_j$ *if and only if* $i < j$.

2. *For two arbitrary nodes* x *and* y *in* A, $x < y$ *if and only if*

(a) $Lm(x) < Lm(y)$ *and*
(b) $\exists$ node z: $Lm(x) < z \leq Lm(y) \wedge \neg(xDz)$. ∎

In words, the leaves of a discotree are totally ordered according to <. For two arbitrary nodes x and y, x precedes y if and only if the leftmost daughter of x precedes the leftmost daughter of y and there exists a node z in between $Lm(x)$ and $Lm(y)$ or identical to the latter, such that x does not dominate z. In the discotree in Figure 1a, we thus have a<b, a<c, a<d, a<e, a<f, b<c, b<d, b<e, b<f, c<d, c<e, c<f, d<e, d<f, e<f, P<c, P<d, P<e, P<f, a<Q, b<Q, Q<d, Q<e, Q<f, P<Q. Note that both P<d and P*I*d holds; that is, there is no "Exclusivity Condition". This is necessary because we want a node n to precede its first context daughter (and < to be a strict partial order) since the latter might be the leftmost daughter of a different constituent that is preceded by n. We say that two nodes x and y are *adjacent* (denoted by $x + y$) if x precedes y and there is no node z such that $x < z < y$. In our example discotree, the following adjacency relations hold: a+b, b+c, c+d, d+e, e+f, P+c, b+Q, P+Q, Q+d.

A CFG rule is used to rewrite its left-hand side category as a sequence of pairwise adjacent constituents. If we directly apply this concept to the generation of discotrees, we encounter a problem. To see this, consider again the discotree in Figure 1a which we would like to be generated by the rules in (1).

(1) S → P Q f
P → a b [c] d
Q → c [d] e

(2) S → P Q [d] [e] f
P → a b [c] d
Q → c [d] e

But the first rule is not applicable, since Q and f are not adjacent. We would thus be forced to use the rules in (2) which are quite awkward and counterintuitive. To eliminate this problem, we relax the condition that the symbols on the right-hand side of a rule have to be pairwise adjacent and instead require them to form an adjacency sequence, as defined below.

Definition 3 (Adjacency sequence)
A sequence of nodes $\langle x_1, x_2, \ldots, x_n \rangle$ *in a subdiscotree A is an* adjacency sequence *if and only if*

1. $\forall 1 \leq i < n$: $\big(x_i{+}x_{i+1} \vee (\exists$ nodes $y_1, \ldots, y_m$ in A: $x_i{+}y_1 \wedge y_1{+}y_2 \wedge \ldots \wedge y_{m-1}{+}y_m \wedge y_m{+}x_{i+1} \wedge \forall 1 \leq j \leq m$: $\exists 1 \leq k \leq i$: $x_k D y_j)\big)$ *and*

2. $\forall 1 \leq i < j \leq n$: $\neg(\exists$ node z in A: $x_i Dz \wedge x_j Dz)$. ∎

In words, we require in clause (1) that every pair $\langle x_i, x_{i+1} \rangle$ in the sequence is either an adjacency pair or is connected by a sequence of adjacency pairs of which all members are dominated by some element in the subsequence $\langle x_1, \ldots, x_i \rangle$ and in clause (2) that the elements of the sequence do not share any constituents. Thus, $\langle$P, Q, f$\rangle$ constitutes an adjacency sequence since P and Q are adjacent, Q and f are connected by the sequence of adjacency pairs Q+d, d+e, e+f, and d and e are dominated by P and Q, respectively. Furthermore, P, Q and f do not share any constituents. We can therefore use the rules in (1) to generate the discotree in Figure 1a.

A Discontinuous Phrase Structure Grammar (DPSG) then essentially consists of a set of rules which can be used in generating a discotree such that the nodes corresponding to the symbols on the rules' right-hand sides form adjacency sequences.

