The Problem of Computing the Most Probable Tree in Data-Oriented Parsing and Stochastic Tree Grammars

Rens Bod
Institute for Logic, Language and Computation
Department of Computational Linguistics
University of Amsterdam
Spuistraat 134, 1012 VB Amsterdam
The Netherlands
rens@mars.let.uva.nl

Abstract

We deal with the question as to whether there exists a polynomial time algorithm for computing the most probable parse tree of a sentence generated by a data-oriented parsing (DOP) model. (Scha, 1990; Bod, 1992, 1993a). Therefore we describe DOP as a stochastic tree-substitution grammar (STSG). In STSG, a tree can be generated by exponentially many derivations involving different elementary trees. The probability of a tree is equal to the sum of the probabilities of all its derivations.

We show that in STSG, in contrast with stochastic context-free grammar, the Viterbi algorithm cannot be used for computing a most probable tree of a string. We propose a simple modification of Viterbi which allows by means of a "*select-random*" search to *estimate* the most probable tree of a string in polynomial time.

Experiments with DOP on ATIS show that only in 68% of the cases, the most probable derivation of a string generates the most probable tree of that string. Therefore, the parse accuracy obtained by the most probable trees (96%) is dramatically higher than the parse accuracy obtained by the most probable derivations (65%).

It is still an open question whether the most probable tree of a string can be *deterministically* computed in polynomial time.

1 Data-Oriented Parsing

A Data-Oriented Parsing model (Scha, 1990; Bod, 1992, 1993a) is characterized by a corpus of analyzed language utterances, together with a set of operations that combine sub-analyses from the corpus into new analyses. We will limit ourselves in this paper to corpora with purely syntactic annotations. For the semantic dimension of DOP, the reader is referred to (van den Berg et al., 1994). Consider the imaginary example corpus consisting of only two trees in figure 1. We will assume one operation for combining subtrees. This operation is called "composition", and is indicated by the infix operator $\circ$. The composition of t and u, $t \circ u$, yields a copy of t in which its leftmost nonterminal leaf node has been identified with the root node of u (i.e., u is *substituted* on the leftmost nonterminal leaf node of t). For reasons of simplicity we will write in the following $(t \circ u) \circ v$ as: $t \circ u \circ v$.

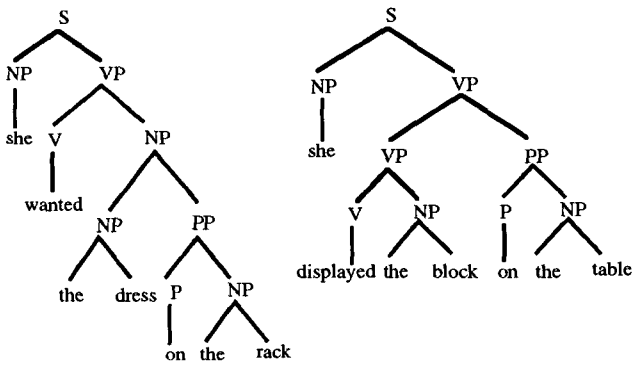

Figure 1. Example corpus of two trees.

Now the (ambiguous) sentence *"She displayed the dress on the table"* can be parsed by combining subtrees from the corpus. For instance:

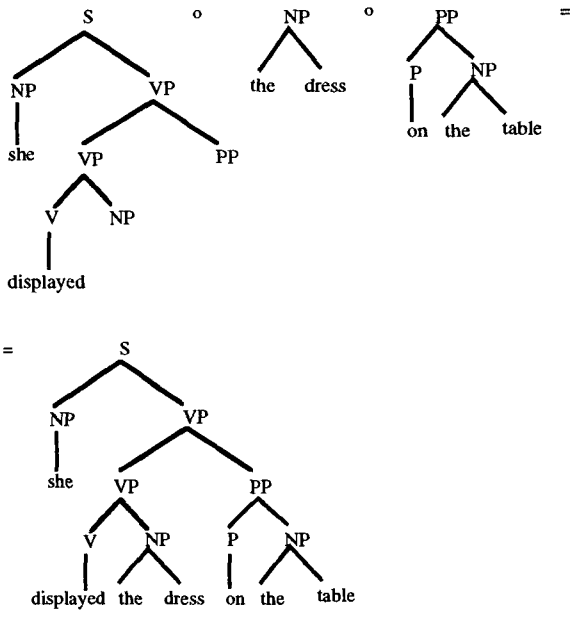

Figure 2. Derivation and parse tree for *"She displayed the dress on the table"*

As the reader may easily ascertain, a different derivation may yield a different parse tree. However, a different derivation may also very well yield the same parse tree; for instance:

