Computation of the N Best Parse Trees for Weighted and Stochastic Context-Free Grammars*

Víctor M. Jiménez and Andrés Marzal

Dept. de Informática, Universitat Jaume I, 12071 Castellón, Spain
{vjimenez,amarzal}@inf.uji.es

Abstract. Context-Free Grammars are the object of increasing interest in the pattern recognition research community in an attempt to overcome the limited modeling capabilities of the simpler regular grammars, and have application in a variety of fields such as language modeling, speech recognition, optical character recognition, computational biology, etc. This paper proposes an efficient algorithm to solve one of the problems associated to the use of weighted and stochastic Context-Free Grammars: the problem of computing the N best parse trees of a given string. After the best parse tree has been computed using the CYK algorithm, a large number of alternative parse trees are obtained, in order by weight (or probability), in a small fraction of the time required by the CYK algorithm to find the best parse tree. This is confirmed by experimental results using grammars from two different domains: a chromosome grammar, and a grammar modeling natural language sentences from the Wall Street Journal corpus.

Keywords: Weighted Context-Free Grammars, Stochastic Context-Free Grammars, CYK Algorithm, N Best Parse Trees.

1 Introduction

Syntactic pattern recognition makes use of formal languages theory to describe the underlying structure of pattern classes, in applications where the relationships between primitive elements are important [3,4]. Stochastic grammars are used to model the fact that some structures and patterns are more frequent than others. In this framework, the stochastic Context-Free Grammars (CFGs) are the object of increasing interest in the research community in an attempt to overcome the limited modeling capabilities of the simpler regular grammars, even though this implies using more costly training and parsing algorithms. This is the case for instance of language modeling in speech recognition/understanding [1, 2,6,8,9,11] or RNA modeling in computational biology [12].

* This work has been supported in part by the Spanish *Generalitat Valenciana* under contract GV98-14-134.

F.J. Ferri et al. (Eds.): SSPR&SPR 2000, LNCS 1876, pp. 183–192, 2000.

This paper proposes an efficient algorithm to solve one of the problems associated to the use of weighted CFGs, formally stated in §2: the problem of computing the N best parse trees of a given string, sorted by weight. A particular case of this problem is the computation of the N most likely parse trees when G is a stochastic CFG, that has proved useful for improved training of stochastic CFGs [13,14] and could have as many applications as the use of the N best decodings in current speech recognition systems (re-scoring using more accurate models, improved acoustic training, etc.).

The proposed algorithm works for weighted CFGs in Chomsky Normal Form (CNF), but this does not imply any loss of generality because any weighted CFG can be automatically converted into this form [3]. The best parse tree is computed by means of a well-known version [6,9] of the Cocke-Younger-Kasami (CYK) algorithm [5] (sometimes called Viterbi-style parsing) described in §3. Once the best parse tree has been computed, the N best parse trees can be computed by the algorithm presented in §4. The experimental results, reported in §5, show the practical efficiency of this algorithm.

2 Notation and Problem Formulation

Let $G = (V, \Sigma, S, P, w)$ be a weighted Context-Free Grammar in CNF [3,4], where V is a finite set of nonterminal symbols, Σ is a finite set (disjoint from V) of terminal symbols, $S \in V$ is the start symbol, P is a finite set of productions of the form $A \to \alpha$ with $A \in V$ and $\alpha \in (V \times V) \cup \Sigma$, and $w : P \to \mathbb{R}$ is a function that assigns a weight to each production in P.

Given G and given a string $x = x_1 x_2 \ldots x_{|x|} \in \Sigma^+$, where $|x|$ denotes the length of x, let us define a set $\mathcal{T}$ of binary trees whose nodes are of the form $A_{i:k}$ with $A \in V$ and $1 \le i \le k \le |x|$, and a weighting function $W : \mathcal{T} \to \mathbb{R}$, as follows:

(i) If there is a production $A \to x_i$ in P then the tree $\langle A_{i:i} \rangle$ with the single node $A_{i:i}$ is in $\mathcal{T}$, and has weight $W(\langle A_{i:i} \rangle) = w(A \to x_i)$.
(ii) If there is a tree T_1 with root $B_{i:j}$ in $\mathcal{T}$, a tree T_2 with root $C_{j+1:k}$ in $\mathcal{T}$, and a production $A \to BC$ in P, then the tree $\langle A_{i:k}, T_1, T_2 \rangle$ with root $A_{i:k}$, left subtree T_1, and right subtree T_2 is in $\mathcal{T}$, and has weight $W(\langle A_{i:k}, T_1, T_2 \rangle) = W(T_1) + W(T_2) + w(A \to BC)$.