Definition 4 (Discontinuous Phrase Structure Grammar)
A Discontinuous Phrase Structure Grammar (DPSG) *is a quadruple* $\langle V_N, V_T, S, R \rangle$, *where*

- V_N *is a finite set of* nonterminal symbols;
- V_T *is a finite set of* terminal symbols; *let* V *denote* $V_N \cup V_T$;
- $S \in V_N$ *is the distinguished* start symbol *of the grammar;*
- R *is a finite set of* rules $X \rightarrow Y^{1,c_1} Y^{2,c_2} \ldots Y^{n,c_n}$, *where* $X \in V_N$, $Y^1, \ldots, Y^n \in V$, *and* $c_i \in \{0,1\}$ $(1 \leq i \leq n)$ *indicates whether* Y^i *is a context daughter in the rule. Since neither the first nor the last daughter may be marked as internal context,* $c_1 = c_n = 0$. ∎

This definition makes apparent the close similarity between CFG and DPSG. In fact, a DPSG not containing rules with context daughters degenerates to a CFG.

3 The Parsing Algorithm

Bunt (1991) outlined an active chart parsing algorithm for DPSG that constructs discotrees bottom-up (see also (van der Sloot, 1990)). Here, we present an agenda-based chart parser that provides us with greater flexibility with respect to the order in which edges are to be processed.

The algorithm makes use of three data structures. *Edges* correspond to (partial) parse trees (sub-discotrees, that is) and an *agenda* stores edges considered for combination with other edges already residing on the *chart*. An edge consists of the starting and ending position of its corresponding subparse, a pointer to the DPSG rule which has been used to construct the edge, and the number of its right-hand side symbols for which constituents have already been found (fields `start`, `end`, `rule` and `dot_pos`, resp.). This information alone, though, is not sufficient to uniquely describe a possibly discontinuous constituent. We also need to know which terminal symbols are dominated by direct daughters of the edge and which by context daughters. To this end, each edge is additionally associated with two bit-strings (fields `covers` and `ctxt_covers`)[3].

For instance, during the construction of the discotree in Figure 1a given the DPSG in (1), our parser creates the *inactive* edge [0,4,P → a b [c] d •, 110100, 001000] for the subparse in Figure 1b. Later on, it also builds the edge [0,4,S → P • Q f, 110100, 000000], which is called *active* since

[3]The use of bit-strings for representing locations of discontinuous constituents is motivated by (Johnson, 1985).

it still needs to be combined with constituents for Q and f. Notice that the input symbol c is not yet dominated by a direct or a context daughter of the edge. Therefore, the third bit in both `covers` (110100) and `ctxt_covers` (000000) is unset (equals 0).

The core of the parsing algorithm works as follows.

```
for i = 0 to n − 1 do
    edge ← new edge [i, i + 1, t_i → •, 0^i 1 0^{n−i−1}, 0^n];
    agenda.add( edge );
while not agenda.is_empty() do
    edge ← agenda.get_next();
    chart.add(edge);
if chart.success(goal) then output parse(s);
```

During initialization, edges corresponding to the terminal symbols $t_0, \ldots, t_{n-1}$ of the input sentence are added to the agenda. Edges are then popped off the agenda and added to the chart, one after the other. In the process of adding an active edge to the chart, it is combined with all matching inactive edges already on the chart, thus giving rise to new edges, which are added to the agenda. Likewise, an inactive edge popped from the agenda is combined with suitable active ones on the chart and, in addition, gives rise to new edges based on matching DPSG rules. This is repeated until the agenda is empty. If the chart then contains an inactive edge that is headed by the goal category and spans the entire input (that is, if $edge.\texttt{covers} = 1^n$), it corresponds to one or more complete parse trees for the input sentence.

Let us make the pecularities due to dealing with discotrees instead of ordinary context-free trees more precise. Firstly, what conditions must hold so that an active edge ae and an inactive edge ie can be combined? The left-hand side category of ie ($ie.\texttt{lhs}$) must match the symbol to the right of the dot on ae's right-hand side ($ae.\texttt{next\_cat}$). Furthermore, in order to ensure that the constituents corresponding to the right-hand side symbols of ae form an adjacency sequence, the next edge to be combined with ae has to start at the position of the first terminal symbol within ae's span that is neither dominated by a direct nor by a context daughter of ae. We therefore compute the bit-wise 'or' ($\vee$) of $ae.\texttt{covers}$ and $ae.\texttt{ctxt\_covers}$ restricted to the interval $[ae.\texttt{start}, ae.\texttt{end}]$, set $ae.\texttt{next\_pos}$ to the position of the first unset bit in this bit-string, and require that $ie.\texttt{start} = ae.\texttt{next\_pos}$ holds. Additionally, we need to check that the two edges do not share any constituents. This is the case, if the bit-wise 'and' of $ae.\texttt{covers}$ and $ie.\texttt{covers}$ and of $ae.\texttt{ctxt\_covers}$ and $ie.\texttt{covers}$ is zero. To summarise,