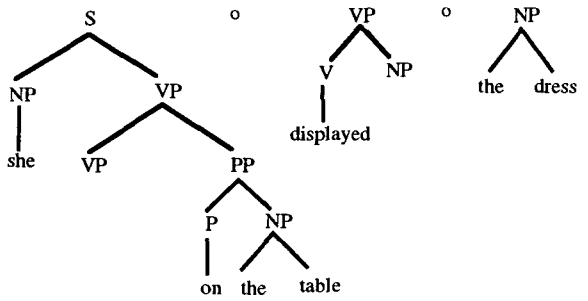

Figure 3. Different derivation generating the same parse tree for *"She displayed the dress on the table"*

or

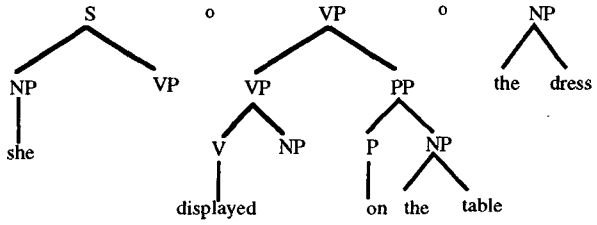

Figure 4. Another derivation generating the same parse tree for *"She displayed the dress on the table"*

Thus, a parse tree can have several derivations involving different subtrees. Using the corpus for our stochastic estimations, we estimate the probability of substituting a certain subtree on a specific node as the probability of selecting this subtree among all subtrees in the corpus that could be substituted on that node.[1] The probability of a derivation can be computed as the product of the probabilities of the substitutions that it involves. The probability of a parse tree is equal to the probability that any of its derivations occurs, which is the sum of the probabilities of all derivations of that parse tree. Finally, the probability of a word string is equal to the sum of the probabilities of all its parse trees.

2 DOP as a Stochastic Tree-Substitution Grammar

In order to deal with the problem of computing the most probable parse tree of a string, it is convenient to describe DOP as a "Stochastic Tree-Substitution Grammar" (STSG). STSG can be seen as a generalization over DOP, where the elementary trees of STSG are the subtrees of DOP, and the probabilities of the elementary trees are the substitution-probabilities of the corresponding subtrees of DOP (Bod, 1993c).

A *Stochastic Tree-Substitution Grammar* G is a five-tuple $<V_N, V_T, S, R, P>$ where
V_N is a finite set of nonterminal symbols.
V_T is a finite set of terminal symbols.
$S \in V_N$ is the distinguished symbol.
R is a finite set of elementary trees whose top nodes and interior nodes are labeled by nonterminal symbols and whose yield nodes are labeled by terminal or nonterminal symbols.
P is a function which assigns to every elementary tree $t \in R$ a probability $p(t)$. For a tree t with a root α, $p(t)$ is interpreted as the probability of substituting t on α. We require, therefore, that $0 < p(t) \leq 1$ and $\Sigma_{t:root(t)=\alpha} p(t) = 1$.

If t_1 and t_2 are trees such that the *leftmost nonterminal yield node* of t_1 is equal to the *root* of t_2, then $t_1 \circ t_2$ is the tree that results from substituting t_2 for this leftmost nonterminal yield node in t_1. The partial function $\circ$ is called *leftmost substitution*. For reasons of conciseness we will use the term substitution for leftmost substitution.

A *leftmost derivation* generated by an STSG G is a tuple of trees $<t_1,...,t_n>$ such that $t_1,...,t_n$ are elements of R, the root of t_1 is labeled by S and the yield of $t_1 \circ ... \circ t_n$ is labeled by terminal symbols. The set of leftmost derivations generated by G is thus given by $Derivations(G) = \{<t_1,...,t_n> \mid t_1,...,t_n \in R \wedge root(t_1) = S \wedge yield(t_1 \circ ... \circ t_n) \in V_T^+\}$. For convenience we will use the term derivation for leftmost derivation. A derivation $<t_1,...,t_n>$ is called a derivation of tree T, iff $t_1 \circ ... \circ t_n = T$. A derivation $<t_1,...,t_n>$ is called a derivation of string s, iff $yield(t_1 \circ ... \circ t_n) = s$. The probability of a derivation $<t_1,...,t_n>$ is defined as $p(t_1) \cdot ... \cdot p(t_n)$.

A *parse tree* generated by an STSG G is a tree T such that there is a derivation $<t_1,...,t_n> \in Derivations(G)$ for which $t_1 \circ ... \circ t_n = T$. The set of parse trees, or *tree language*, generated by G is given by $Parses(G) = \{T \mid \exists <t_1,...,t_n> \in Derivations(G) : t_1 \circ ... \circ t_n = T\}$. For reasons of conciseness we will often use the terms *parse* or *tree* for a parse tree. A parse whose yield is equal to string s, is called a parse of s. The probability of a parse is defined as the sum of the probabilities of all its derivations.