A tree in $\mathcal{T}$ whose root is $A_{i:k}$ is a *partial parse tree* representing a derivation of the substring $x_i \ldots x_k$ from the nonterminal A. A *parse tree* for x according to G is a binary tree $T \in \mathcal{T}$ whose root is $S_{1:|x|}$. The *best parse tree* is the parse tree of minimum weight. The N *best parse trees* are the N parse trees of minimum total weight. The problem we study in this paper can then be formulated as:

> *Given a weighted Context-Free Grammar in CNF G, given a string $x \in \Sigma^+$ and given a positive integer N, find the N best parse trees for x in order by weight.*

Let us denote $T^n(A_{i:k})$ the n-th best tree among those in $\mathfrak{T}$ that have root $A_{i:k}$, and let $W^n(A_{i:k})$ be its weight. The problem is then finding $T^1(S_{1:|x|})$, $T^2(S_{1:|x|}), \ldots, T^N(S_{1:|x|})$.

A particular case of this problem is the computation of the N most likely parse trees for x when G is a stochastic Context-Free Grammar [3,4]. In this case, a function $p : P \to \mathbb{R}$ assigns a probability to each production, verifying $0 \leq p(A \to \alpha) \leq 1$, for all $A \to \alpha \in P$, and $\sum_{A \to \alpha \in P} p(A \to \alpha) = 1$, for all $A \in V$. If T is a parse tree for x, its probability is the product of the probabilities of all the productions involved in its construction. If we assign to the productions in P a weight $w(A \to \alpha) = -\log(p(A \to \alpha))$ then maximizing products of probabilities becomes minimizing sums of weights, and the N parse trees of maximum probability are the N parse trees of minimum weight.

A problem closely related to the computation of the N best parse trees is the enumeration of parse trees until the best one satisfying some desired restriction is obtained, without fixing *a priori* a value for N. The algorithm that we present in §4 also solves this problem.

3 Computing the Best Parse Tree

The CYK algorithm was initially proposed by Cocke, Younger, and Kasami to solve the problem of, given a Context-Free Grammar G in CNF (not necessarily weighted) and given a string $x \in \Sigma^+$, determine whether there is a parse tree for x according to G or not [5]. The CYK algorithm can be easily modified to compute the best parse tree when G is weighted, on the base of the following recursive equations [6,9].

Recursive Equations For every $A \in V$ and $1 \leq i \leq k \leq |x|$ the best parse tree with root $A_{i:k}$ is

$$T^1(A_{i:k}) = \begin{cases} \underset{T \in \mathfrak{T}^1(A_{i:k})}{\operatorname{argmin}} W(T), & \text{if } k > i, \\ \langle A_{i:i} \rangle, & \text{if } k = i \text{ and } A \to x_i \in P, \end{cases} \tag{1}$$

where, for $k > i$,

$$\mathfrak{T}^1(A_{i:k}) = \{\langle A_{i:k}, T^1(B_{i:j}), T^1(C_{j+1:k}) \rangle : A \to BC \in P, i \leq j < k\} \tag{2}$$

denotes a set of candidates to be the best parse tree with root $A_{i:k}$. According to the definition given in §2, the weight of a tree $T = \langle A_{i:k}, T^1(B_{i:j}), T^1(C_{j+1:k}) \rangle$ in $\mathfrak{T}^1(A_{i:k})$ is $W(T) = W^1(B_{i:j}) + W^1(C_{j+1:k}) + w(A \to BC)$. The weight of the best parse tree with root $A_{i:k}$ is

$$W^1(A_{i:k}) = \begin{cases} \min\limits_{T \in \mathfrak{T}^1(A_{i:k})} W(T), & \text{if } k > i, \\ w(A \to x_i), & \text{if } k = i \text{ and } A \to x_i \in P. \end{cases} \tag{3}$$

If $k > i$ and $\mathfrak{T}^1(A_{i:k})$ is empty, or $k = i$ and $A \to x_i \notin P$, then $T^1(A_{i:k})$ does not exist.

```
1: Algorithm CYK for weighted CFGs
2:   for i := 1 to |x| do
3:     for all A → x_i in P do
4:       T^1(A_{i:i}) := ⟨A_{i:i}⟩
5:   for l := 1 to |x| − 1 do
6:     for i := 1 to |x| − l do
7:       k := i + l
8:       for all A in V do
9:         T^1(A_{i:k}) := arg min_{T∈𝔗^1(A_{i:k})} W(T)
10:  return T^1(S_{1:|x|})
```

Fig. 1. CYK algorithm for weighted CFGs. Given a weighted grammar (V, Σ, P, S, w) in CNF and a string x, the algorithm returns its best parse tree.