$$(3)\quad \begin{array}{lll} ae.\texttt{next\_pos} = ie.\texttt{start}, & & ae.\texttt{next\_cat} = ie.\texttt{lhs}, \\ ae.\texttt{covers} \wedge ie.\texttt{covers} = 0, & \text{and} & ae.\texttt{ctxt\_covers} \wedge ie.\texttt{covers} = 0 \end{array}$$

must hold so that ae and ie can be combined, resulting in a new edge n with

$$(4)\quad \begin{array}{lll} n.\texttt{start} & \leftarrow & ae.\texttt{start} \\ n.\texttt{end} & \leftarrow & \max\{ae.\texttt{end}, ie.\texttt{end}\} \\ n.\texttt{rule} & \leftarrow & ae.\texttt{rule} \\ n.\texttt{dot\_pos} & \leftarrow & ae.\texttt{dot\_pos} + 1 \\ n.\texttt{covers} & \leftarrow & \begin{cases} ae.\texttt{covers} & \text{if } ie \text{ is context daughter} \\ ae.\texttt{covers} \vee ie.\texttt{covers} & \text{otherwise} \end{cases} \end{array}$$

$$n.\texttt{ctxt\_covers} \leftarrow \begin{cases} ae.\texttt{ctxt\_covers} \vee ie.\texttt{covers} & \text{if } ie \text{ is context daughter} \\ ae.\texttt{ctxt\_covers} & \text{otherwise.} \end{cases}$$

Suppose, for example, that the inactive edge ie=[2, 5, Q → c [d] e •, 001010, 000100] has just been popped from the agenda and added to the chart, and that the active edge ae=[0, 4, S → P • Q f, 110100, 000000] is already contained in the chart. We thus check whether these two edges can be combined. ae.next_pos is 2 (we start counting at 0) and equals ie.start. The left-hand side category of ie and the symbol to the right of the dot in ae are both Q. Furthermore, ae.covers ∧ ie.covers = 110100∧001010 = 0 and ae.ctxt_covers∧ie.covers = 000000∧001010 = 0. All necessary conditions are fulfilled, so ae and ie are combined, yielding the new edge [0, 5, S → P Q • f, 111110, 000000], which is added to the agenda. Additionally, ip is combined with each rule in the given DPSG whose left-corner category equals the left-hand side category of ip. Suppose that the input DPSG contains a rule X → Q R. This would result in a new edge [2, 2, X → • Q R, 000000, 000000] to be added to the agenda[4].

The mechanisms described above are sufficient to ensure that constituents corresponding to right-hand side symbols of edges constructed by our parser form adjacency sequences, as is required by the DPSG formalism. One problem remains, though. Consider the DPSG in Figure 2a.

a) S → a P X
P → b [Y] e
X → c d
Y → c d

b)

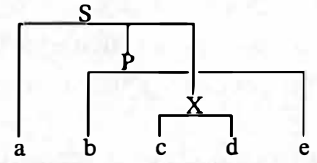


Figure 2: The discotree in (b) should not be constructed from the DPSG in (a)

The parser would construct the discotree shown in Figure 2b, even though the P constituent assumes a context daughter Y dominating the input symbols c and d, whereas the direct daughter corresponding to Y in span is headed by an X. To put it differently, the X constituent is not licensed as a context daughter of P according to rule P → b [Y] e. In order to prevent the construction of such ill-formed structures, our algorithm additionally checks that context daughters and corresponding direct daughters match in category[5].