A *string* generated by an STSG G is an element of V_T^+ such that there is a parse generated by G whose yield is equal to the string. The set of strings, or *string language*, generated by G is given by $Strings(G) = \{s \mid \exists T : T \in Parses(G) \wedge s = yield(T)\}$. The probability of a string is defined as the sum of the probabilities of all its parses. This means that the probability of a string is also equal to the sum of the probabilities of all its derivations.

[1] Very small frequencies are smoothed by Good-Turing.

3 Computing a most probable parse tree in STSG

In order to deal with the problem of computing the most probable parse tree of a sentence, we will distinguish between parsing and disambiguation. By parsing we mean the creation of a parse forest for an input sentence. By disambiguation we mean the selection of the most probable parse[2] from the forest. The creation of a parse forest is an intermediate step for computing the most probable parse.

3.1 Parsing

From the way STSG combines elementary trees by means of substitution, it follows that an input sentence can be parsed by the same algorithms as (S)CFGs. Every elementary tree t is used as a context-free rewrite rule $root(t) \rightarrow yield(t)$. Given a chart parsing algorithm, an input sentence of length n can be parsed in n^3 time.

In order to obtain a chart-like forest for a sentence parsed in STSG, we need to label the well-formed substrings in the chart not only with the syntactic categories of that substring but with the full elementary trees t that correspond to the use of the derived rules $root(t) \rightarrow yield(t)$. Note that in a chart-like forest generated by an STSG, different derivations that generate a same tree do not collapse. We will therefore talk about a *derivation forest* generated by an STSG (cf. Sima'an et al., 1994).

The following formal example illustrates what a derivation forest of a string may look like. In the example, we leave out the probabilities, which are needed only in the disambiguation process. The visual representation comes from (Kay, 1980): every entry (i,j) in the chart is indicated by an edge and spans the words between the i-th and the j-th position of a sentence. Every edge is labeled with the elementary trees that denote the underlying phrase. The example-STSG consists of the following elementary trees:

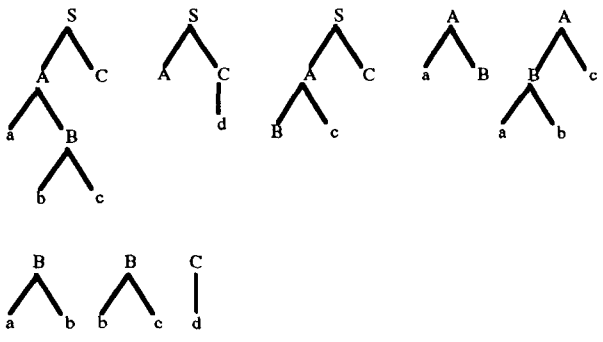

Figure 5. Elementary trees of an example-STSG

For the input string **abcd**, the following derivation forest is then obtained:

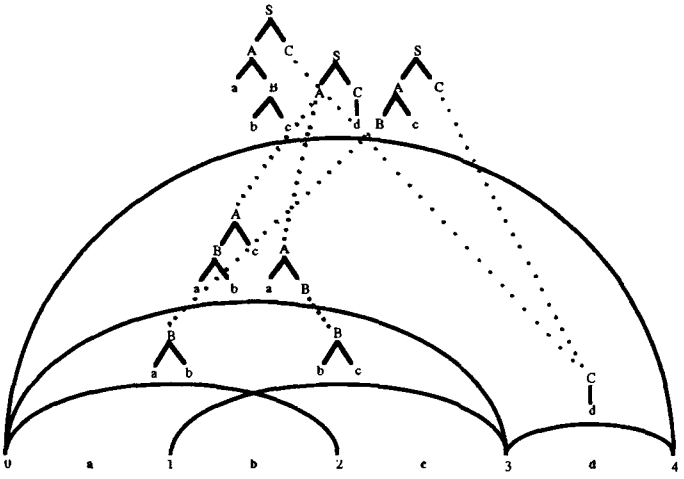

Figure 6. Derivation forest for **abcd**

Note that different derivations in the forest generate the same tree. By exhaustively unpacking the forest, four different derivations generating two different trees are obtained. We may ask whether we can pack the forest by collapsing spurious derivations. Unfortunately, no efficient procedure is known that accomplishes this (remember that there can be exponentially many derivations for one tree).

3.2 Disambiguation

Cubic time parsing does not guarantee cubic time disambiguation, as a sentence may have exponentially many parses and any such parse may have exponentially many derivations. Therefore, in order to find the most probable parse of a sentence, it is not efficient to compare the probabilities of the parses by exhaustively unpacking the chart. Even for determining the probability of one parse, it is not efficient to add the probabilities of all derivations of that parse.