CYK Algorithm The problem of computing the best parse tree for x consists then in solving the equations (1–3) to find $T^1(S_{1:|x|})$. The CYK algorithm (Fig. 1) is a dynamic programming algorithm that computes $T^1(A_{i:k})$ iteratively for increasing values of the length $l = k - i$, thus guaranteeing that the right-hand sides of the equations have been previously computed when they are going to be used. Its running time is $O(|x|^3|P|)$, and the required space is $O(|x|^2|V|)$, where $|P|$ is the number of productions and $|V|$ is the number of nonterminals in G.

4 Computing the N Best Parse Trees

Let us now consider how to calculate $T^n(A_{i:k})$ in general for $A \in V$, $1 \le i \le k \le |x|$, and $1 \le n \le N$. In particular, $T^n(S_{1:|x|})$ for $1 \le n \le N$ will be the solution to the problem we are addressing. Let us assume in what follows that $|x| > 1$ (otherwise $T^2(S_{1:|x|})$ does not exist). We will first generalize the recursive equations given in §3 to compute the N best parse trees, and then we will propose an algorithm to solve the generalized equations.

Recursive Equations Let us study which trees should be considered candidates to $T^n(A_{i:k})$. Clearly, $T^n(A_{i:k})$ does not exist for $n > 1$ if $k = i$ (while $T^1(A_{i:k})$ will exist or not depending on whether there is a production $A \to x_i$ in P, as we have seen in §3). Let us then examine the case $k > i$. It should also be clear that in order to calculate $T^n(A_{i:k})$ we do not need to consider the trees of the form $\langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle$ with $p > n$ or $q > n$ (because there are at least n trees among those with $p \le n$ and $q \le n$, with lower or equal weight). Therefore, $T^n(A_{i:k})$ can be chosen as the best tree different from $T^1(A_{i:k}), \ldots, T^{n-1}(A_{i:k})$ in the set

$$\{\langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle : A \to BC \in P, i \le j < k, 1 \le p \le n, 1 \le q \le n\}.$$

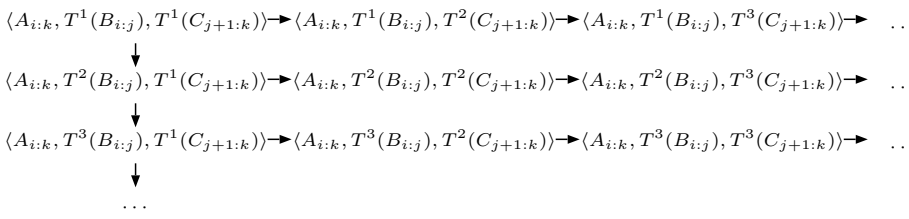

Fig. 2. Schematic representation of the partial order among some candidate trees.

But we can do quite better than computing the N best parse trees for every possible root $A_{i:k}$, because there is a partial order defined among some elements of this set of trees. Based on the relations (schematically represented in Fig. 2)

$$W^p(B_{i:j}) + W^1(C_{j+1:k}) \leq W^{p+1}(B_{i:j}) + W^1(C_{j+1:k}), \tag{4}$$
$$W^p(B_{i:j}) + W^q(C_{j+1:k}) \leq W^p(B_{i:j}) + W^{q+1}(C_{j+1:k}), \tag{5}$$

we can define a smaller set $\mathcal{T}^n(A_{i:k})$ of trees with root $A_{i:k}$ in such a way that we still have the guarantee that $T^n(A_{i:k})$ is the best among them. Let $\mathcal{T}^1(A_{i:k})$ be the set of trees defined in (2). For $n > 1$, let us assume that $T^{n-1}(A_{i:k}) = \langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle$, and if $q = 1$, let