In order to be able to reconstruct all (partial) parse trees corresponding to an edge e, we associate with e a list of pairs of edge pointers (field children), each pair representing one possible way in which e has been constructed from an active and an inactive edge. When a new edge n is to be added to the chart, we check whether an edge e corresponding to an equivalent subparse already exists. If yes, the children list of n is appended to e's list, and n is discarded. Two edges correspond to equivalent subparses if they would behave in the same way would parsing proceed without them being merged. It turns out that e and n can only be merged safely, if they both correspond to a continuous constituent[6]. Additionally, if e and n are inactive, they have to agree in the values of their fields start, end and lhs. If the two edges are active, their fields start, end, rule and dot_pos have to contain the same values.

[4]To improve efficiency, we perform a lookahead test for each new active edge n to check whether the remaining symbols on n's right-hand side can be "satisfied" at all with the input symbols not already dominated by n. If not, we discard n. The details of this test are beyond the scope of this paper.

[5]As far as we can tell from the information available in (Bunt, 1991) and (van der Sloot, 1990), Bunt's parser would build the discotree in Figure 2b when presented with the DPSG in Figure 2a.

[6]We say that an edge e is *continuous*, if and only if $\prod_{i=e.\texttt{start}}^{e.\texttt{end}-1} b_i = 1$ where b_i is the i-th bit in e.covers.

Note, finally, that the parsing algorithm described above constructs context-free trees when presented with a CFG as input. The additional mechanisms for dealing with discotrees come into play only if at least some of the input rules contain context daughters.

4 Computing the Most Probable Parse

As in the CFG/PCFG case, we can extend a DPSG with a probability distribution on its rule set, yielding a *Probabilistic Discontinuous Phrase Structure Grammar* (PDPSG).

Definition 5 (Probabilistic Discontinuous Phrase Structure Grammar)
A Probabilistic Discontinuous Phrase Structure Grammar (PDPSG) *is a quintuple* $\langle V_N, V_T, S, R, P\rangle$, *where* $\langle V_N, V_T, S, R\rangle$ *is a DPSG and* P *is a function* $R \mapsto [0,1]$ *that assigns a probability to each rule such that* $\forall X \in V_N: \sum_{\Delta} P(X \to \Delta) = 1$. ∎

In words, we assign a probability to each rule such that the probabilities of all rules with the same left-hand side category sum to 1. Note that Δ denotes right-hand sides of DPSG rules and as such contains information about which symbols are marked as context daughters. We define the probability of a string of terminal symbols as the sum of the probabilities of all parse trees that yield this string. The probability of a parse tree is the product of the probabilities of all rule instances used in constructing this tree.

We adapted the algorithm for computing the MPP given a PCFG (see e.g. (Brants, 1999) for a nice presentation) to perform the same task for PDPSGs. The algorithm maintains a set of accumulators $\delta_n(Y)$ for each symbol $Y \in V$ and each node n in the parse tree. Contrary to the context-free case, a node in a discontinuous tree cannot uniquely be described by the starting and ending position of its corresponding subparse. We instead use two bit-strings for this purpose, as explained in the previous section, which we shall below denote by b (=`covers`) and $\hat{b}$ (=`ctxt_covers`) for the sake of brevity. A node n dominates all terminal symbols for which the corresponding bit in b is set (equals 1); $\hat{b}$ indicates which input symbols are dominated by context daughters of n. That is to say, given a DPSG $G = \langle V_N = \{X^1, \ldots, X^N\}, V_T, X^1, R, P\rangle$ and an input string $w_{0,T} = w_0, \ldots, w_{T-1}$, we define the accumulators as variables $\delta_{b,\hat{b}}(Y)$, where $Y \in V$ and $b, \hat{b} \in \{0,1\}^T$, and compute their values bottom-up as follows.