3.2.1 Viterbi optimization is not feasible for finding the most probable parse

There exists a heuristic optimization algorithm, known as Viterbi optimization, which selects on the basis of an SCFG the most probable derivation of a sentence in cubic time (Viterbi, 1967; Fujisaki et al., 1989; Jelinek et al., 1990). In STSG, however, the most probable derivation does not necessarily generate the most probable parse, as the probability of a parse is defined as the sum of the probabilities of all its derivations. Thus, there is an important question as to whether we can adapt the Viterbi algorithm for finding the most probable parse.

To understand the difficulty of the problem, we look in more detail at the Viterbi algorithm. The basic idea of the Viterbi algorithm is the early pruning of low probability subderivations in a bottom-up fashion. Two different subderivations of the same part of the sentence and whose resulting

[2] Although theoretically there can be more than one most probable parse for a sentence, in practice a system that employs a non-trivial treebank tends to generate exactly one most probable parse for a given input sentence.

subparses have the same root can both be developed (if at all) to derivations of the whole sentence in the same ways. Therefore, if one of these two subderivations has a lower probability, then it can be eliminated. This is illustrated by a formal example in figure 7. Suppose that during bottom-up parsing of the string *abcd* the following two subderivations *d1* and *d2* have been generated for the substring *abc*. (Actually represented are their resulting subparses.)

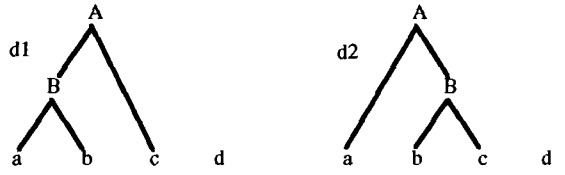

Figure 7. Two subparses for the string *abcd*

If the probability of *d1* is higher than the probability of *d2*, we can eliminate *d2* if we are only interested in finding the most probable derivation of *abcd*. But if we are interested in finding the most probable parse of *abcd* (generated by STSG), we are not allowed to eliminate *d2*. This can be seen by the following. Suppose that we have the additional elementary tree given in figure 8.

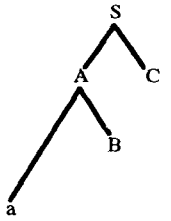

Figure 8. Elementary tree.

This elementary tree may be developed to the same tree that can be developed by *d2*, but not to the tree that can be developed by *d1*. And since the probability of a parse tree is equal to the sum of the probabilities of all its derivations, it is still possible that *d2* contributes to the generation of the most probable parse. Therefore we are not allowed to eliminate *d2*.

This counter-example does not prove that there is no heuristic optimization that allows polynomial time selection of the most probable parse. But it makes clear that a "*select-best*" search, as accomplished by Viterbi, is not adequate for finding the most probable parse in STSG. So far, it is unknown whether the problem of finding the most probable parse in a deterministic way is inherently exponential or not (cf. Sima'an et al., 1994). One should of course ask how often in practice the most probable derivation produces the most probable parse, but this can only be answered by means of experiments on real life corpora. Experiments on the ATIS corpus (see session 4) show that only in 68% of the cases the most probable derivation of a sentence generates the most probable parse of that sentence. Moreover, the parse accuracy obtained by the most probable parse is dramatically higher than the parse accuracy obtained by the parse generated by the most probable derivation.

3.2.2 Estimating the most probable parse by Monte Carlo search

We will leave it as an open question whether the most probable parse can be deterministically derived in polynomial time. Here we will ask whether there exists a polynomial time approximation procedure that *estimates* the most probable parse with an estimation error that can be made arbitrarily small.

We have seen that a "*select-best*" search, as accomplished by Viterbi, can be used for finding the most probable derivation but not for finding the most probable parse. If we apply instead of a select-best search, a "*select-random*" search, we can generate a random derivation. By iteratively generating a large number of random derivations we can estimate the most probable parse as the parse which results most often from these random derivations (since the probability of a parse is the probability that any of its derivations occurs). The most probable parse can be estimated as accurately as desired by making the number of random samples as large as desired. According to the Law of Large Numbers, the most often generated parse converges to the most probable parse. Methods that estimate the probability of an event by taking random samples are known as Monte Carlo methods (Hammersley & Handscomb, 1964).[3]

The selection of a random derivation is accomplished in a bottom-up fashion analogous to Viterbi. Instead of selecting the most probable subderivation at each node-sharing in the chart, a random subderivation is selected (i.e. sampled) at each node-sharing (that is, a subderivation that has n times as large a probability as another subderivation should also have n times as large a chance to be chosen as this other subderivation). Once sampled at the S-node, the random derivation of the whole sentence can be retrieved by tracing back the choices made at each node-sharing. Of course, we may postpone sampling until the S-node, such that we sample directly from the distribution of all S-derivations. But this would take exponential time, since there may be exponentially many derivations for the whole sentence. By sampling bottom-up at every node where ambiguity appears, the maximum number of different subderivations at each node-sharing is bounded to a constant (the total number of rules of that node), and therefore the time complexity of generating a random derivation of an input sentence is equal to the time complexity of finding the most probable derivation, $O(n^3)$. This is exemplified by the following algorithm.