$$\begin{aligned}\mathcal{T}^n(A_{i:k}) = {} & \big(\mathcal{T}^{n-1}(A_{i:k}) - \{T^{n-1}(A_{i:k})\}\big) \cup \{\langle A_{i:k}, T^p(B_{i:j}), T^{q+1}(C_{j+1:k})\rangle\} \\ & \cup \{\langle A_{i:k}, T^{p+1}(B_{i:j}), T^q(C_{j+1:k})\rangle\};\end{aligned} \tag{6}$$

otherwise (if $q > 1$), let

$$\mathcal{T}^n(A_{i:k}) = \big(\mathcal{T}^{n-1}(A_{i:k}) - \{T^{n-1}(A_{i:k})\}\big) \cup \{\langle A_{i:k}, T^p(B_{i:j}), T^{q+1}(C_{j+1:k})\rangle\} \tag{7}$$

assuming always that $\{\langle A_{i:k}, T_1, T_2\rangle\}$ denotes the empty set if T_1 or T_2 does not exist. Then we have

$$T^n(A_{i:k}) = \operatorname*{argmin}_{T \in \mathcal{T}^n(A_{i:k})} W(T), \tag{8}$$
$$W^n(A_{i:k}) = \min_{T \in \mathcal{T}^n(A_{i:k})} W(T), \tag{9}$$

if $\mathcal{T}^n(A_{i:k})$ is not empty (otherwise $T^n(A_{i:k})$ does not exist).

Recursive Enumeration of the N Best Parse Trees The problem of computing the N best parse trees consists then in solving the equations (6–9) to find $T^1(S_{1:|x|}), T^2(S_{1:|x|}), \ldots, T^N(S_{1:|x|})$. The algorithm in Fig. 4 solves them for increasing values of n, after the best parse tree has been computed by the CYK algorithm, recursively starting from the node $S_{1:|x|}$.

The algorithm makes use of the recursive procedure *NextTree*. For $n > 1$ and $k > i$, and once $T^{n-1}(A_{i:k}) = \langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle$ is available,

A1: **Algorithm** Recursive enumeration of the N-best parse trees
A2: Compute $T^1(A_{i:k})$ for all $A \in V$, $1 \leq i \leq k \leq |x|$ using the CYK algorithm
A3: **for** $n := 2$ **to** N **do** $NextTree(T^{n-1}(S_{1:|x|}), n)$
A4: **return** $\{T^1(S_{1:|x|}), T^2(S_{1:|x|}), \ldots, T^N(S_{1:|x|})\}$

B1: **procedure** $NextTree(\langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle, n)$
B2: **if** $n = 2$ **then**
B3: $\mathfrak{T}[A_{i:k}] := \{\langle A_{i:k}, T^1(B_{i:j}), T^1(C_{j+1:k})\rangle : A \to BC \in P, i \leq j < k\} - \{T^1(A_{i:k})\}$
B4: **if** $q = 1$, $j > i$, and $T^{p+1}(B_{i:j})$ has not been computed **then**
B5: $NextTree(T^p(B_{i:j}), p+1)$
B6: **if** $q = 1$ and $T^{p+1}(B_{i:j})$ exists **then**
B7: $\mathfrak{T}[A_{i:k}] := \mathfrak{T}[A_{i:k}] \cup \{\langle A_{i:k}, T^{p+1}(B_{i:j}), T^q(C_{j+1:k})\rangle\}$
B8: **if** $k > j+1$ and $T^{q+1}(C_{j+1:k})$ has not been computed **then**
B9: $NextTree(T^q(C_{j+1:k}), q+1)$
B10: **if** $T^{q+1}(C_{j+1:k})$ exists **then**
B11: $\mathfrak{T}[A_{i:k}] := \mathfrak{T}[A_{i:k}] \cup \{\langle A_{i:k}, T^p(B_{i:j}), T^{q+1}(C_{j+1:k})\rangle\}$
B12: **if** $\mathfrak{T}[A_{i:k}] \neq \emptyset$ **then**
B13: $T^n(A_{i:k}) := \arg\min_{T \in \mathfrak{T}[A_{i:k}]} W(T)$
B14: $\mathfrak{T}[A_{i:k}] := \mathfrak{T}[A_{i:k}] - \{T^n(A_{i:k})\}$
B15: **else**
B16: $T^n(A_{i:k})$ does not exist

Fig. 3. Algorithm to compute the N best parse trees.