Initialization:

$$\text{(5)}\quad \delta_{b,\hat{b}}(Z) = \begin{cases} 1 & \text{if } Z = w_{t-1} \\ 0 & \text{if } Z \neq w_{t-1} \end{cases} \qquad 1 \leq t \leq T,\ Z \in V_T,\ b = 0^{t-1}\,1\,0^{T-t},\ \hat{b} = 0^T$$

Recursion:

$$\text{(6)}\quad \delta_{b,\hat{b}}(X^i) = \max_{\substack{(X^i \to \Delta) \in R,\\ \Delta = Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k},\\ \mathrm{AdjSeq}\left(Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}\right),\\ b = \bigvee_{\substack{1 \leq j \leq k \\ c_j = 0}} b_j,\ \hat{b} = \bigvee_{\substack{1 \leq j \leq k \\ c_j = 1}} b_j}} P(X^i \to \Delta) \prod_{\substack{1 \leq j \leq k \\ c_j = 0}} \delta_{b_j,\hat{b}_j}(Y^j) \qquad 1 \leq i \leq N,\ b, \hat{b} \in \{0,1\}^T$$

Termination:

$$\text{(7)}\quad P_{\mathrm{MPP}}(w_{0,T} \mid G) = \delta_{b,\hat{b}}(X^1) \qquad b = 1^T,\ \hat{b} = 0^T$$

We initialize the algorithm by assigning a value of 1 to $\delta_{b,\hat{b}}(Z)$ for each terminal symbol $w_{t-1} = Z$ $(1 \leq t \leq T)$ in the input, such that b is a bit-string in which only the t-th bit is set and $\hat{b}$ is an all-zero

bit-string. Next, we recursively compute the values of accumulators for larger and larger subparses. For each $\delta_{b,\hat{b}}(X^i)$, the algorithm explores all ways in which the subparse $X^i_{b,\hat{b}}$ can be constructed[7] from smaller parts and maximises over the respective probabilities. The basic idea behind the algorithm is thus the same as in the PCFG case. The PDPSG version of the algorithm differs from the PCFG one only with respect to how partial parses are combined to yield larger constituents. Each alternative for $X^i_{b,\hat{b}}$ is based on a DPSG rule $X^i \rightarrow Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}$, where $Y^j_{b_j,\hat{b}_j}$ denotes the subparse corresponding to the j-th right-hand side symbol of the rule and $c_j \in \{0,1\}$ indicates whether this symbol is marked as a context daughter. A (partial) parse $X^i_{b,\hat{b}}$ can be constructed from the smaller parts corresponding to the $Y^j_{b_j,\hat{b}_j}$'s if the latter form an adjacency sequence, as indicated by the term $\mathrm{AdjSeq}\left(Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}\right)$ in the recursion formula (6). Given the two bit-strings of each subconstituent, we can perform the checks described in the previous section to ensure that the subconstituents form an adjacency sequence.

The probability of each alternative is computed as the product of the probability of the rule used to construct it and the accumulators for all right-hand side symbols corresponding to its direct daughters, and $\delta_{b,\hat{b}}(X^i)$ is set to the maximum of all alternatives' probabilities. After the values for all $\delta_{b,\hat{b}}(X^i)$ have been computed, the probability of the MPP is that of the accumulator for parse trees headed by the start symbol of the input grammar (X^1) that dominate all terminal symbols in the input ($b = 1^T$, $\hat{b} = 0^T$).

The computation of these accumulators can easily be incorporated within a slightly modified version of our DPSG parser. To this end, we associate each edge with an additional field `prob` that stores the accumulator value for the subparse the edge corresponds to. Edges for the terminal symbols in the input added to the agenda during initialization are assigned a `prob` value of 1, mirroring Equation (5). A new active edge that results from a DPSG rule matching an inactive edge that has been added to the chart inherits its probability from the underlying rule. When an active edge *ae* and an inactive edge *ie* are combined, the probability of the new edge is computed as the product of *ae*'s and *ie*'s probabilities, if *ie* is a direct daughter in the new edge, and is set to *ae*'s probability otherwise (cf. Equation (6)). Furthermore, we do not store more than one subparse with each edge, but only the most probable one found so far. In other words, if we encounter a new edge that would be merged with an already existing one, we instead discard the edge that has a lower probability value. This implies that the discarded edge must not already be contained in a higher subparse because, if this were the case, the probability of the higher subparse would have been computed incorrectly. To prevent this, we organise the agenda in such a way that parsing proceeds strictly bottom-up. Finally, when the agenda is empty, the edge that is headed by the start symbol of the grammar and that spans the entire input represents the Most Probable Parse for the given input sentence. The probability of the MPP is contained in the `prob` field of this edge (cf. Equation (7)).