[3] Note that Monte Carlo estimation of the most probable parse is more reliable than the estimation of the most probable parse by generating the n most probable derivations by Viterbi, since it might be that the most probable parse is exclusively generated by many low probable derivations. The Monte Carlo method is guaranteed to converge to the most probable parse.

Sampling a random derivation from a derivation forest
Given a derivation forest, of a sentence of n words, consisting of labeled entries (i,j) that span the words between the i-th and the j-th position of the sentence. Every entry is labeled with linked elementary trees, together with their probabilities, that constitute subderivations of the underlying subsentence. Sampling a derivation from the chart consists of choosing at every labeled entry (bottom-up, breadth-first) a random subderivation of each root-node:

```
for k := 1 to n do
 for i := 0 to n-k do
  for chart-entry (i,i+k) do
   for each root-node X do
        select[4] a random subderivation of root X
        eliminate the other subderivations
```

We now have an algorithm that selects a random derivation from a derivation forest. Converting this derivation into a parse tree gives a first estimation for the most probable parse. Since one random sample is not a reliable estimate, we sample a large number of random derivations and see which parse is generated most frequently. This is exemplified by the following algorithm. (Note that we might also estimate the most probable *derivation* by random sampling, namely by counting which derivation is sampled most often; however, the most probable derivation can be more effectively generated by Viterbi.)

Estimating the most probable parse (MPP)
Given a derivation forest for an input sentence:

```
repeat until the MPP converges
   sample a random derivation from the forest
   store the parse generated by the random derivation
   MPP := the most frequently occurring parse
```

There is an important question as to how long the convergence of the most probable parse may take. Is there a tractable upper bound on the number of derivations that have to be sampled from the forest before stability in the top of the parse distribution occurs? The answer is yes: the worst case time complexity of achieving a maximum estimation error ε by means of random sampling is $O(\varepsilon^{-2})$, independently of the probability distribution. This is a classical result from sampling theory (cf. Hammersley and Handscomb, 1964), and follows directly from Chebyshev's inequality. In practice, it means that the error ε is inversely proportional to the square-root of the number of random samples N and therefore, to reduce ε by a factor of k, the number of samples N needs to be increased k^2-fold. In practical experiments (see §4), we will limit the number of samples to a pre-determined, sufficiently large bound N.

What is the theoretical worst case time complexity of parsing and disambiguation together? That is, given an STSG and an input sentence, what is the maximal time cost of finding the most probable parse of a sentence? If we use a CKY-parser, the creation of a derivation forest for a sentence of n words takes $O(n^3)$ time. Taking also into account the size G of an STSG (defined as the sum of the lengths of the yields of all its elementary trees), the time complexity of creating a derivation forest is proportional to Gn^3. The time complexity of disambiguation is both proportional to the cost of sampling a derivation, i.e. Gn^3, and to the cost of the convergence by means of iteration, which is ε^{-2}. This means that the time complexity of disambiguation is given by $O(Gn^3\varepsilon^{-2})$. The *total* time complexity of parsing and disambiguation is equal to $O(Gn^3) + O(Gn^3\varepsilon^{-2}) = O(Gn^3\varepsilon^{-2})$. Thus, there exists a tractable procedure that estimates the most probable parse of an input sentence.

Notice that although the Monte Carlo disambiguation algorithm estimates the most probable parse of a sentence in polynomial time, it is not in the class of polynomial time decidable algorithms. The Monte Carlo algorithm cannot *decide* in polynomial time what is the most probable parse; it can only make the error-probability of the estimated most probable parse arbitrarily small. As such, the Monte Carlo algorithm is a probabilistic algorithm belonging to the class of *B*ounded error *P*robabilistic *P*olynomial time (*BPP*) algorithms.

We hypothesize that Monte Carlo disambiguation is also relevant for other stochastic grammars. It turns out that all stochastic extensions of CFGs that are stochastically richer than SCFG need exponential time algorithms for finding a most probable parse tree (cf. Briscoe & Carroll, 1992; Black et al., 1993; Magerman & Weir, 1992; Schabes & Waters, 1993). To our knowledge, it has never been studied whether there exist *BPP*-algorithms for these models. Alhough it is beyond the scope of our research, we conjecture that there exists a Monte Carlo disambiguation algorithm for at least Stochastic Tree-Adjoining Grammar (Schabes, 1992).