$NextTree(\langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle, n)$ computes $T^n(A_{i:k})$ according to equation (8). In first place, it builds $\mathfrak{T}^n(A_{i:k})$ from $\mathfrak{T}^{n-1}(A_{i:k})$ according to equations (6–7). This may require inserting in this set at most two new candidate trees: $\langle A_{i:k}, T^{p+1}(B_{i:j}), T^q(C_{j+1:k})\rangle$ and $\langle A_{i:k}, T^p(B_{i:j}), T^{q+1}(C_{j+1:k})\rangle$. If $T^{p+1}(B_{i:j})$ (or $T^{q+1}(C_{j+1:k})$) is required and has not been computed before, it is computed by calling $NextTree(T^p(B_{i:j}), p+1)$ (or $NextTree(T^q(C_{j+1:k}), q+1)$).

Both $\mathfrak{T}^{n-1}(A_{i:k})$ and $\mathfrak{T}^n(A_{i:k})$ can be implemented by the same structure $\mathfrak{T}[A_{i:k}]$, because once $T^{n-1}(A_{i:k})$ has been calculated, $\mathfrak{T}^{n-1}(A_{i:k})$ is no longer necessary. On the other hand, the set $\mathfrak{T}[A_{i:k}]$ is initialized only when the second best parse tree with root $A_{i:k}$ is required, because there could be nodes $A_{i:k}$ for which it is not necessary to compute alternative trees.

Finiteness of the recursion is guaranteed by the fact that the first arguments of the recursive calls are trees with root $A_{i:k}$ for decreasing values of $k - i$, and are only performed if $k - i > 0$, so that the number of recursive calls produced by $NextTree(T^{n-1}(S_{1:|x|}), n)$ to compute $T^n(S_{1:|x|})$ is at most $|x|$.

Data Structures and Implementation Issues The trees $T^1(A_{i:k})$, $T^2(A_{i:k})$, $T^3(A_{i:k})$, ... can be stored ordered by weight in a linked list associated to node $A_{i:k}$. Every tree of the form $\langle A_{i:k}, T^p(B_{i:j}), T^q(C_{j+1:k})\rangle$ can be efficiently represented in memory by just three values: its weight, a pointer to its left subtree $T^p(B_{i:j})$, and a pointer to its right subtree $T^q(C_{j+1:k})$. In this way, $T^n(A_{i:k})$ can be inserted, in step B13, in constant time following $T^{n-1}(A_{i:k})$ in the list of trees

associated with node $A_{i:k}$, and steps B4, B6, B8, and B10 take constant time to check whether $T^{p+1}(B_{i:j})$ or $T^{q+1}(C_{j+1:k})$ are available.

The only operations performed by the algorithm with the sets of candidates $\mathcal{T}[A_{i:k}]$ are those supported by priority queues: insertion of new elements and selection/deletion of the best element. Several data structures allow to perform these operations in time logarithmic with respect to the number of elements in the priority queue [7]. The results reported in §5 correspond to an implementation using *leftist trees* [7].

An additional improvement in the algorithm is possible: of all the candidate trees with the same value of j, only the best one needs to be inserted in $\mathcal{T}[A_{i:k}]$ when it is initialized in step B3. The rest of candidates with that value of j only need to be inserted if the best one is extracted (after step B14).

Computational Complexity The CYK algorithm runs in time $O(|x|^3|P|)$. The number of different sets $\mathcal{T}[A_{i:k}]$ is $O(|x|^2|V|)$ and, in the worst case, all of them are initialized by step B3 (in different calls to *NextTree*) in total time $O(|x|^3|P|)$, because each initialization can be performed in linear time with respect to the size of the set.

The computation of the N best parse trees requires at most $N|x|$ calls to *NextTree*. Each call may require to insert at most two new elements in a set of candidates (steps B7 and B11), and to select and delete the best candidate (steps B13 and B14) from it. Since no more than N trees with root $A_{i:k}$ may need to be computed, the size of $\mathcal{T}[A_{i:k}]$ is bounded by its initial size plus N. Thus, the total time required by the whole algorithm to compute the N best parse trees is $O(|x|^3|P| + N|x| \log(|x| \frac{|P|}{|V|} + N))$.

On the other hand, the space complexity of the algorithm is $O(|x|^3|P|+N|x|)$.

This computational complexity analysis is based on worst case assumptions that could be too pessimistic. In practice, it can be expected that even for large values of N, not all the sets of candidates are initialized and the number of recursive calls can be much lower than $N|x|$.