Again, this algorithm can be used to find the MPP for either a PCFG or a PDPSG, depending on whether the input grammar contains rules with context daughters.

[7] We use $X^i_{b,\hat{b}}$ to denote a subparse headed by the nonterminal symbol X^i whose direct daughters dominate the input symbols for which the corresponding bits in b are set and whose context daughters dominate the symbols for which the bits in $\hat{b}$ are set.

5 Experiments

All experiments we conducted were based on grammars extracted from the NEGRA corpus (Skut et al., 1997), which consists of German newspaper text. The version we used contains 20 571 sentences. All sentences are part-of-speech tagged, and their syntactic structures are represented as discontinuous trees. In a preprocessing step, we removed sentences without syntactic structure, attached punctuation marks to suitable nodes in the discotree, and removed unbound tokens. The corpus of discotrees thus obtained consists of 19 445 sentences with an average length of 17 tokens. In order to have some sort of baseline against which we could compare our results from PDPSG parsing, we additionally transformed the discotrees in this corpus to context-free trees by re-attaching all continuous parts of discontinuous constituents to higher nodes. We kept 1 005 sentences from each corpus to be used in case of unforeseen events and split the remaining corpus into a training set of 16 596 sentences (90%) and a test set of 1 844 sentences (10%), such that both test sets (and both training sets) contained the same sentences, albeit with different structures. We extracted rule instances from both training sets[8] and computed each rule's probability as the ratio of its frequency to the frequency of all rules with the same left-hand side category, thus obtaining a PCFG and a PDPSG[9].

Due to limited computational resources, we restricted the two test sets to sentences with a maximum length of 15 tokens. We ran our parser on the part-of-speech tag sequences of these 959 sentences, once with the PCFG as input, and once using the PDPSG. The parse trees from the PCFG (PDPSG) experiment were then compared against the correct context-free trees (discotrees) in the test set. We determined average CPU time[10] per sentence and various accuracy measures for both experiments, which are summarised in Figure 3.

	PCFG	PDPSG
Precision	79.24%	77.75%
Recall	78.09%	76.81%
F-score	78.66%	77.28%
Labeled precision	75.12%	73.61%
Labeled recall	74.03%	72.72%

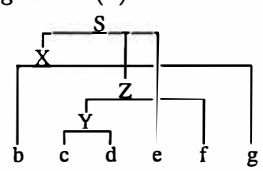
	PCFG	PDPSG
Labeled F-score	74.57%	73.16%
Coverage	96.35%	96.04%
Exact matches	39.83%	39.00%
Structural matches	43.07%	42.23%
CPU time per sent.	0.53 secs	24.78 secs

Figure 3: Accuracy results and average CPU time per sentence for both experiments

[8]For context-free trees, we extract rules in the usual way. For each nonterminal n in the tree, a rule is generated in which n's label constitutes the left-hand side and the labels of n's direct daughters form the right-hand side. Given a discotree, we also need to determine context daughters of discontinuous constituents, such that right-hand sides form adjacency sequences. This is done as follows. For each terminal t between two direct daughters d_1 and d_2 of a nonterminal n such that n does not dominate t, we determine the highest node dominating t that does not dominate n and whose yield is contained within the bounds of n. These nodes are the context daughters of n between d_1 and d_2 (with duplicates removed). For instance, we extract the rules in (i) from the discotree given in (ii).

(i) S → X Z e
X → b [Z] [e] g
Z → Y [e] f
Y → c d

(ii)



[9]The PCFG training corpus gave rise to 120 831 rule instances and 18 709 rules, of which 13 419 appear only once in the corpus. The PDPSG consists of 21 965 rules (generated from 123 528 rule instances); 16 317 of these rules appear only once in the training corpus.

[10]On a Sun Ultra Sparc 300 MHz with 1 GB main memory running Solaris 2.6.