3.2.3 Psychological relevance of Monte Carlo disambiguation

As has been noted, an important difference between the Viterbi algorithm and the Monte Carlo algorithm is, that with the latter we never have 100% confidence. In our opinion, this should not be seen as a disadvantage. In fact, absolute confidence about the most probable parse does not have any significance, as the probability assigned to a parse is already an estimation of its actual probability. One may ask as to whether Monte Carlo is appropriate for modeling

[4] Let $\{ (e_1, p_1), (e_2, p_2), \ldots, (e_n, p_n) \}$ be a probability distribution of events $e_1, e_2, \ldots, e_n$; an event e_i is said to be randomly selected iff its probability of being selected is equal to p_i. In order to allow for "direct sampling", one must convert the probability distribution into a corresponding sample space for which holds that the frequency of occurrence f_i of each event e_i is a positive integer equal to Np_i, where N is the size of the sample space.

human sentence perception. The following lists some properties of Monte Carlo disambiguation that may be of psychological interest:

1. As mentioned above, Monte Carlo never provides 100% confidence about the best analysis. This corresponds to the psychological observation that people never have absolute confidence about their interpretation of an ambiguous sentence.
2. Although conceptually Monte Carlo uses the total space of possible analyses, it tends to sample only the most likely ones. Very unlikely analyses may only be sampled after considerable time, but it is not guaranteed that *all* analyses are found in finite time. This matches with experiments on human sentence perception where very implausible analyses are only perceived with great difficulty and after considerable time.
3. Monte Carlo does not necessarily give the same results for different sequences of samples, especially if different analyses in the top of the distribution are almost equally likely. In the case there is more than one most probable analysis, Monte Carlo does not converge to one analysis but keeps alternating, however large the number of samples is made. In experiments with human sentence perception, it has often been shown that different analyses can be perceived for one sentence. And in case these analyses are equally plausible, people perceive so-called fluctuation effects. This fluctuation phenomenon is also well-known in the perception of ambiguous visual patterns.
4. Monte Carlo can be made parallel in a very straightforward way: N samples can be computed by N processing units, where equal outputs are reinforced. The more processing units are employed, the better the estimation. However, since the number of processing units is finite, there is never absolute confidence. This has some similarity with the Parallel Distributed Processing paradigm for human (language) processing (Rumelhart & McClelland, 1986).

4 Experiments

In this section, we report on experiments with an implementation of DOP that parses and disambiguates part-of-speech strings. In (Bod, 1995) it is shown how DOP is extended to parse word strings that possibly contain unknown words.

4.1 The test environment

For our experiments, we used a manually corrected version of the Air Travel Information System (ATIS) spoken language corpus (Hemphill et al., 1990) annotated in the Pennsylvania Treebank (Marcus et al., 1993). We employed the "blind testing" method, dividing the corpus into a 90% training set and a 10% test set by randomly selecting sentences. The 675 trees from the training set were converted into their subtrees together with their relative frequencies, yielding roughly $4*10^5$ different subtrees. The 75 part-of-speech sequences from the test set served as input strings that were parsed and disambiguated using the subtrees from the training set. As motivated in (Bod, 1993b), we use the notion of *parse accuracy* as our accuracy metric, defined as the percentage of the test strings for which the most probable parse is *identical* to the parse in the test set.

4.2 Accuracy as a function of subtree-depth

It is one of the most essential features of DOP, that arbitrarily large subtrees are taken into consideration to estimate the probability of a parse. In order to test the usefulness of this feature, we performed different experiments constraining the *depth* of the subtrees. The following table shows the results of seven experiments for different maximum depths of the training set subtrees. The accuracy refers to the parse accuracy at 400 randomly sampled parses, and is rounded off to the nearest integer. The CPU time refers to the average CPU time per string employed by a Spark II.

depth of subtrees	parse accuracy	CPU time (hours)
1	52 %	.04 h
≤2	87 %	.21 h
≤3	92 %	.72 h
≤4	93 %	1.6 h
≤5	93 %	1.9 h
≤6	95 %	2.2 h
unbounded	96 %	3.5 h

Table 1. Parse results on the ATIS corpus

The table shows a dramatic increase in parse accuracy when enlarging the maximum depth of the subtrees from 1 to 2. (Remember that for depth one, DOP is equivalent to a stochastic context-free grammar.) The accuracy keeps increasing, at a slower rate, when the depth is enlarged further. The highest accuracy is obtained by using all subtrees from the training set: 72 out of the 75 sentences from the test set are parsed correctly. Thus, the accuracy increases if larger subtrees are used, though the CPU time increases considerably as well.