5 Experimental Results

In order to assess the behavior of the algorithm in practice, we have performed experiments with strings and grammars corresponding to two different domains. All the experiments have been run on a 400 MHz Pentium-II computer running under Linux 2.2. The algorithm has been implemented in C and compiled with `gcc` 2.91 using the optimization level '`-O2`'.

Chromosome Grammar Strings of different lengths have been randomly generated using the chromosome grammar described in [4, §2.3.1] and assuming that all the productions with the same left-hand symbol have the same probability. The grammar in CNF has 9 nonterminals, 5 terminals, and 20 productions. Fig. 4 shows the observed dependency of the running time of the algorithm in

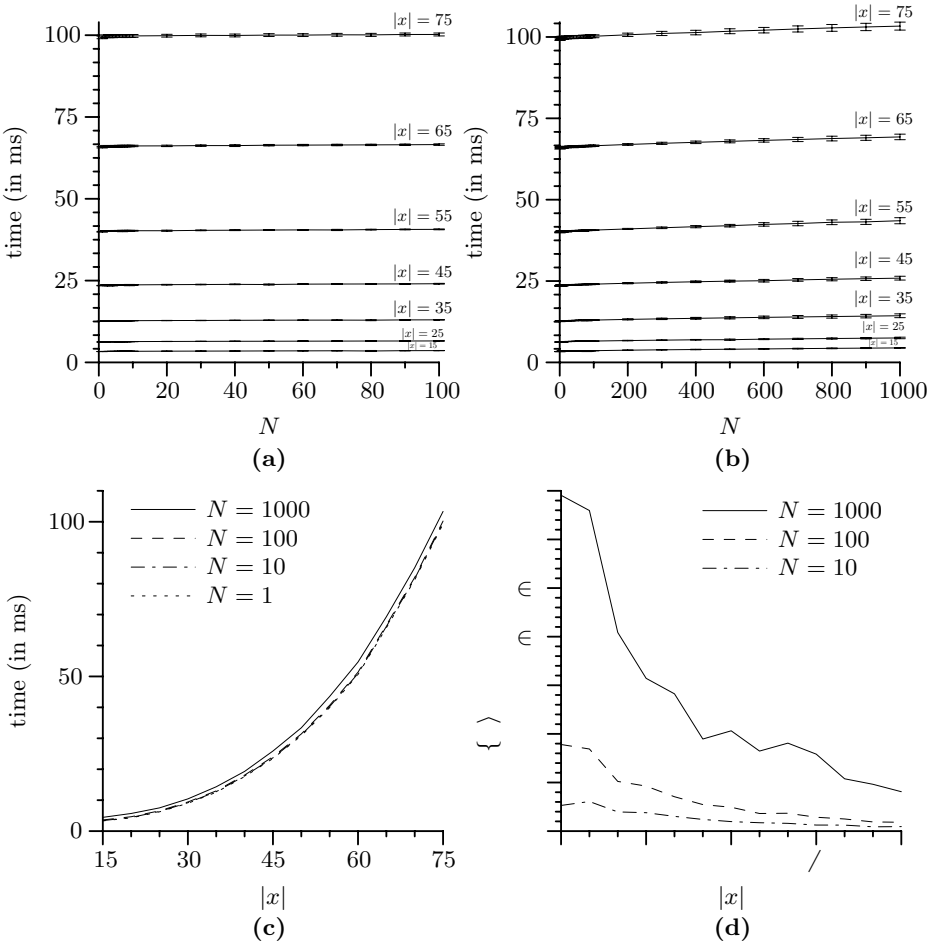

Fig. 4. Dependency of running time with N and $|x|$ using the chromosome grammar.

milliseconds, averaged for 20 strings of each length, as a function of N (for different string lengths) and as a function of the string length (for different values of N).

Figs. 4a and 4b show the average time ($\pm$ two times the standard deviation) required to compute up to 100 and up to 1000 best parse trees, respectively. Fig. 4c depicts the dependency of time with the string length for different values of N. Time for $N = 1$ corresponds to the CYK algorithm. It can be clearly observed that once the best parse tree has been found by the CYK algorithm, the rest of the N best parse trees are computed very efficiently, in just a small fraction of the total running time. This is made more explicit in Fig. 4d, where the time to compute from the second to the N-th best parse tree is shown as a