Precision is defined as the percentage of constituents proposed by our parser which are actually correct according to the tree in the corpus. "Correct" means that the two constituents dominate the same terminals; for the different "labeled" measures, the node labels must match in addition. Recall is the percentage of constituents in the test set trees which are found by our parser. We define F-score as the harmonic mean of recall R and precision P, that is, as $F = \frac{2PR}{P+R}$. Coverage denotes the percentage of sentences for which a complete parse has been found by our parser. A proposed parse tree with an F-score of 100% is a structural match; if the *labeled* F-score is 100%, the tree is called an exact match.

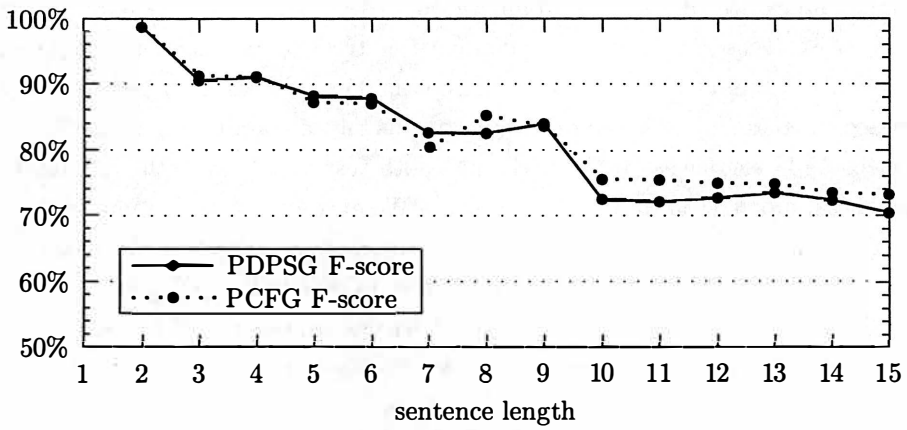


Figure 4: F-scores from both experiments plotted against sentence length

In comparing the results from the PDPSG experiment against the PCFG results, it is important to keep in mind that parsing with discotrees is a much harder task than parsing with context-free trees. The complexity of the latter is cubic in the sentence length, whereas our parsing algorithm for PDPSG takes, in the worst case, exponential time (see also (Reape, 1991)). Therefore, unsurprisingly, the average CPU time per sentence in the PDPSG experiment is almost 50 times larger than in the PCFG case. To make things worse, the difference in running time would be even larger for longer sentences.

The good news, on the other hand, is that accuracy drops only slightly (see Figure 4) when we accommodate within our parser the possibility of constituents to be discontinuous.

6 Conclusion and Future Work

We have presented a probabilistic extension of Discontinuous Phrase Structure Grammar, a formalism suitable for describing restricted discontinuities in a perspicuous single-level representation. Furthermore, we have developed a parser that can be used with probabilistic and non-probabilistic versions of both Context-Free Grammar and Discontinuous Phrase Structure Grammar, constructing context-free or discontinuous trees, respectively.

Although the probabilistic method applied is rather simplistic, the accuracy results obtained in the PDPSG experiment are comparable to the PCFG results. A severe drawback of our approach is its worst-case exponential running time. Future work will thus be primarily aimed at reducing this complexity (and additionally at increasing accuracy). There are at least two avenues towards this goal.

Firstly, we could try to restrict the formalism of DPSG further with respect to the kinds of discontinuities it is capable of representing. Vogel and Erjavec (1994) presented a restricted version of DPSG, called DPSG^R, and claimed that this formalism is properly contained in the class of mildly context-sensitive languages and that consequently a polynomial time recognition procedure exists for it. Note, however, that DPSG^R does not allow for the description of cross-serial dependencies; it is therefore too restricted to represent the discontinuity phenomena occuring in the NEGRA corpus[11]. A similar account, although within a different grammatical framework, was provided by Müller (1999) who presented an HPSG system in which linguistically motivated constraints on (dis)continuity were imposed, which led to a notable decrease in running time.