4.3 Does the most probable derivation generate the most probable parse?

Another important feature of DOP is that the probability of a resulting parse tree is computed as the sum of the probabilities of all its derivations. Although the most probable parse of a sentence is not necessarily generated by the most probable derivation of that sentence, there is a question as to how often these two coincide. In order to study this, we also calculated the *derivation accuracy*, defined as the percentage of the test strings for which the parse generated by the most probable derviation is identical to the parse in the test set. The following table shows the derivation accuracy against the parse accuracy for the 75 test set strings from the ATIS corpus, using different maximum depths for the corpus subtrees.

depth of subtrees	derivation accuracy	parse accuracy
1	52%	52%
≤2	47%	87%
≤3	49%	92%
≤4	57%	93%
≤5	60%	93%
≤6	65%	95%
unbounded	65%	96%

Table 2. Derivation accuracy vs. parse accuracy

The table shows that the derivation accuracy is equal to the parse accuracy if the depth of the subtrees is constrained to 1. This is not surprising, as for depth 1, DOP is equivalent with SCFG where every parse is generated by exactly one derivation. What is remarkable, is, that the derivation accuracy decreases if the depth of the subtrees is enlarged to 2. If the depth is enlarged further, the derivation accuracy increases again. The highest derivation accuracy is obtained by using all subtrees from the corpus (65%), but remains far behind the highest parse accuracy (96%). From this table we conclude that if we are interested in the most probable analysis of a string we must not look at the probability of the *process* of achieving that analysis but at the probability of the *result* of that process.

4.4 The significance of once-occurring subtrees

There is an important question as to whether we can reduce the "grammar constant" of DOP by eliminating very infrequent subtrees, without affecting the parse accuracy. In order to study this question, we start with a test result. Consider the test set sentence "`Arrange the flight code of the flight from Denver to Dallas Worth in descending order`", which has the following parse in the test set:

```
(
 (S (NP *)
    (VP VB/Arrange
        (NP (NP DT/the NN/flight NN/code)
            (PP IN/of
                (NP (NP DT/the NN/flight)
                    (PP (PP IN/from
                            (NP NP/Denver))
                        (PP TO/to
                            (NP NP/Dallas
                                NP/Worth))))))
        (PP IN/in
            (NP (VP VBG/descending)
                NN/order)))
  .))
```

The corresponding p-o-s sequence of this sentence is the test set string "`VB DT NN NN IN DT NN IN NP TO NP NP IN VBG NN`". At subtree-depth ≤ 2, the following most probable parse was estimated for this string (where for reasons of readability the words are added to the p-o-s tags):

```
(
 (S (NP *)
    (VP VB/Arrange
        (NP (NP DT/the NN/flight NN/code)
            (PP IN/of
                (NP (NP DT/the NN/flight)
                    (PP (PP IN/from
                            (NP NP/Denver))
                        (PP TO/to
                            (NP NP/Dallas
                                NP/Worth)))
                (PP IN/in
                    (NP (VP VBG/descending)
                        NN/order))))))
  .))
```

In this parse, we see that the prepositional phrase "`in descending order`" is incorrectly attached to the NP "`the flight`" instead of to the verb "`arrange`". This wrong attachment may be explained by the high relative frequencies of the following subtrees of depth 2 (that appear in structures of sentences like "`Show me the transportation from SFO to downtown San Francisco in August`", where the PP "`in August`" is attached to the NP "`the transportation`", and not to the verb "`show`"):

```
NP NP          NP NP
   PP             PP PP
   PP IN             PP
      NP          PP IN
                     NP
```

Only if the maximum depth was enlarged to 4, subtrees like the following were available, which led to the estimation of the correct tree.

```
VP VB
   NP NP
      PP
   PP IN
      NP VP VBG
         NN
```

It is interesting to note that this subtree occurs only once in the training set. Nevertheless, it induces the correct parsing of the test string. This seems to contradict the fact that probabilities based on sparse data are not reliable. Since many large subtrees are once-occurring events (hapaxes), there seems to be a preference in DOP for an occurrence-based approach if enough context is provided: large subtrees, even if they occur once, tend to contribute to the generation of the correct parse, since they provide much contextual information. Although these subtrees have low probabilities, they tend to induce the correct parse because fewer subtrees are needed to construct a parse.