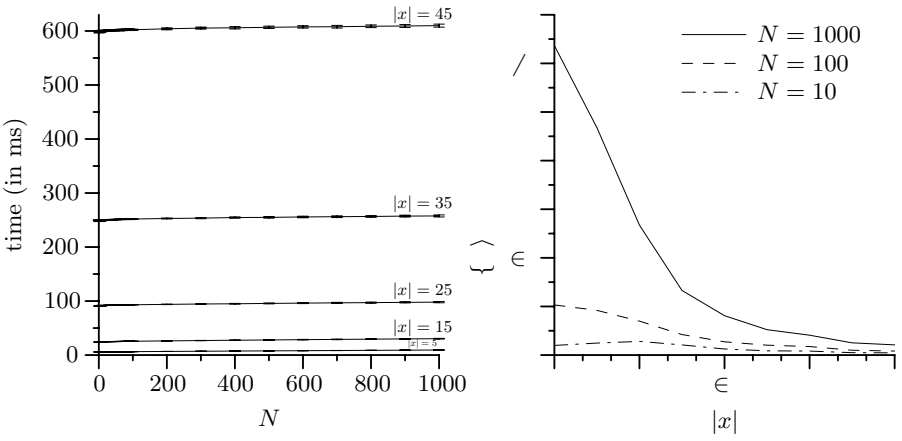

Fig. 5. Dependency of running time with N and $|x|$ using the Wall Street Journal grammar.

percentage of the time required by the CYK algorithm to find the optimal parse tree.

Wall Street Journal The algorithm has also been tested using sentences from the Wall Street Journal (WSJ) corpus, as annotated in the Penn Treebank [10]. A grammar with 42 terminals (part of speech tags), 14 nonterminals, and 518 productions has been obtained using the Inside-Outside algorithm with the sentences shorter than 16 words in sections 00–19 of this corpus [14]. Sentences from sections 20–24 have been used to measure the running time of the algorithm to compute the N best parse trees with this grammar. The results, shown in Fig. 5 (also averaged for 20 sentences of each length), are similar to those obtained with the chromosome grammar.

6 Conclusions

In this paper, a new algorithm to compute the N best parse trees for weighted (or stochastic) Context-Free Grammars has been presented. The experimental results with two different pattern recognition tasks have shown the practical efficiency of this algorithm, which can be used to compute a large number of parse trees, in order by weight, in a small fraction of the time required by the CYK algorithm to compute the best one. Since there is no need to fix *a priori* the value of N, the algorithm can be used to enumerate parse trees until the best one satisfying some desired constraint is obtained.

Acknowledgments

The authors wish to thank to J. A. Sánchez and J. M. Benedí for providing the Wall Street Journal grammar used in the experiments.

References

1. R. Cole, editor. *Survey of the State of the Art in Human Language Technology.* Studies in Natural Language Processing. Cambridge University Press, 1998.
2. A. Corazza, R. De Mori, R. Gretter, and G. Satta. Optimal probabilistic evaluation functions for search controlled by stochastic context-free grammars. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(10):1018–1027, 1994.
3. K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.
4. R. C. Gonzalez and M. G. Thomason. *Syntactic Pattern Recognition, An Introduction.* Addison-Wesley, Reading, MA, 1978.
5. M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.
6. F. Jelinek, J. D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context free grammars. In P. Laface and R. De Mori, editors, *Speech Recognition and Understanding*, volume F75 of *NATO ASI*, pages 345–360. Springer-Verlag, 1992.
7. D. E. Knuth. *The Art of Computer Programming*, volume 3 / Sorting and Searching. Addison-Wesley, Reading, MA, 1973.
8. K. Lari and S. J. Young. Applications of stochastic context-free grammars using the Inside-Outside algorithm. *Computer, Speech and Language*, 5:237–257, 1991.
9. S. E. Levinson. Structural methods in automatic speech recognition. *Proceedings of the IEEE*, 73(11):1625–1650, 1985.
10. M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.
11. H. Ney. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Trans. on Signal Processing*, 39(2):336–340, 1991.
12. Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjolander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, 1994.
13. J. A. Sánchez and J. M. Benedí. Estimation of the probability distributions of stochastic context free grammars from the K-best derivations. In *Proc. Int. Conf. on Spoken Language Processing (ICSLP)*, pages 2495–2498, 1998.
14. J. A. Sánchez and J. M. Benedí. Learning of stochastic context-free grammars by means of estimation algorithms. In *Proc. of the European Conf. on Speech Communication and Technology (EUROSPEECH)*, 1999.