An alternative, orthogonal approach towards faster mechanisms for probabilistic parsing with discontinuous constituents is to stick to DPSG and try to find suitable approximation algorithms that reduce the *observed* running time. Several such approaches exist in the literature on probabilistic parsing, mostly based on (P)CFGs. Since DPSG is a straightforward extension of CFG, we expect that at least some of these can be extended with reasonable effort to also deal with discontinuous trees. In fact, the work presented in this paper serves as a first indication confirming this intuition. A more sophisticated approach could, for instance, be based on the statistical parser devised by Ratnaparkhi (1997). He utilized a set of procedures implementing certain actions to incrementally construct (context-free) parse trees. The probabilities of these actions were computed by Maximum Entropy models based on certain syntactic characteristics (features) of the current context, and effectively rank different parse trees. A beam search heuristic was used that attempts to find the highest scoring parse tree. In order to extend this approach to DPSG, we would need a different set of procedures capable of constructing discontinuous trees and a suitable set of features.

Edge-based best-first chart parsing (Charniak, Goldwater, & Johnson, 1998; Charniak & Caraballo, 1998) is another very promising approach. Charniak et al. (1998) proposed to judge edges according to some probabilistic figure of merit that is meant to approximate the likelihood that an edge will ultimately appear in a correct parse. Edges are processed in decreasing order of this value until a complete parse has been found (or perhaps several ones), leaving edges on the agenda. They reported results equivalent to the best prior ones using only one twentieth the number of edges.

The edge-based best-first parsing approach could easily be applied to DPSG and our chart parsing algorithm as well. To this end, we would need to organise the agenda as a priority queue, so that edges are processed in decreasing order of their respective figure of merit values. We are confident that suitable figures of merits can be found for DPSG chart parsing that will lead to a significant decrease in running time.

References

Blevins, J. P. (1990). *Syntactic complexity: Evidence for discontinuity and multidomination.* PhD thesis, University of Massachusetts, Amherst, MA.

Brants, T. (1999). *Tagging and parsing with cascaded Markov models — automation of corpus annotation.* PhD thesis, University of the Saarland, Saarbrücken, Germany.

[11] Bresnan, Kaplan, Peters, and Zaenen (1982) provided additional evidence that formalisms capable of representing cross-serial dependencies are desirable.

Bresnan, J., Kaplan, R. M., Peters, S., & Zaenen, A. (1982). Cross-serial dependencies in dutch. *Linguistic Inquiry, 13*(4), 613–635.

Bunt, H. (1991). Parsing with discontinuous phrase structure grammar. In M. Tomita (Ed.), *Current issues in parsing technology* (pp. 49–63). Dordrecht, Boston, London: Kluwer Academic Publishers.

Bunt, H. (1996). Formal tools for describing and processing discontinuous constituency structure. In H. Bunt & A. van Horck (Eds.), *Discontinuous constituency* (pp. 63–83). Berlin, New York: Mouton de Gruyter.

Charniak, E., & Caraballo, S. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics, 24*(2), 275–298.

Charniak, E., Goldwater, S., & Johnson, M. (1998). Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora.* Montreal, Canada.

Johnson, M. (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd ACL meeting* (pp. 127–132). Chicago: Association for Computational Linguistics.

McCawley, J. D. (1982). Parentheticals and discontinuous constituent structure. *Linguistic Inquiry, 13*(1), 91–106.

Müller, S. (1999). Restricting discontinuity. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium 1999 (NLPRS '99).* Peking.

Plaehn, O. (1999). *Probabilistic parsing with discontinuous phrase structure grammar.* Diplom thesis, University of the Saarland, Saarbrücken, Germany. (`http://www.coli.uni-sb.de/~plaehn/papers/dt.html`)

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP-97.* Providence, RI.

Reape, M. (1991). Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In T. van der Wouden & W. Sijtsma (Eds.), *Computational Linguistics in the Netherlands. Papers from the First CLIN-meeting.* Utrecht: Utrecht University-OTS.

Skut, W., Krenn, B., Brants, T., & Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97.* Washington, DC.

van der Sloot, K. (1990). *The TENDUM 2.7 parsing algorithm for DPSG* (ITK Research Memo). Tilburg: ITK.

Vogel, C., & Erjavec, T. (1994). Restricted discontinuous phrase structure grammar and its ramifications. In C. Martin-Vide (Ed.), *Current issues in mathematical linguistics* (pp. 131–140). Amsterdam: Elsevier.