Additional experiments seemed to confirm this hypothesis. Throwing away all hapaxes yielded an accuracy of 92%, which is a decrease of 4%. Distinguishing between small and large hapaxes, showed that the accuracy was not affected by eliminating the hapaxes of depth 1 (however, as an advantage, the convergence seemed to get slightly faster). Eliminating hapaxes larger than depth 1, decreased the accuracy. The following table shows the parse accuracy after eliminating once-occurring subtrees of different maximum depths.

depth of hapaxes	parse accuracy
1	96%
≤2	95%
≤3	95%
≤4	93%
≤5	92%
≤6	92%
unbounded	92%

Table 3. Parse accuracy after eliminating once-occurring subtrees

Conclusions

We have shown that in DOP and STSG the Viterbi algorithm cannot be used for computing a most probable tree of a string. We developed a modification of Viterbi which allows by means of an iterative Monte Carlo search to *estimate* the most probable tree of a string in polynomial time. Experiments on ATIS showed that only in 68% of the cases, the most probable derivation of a string generates the most probable tree of that string, and that the parse accuracy is dramatically higher than the derivation accuracy. We conjectured that the Monte Carlo algorithm can also be applied to other stochastic grammars for computing the most probable tree of a string. The question as to whether the most probable tree of a string can also be *deterministically* derived in polynomial time is still unsolved.

Acknowledgments

The author is indebted to Remko Scha for valuable comments on an earlier version of this paper, and to Khalil Sima'an for useful discussions.

References

M. van den Berg, R. Bod & R. Scha, 1994. "A Corpus-Based Approach to Semantic Interpretation", *Proceedings Ninth Amsterdam Colloquium*, Amsterdam.

E. Black, R. Garside and G. Leech, 1993. *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach*, Rodopi: Amsterdam-Atlanta.

R. Bod, 1992. "A Computational Model of Language Performance: Data Oriented Parsing", *Proceedings COLING'92*, Nantes.

R. Bod, 1993a. "Using an Annotated Corpus as a Stochastic Grammar", *Proceedings European Chapter fo the ACL'93*, Utrecht.

R. Bod, 1993b. "Monte Carlo Parsing", *Proceedings Third International Workshop on Parsing Technologies*, Tilburg/Durbuy.

R. Bod, 1993c. "Data Oriented Parsing as a General Framework for Stochastic Language Processing", in: K.Sikkel & A. Nijholt (eds.), *Parsing Natural Language*, TWLT6, Twente University.

R. Bod, 1995. *Enriching Linguistics with Statistics: Performance Models of Natural Language*. PhD-thesis, University of Amsterdam (forthcoming).

T. Briscoe and J. Carroll, 1993. "Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars", *Computational Linguistics* 19(1), 25-59.

T. Fujisaki, F. Jelinek, J. Cocke, E. Black and T. Nishino, 1989. "A Probabilistic Method for Sentence Disambiguation", *Proceedings 1st Int. Workshop on Parsing Technologies*, Pittsburgh.

J.M. Hammersley and D.C. Handscomb, 1964. *Monte Carlo Methods*, Chapman and Hall, London.

C.T. Hemphill, J.J. Godfrey and G.R. Doddington, 1990. "The ATIS spoken language systems pilot corpus". *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, Morgan Kaufmann.

F. Jelinek, J.D. Lafferty and R.L. Mercer, 1990. *Basic Methods of Probabilistic Context Free Grammars*, Technical Report IBM RC 16374 (#72684), Yorktown Heights.

M. Kay, 1980. *Algorithmic Schemata and Data Structures in Syntactic Processing*. Report CSL-80-12, Xerox PARC, Palo Alto, Ca.

D. Magerman and C. Weir, 1992. "Efficiency, Robustness and Accuracy in Picky Chart Parsing", *Proceedings ACL'92*, Newark, Delaware.

M. Marcus, B. Santorini and M. Marcinkiewicz, 1993. "Building a Large Annotated Corpus of English: the Penn Treebank", *Computational Linguistics* 19(2).

D. Rumelhart and J. McClelland, 1986. *Parallel Distributed Processing*, The MIT Press, Cambridge, Mass.

R. Scha, 1990. "Language Theory and Language Technology; Competence and Performance" (in Dutch), in Q.A.M. de Kort & G.L.J. Leerdam (eds.), *Computertoepassingen in de Neerlandistiek*, Almere: Landelijke Vereniging van Neerlandici (LVVN-jaarboek).

Y. Schabes, 1992. "Stochastic Lexicalized Tree-Adjoining Grammars", *Proceedings COLING'92*, Nantes.

Y. Schabes and R. Waters, 1993. "Stochastic Lexicalized Context Free Grammars", *Proceedings Third International Workshop on Parsing Technologies*, Tilburg/Durbuy.

K. Sima'an, R. Bod, S. Krauwer and R. Scha, 1994. "Efficient Disambiguation by means of Stochastic Tree Substitution Grammars", *Proceedings International Conference on New Methods in Language Processing*, UMIST, Manchester.

A. Viterbi, 1967. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Trans. Information Theory*, IT-13, 260-269.