Fast Stochastic Context-Free Parsing: A Stochastic Version of the Valiant Algorithm

José-Miguel Benedí and Joan-Andreu Sánchez

Universidad Politécnica de Valencia
46022 Valencia (Spain)
{jbenedi,jandreu}@dsic.upv.es

Abstract. In this work, we present a fast stochastic context-free parsing algorithm that is based on a stochastic version of the Valiant algorithm. First, the problem of computing the string probability is reduced to a transitive closure problem. Then, the closure problem is reduced to a matrix multiplication problem of matrices of a special type. Afterwards, some fast algorithm can be used to solve the matrix multiplication problem. Preliminary experiments show that, in practice, an important time savings can be obtained.

1 Introduction

Stochastic Context-Free Grammars (SCFGs) are an important specification formalism that are frequently used in Syntactic Pattern Recognition. SCFGs have been widely used to characterize the probabilistic modeling of language in Computational Linguistics [1,2], Speech Recognition and Understanding [3], and Biological Sequence Analysis [4]. The main advantages of this formalism are: the capability to model the long-term dependencies that can be established between the different parts of a sentence, and the possibility of incorporating the stochastic information to allow for an adequate modeling of the variability phenomena that are always present in complex problems. However, a notable obstacle to using these models is the time complexity of both the stochastic parsing algorithms and the algorithms that are used for the probabilistic estimation of the models from a training corpus.

Most of the stochastic parsing algorithms are based either on the Earley algorithm, for SCFGs in general format [1], or on the Cocke-Younger-Kasami (CYK) algorithm, for SCFGs in Chomsky Normal Form (CNF) [3]. Both algorithms are based on a Dynamic Programming scheme and have a time complexity $O(gn^3)$, where n is the length of the string and g is the size of the grammar. One of the well-known algorithms for computing the probability of a string given a SCFG in CNF is the inside algorithm [5].

There are theoretical works that attempt to improve the time complexity of the parsing algorithms for context-free grammars. In [6], a version of the CYK algorithm was proposed whose time complexity is $O(M(n))$, where $M(n)$ is the time complexity of the product of two boolean matrices of dimension n. The close relation between context-free parsing and boolean matrix multiplication is demonstrated in [7]. A version of the Valiant algorithm that is based on the computation of shortest paths is presented in [8].

There are a lot of interesting works in the literature for matrix multiplication [9,10]. The classical method is the well-known Strassen algorithm [9]. This algorithm has a time complexity $O(n^{2.8})$. Another method is the Coppersmith & Winograd algorithm, which has a time complexity $O(n^{2.38})$ [10]. Although this algorithm is asymptotically faster, it involves such huge hidden constants that it is not practical.

In this work, we present a fast stochastic context-free parsing algorithm that is based on a stochastic version of the Valiant algorithm. The parsing problem is reduced to a multiplication problem of matrices of a special type. Fast algorithms can be used for this matrix multiplication problem. However, the constant that is associated to fast matrix multiplication algorithms is large. Moreover, in the case of parsing algorithms that are based on matrix multiplication, the size of the grammar is another factor that also affects the time complexity of the algorithm. In real tasks, the grammar can have thousands of rules.

In the following section, some preliminary concepts are introduced. In Section 3, we will reduce the computation of the probability of a string to the transitive closure of matrices. In Section 4, we will reduce the transitive closure to a matrix product. In Section 5, preliminary experiments are reported.

2 Preliminaries

First, we introduce the notation for SCFGs. Then, we present the preliminary definitions that constitute the formal framework used in this work.

A *Context-Free Grammar* (CFG) G is a four-tuple (N, Σ, S, P), where N is a finite set of non-terminals, Σ is a finite set of terminals ($N \cap \Sigma = \emptyset$), $S \in N$ is the initial non-terminal, and P is a finite set of rules: $A \rightarrow \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^+$ (without loss of generality, we only consider grammars with no empty rules). A CFG in Chomsky Normal Form (CNF) is a CFG in which the rules are of the form $A \rightarrow BC$ or $A \rightarrow a$ ($A, B, C \in N$ and $a \in \Sigma$). A *left-derivation* of $x \in \Sigma^+$ in G is a sequence of rules $d_x = (q_1, q_2, \ldots, q_m)$, $m \geq 1$, such that: $(S \overset{q_1}{\Rightarrow} \alpha_1 \overset{q_2}{\Rightarrow} \alpha_2 \overset{q_3}{\Rightarrow} \ldots \overset{q_m}{\Rightarrow} x)$, where $\alpha_i \in (N \cup \Sigma)^+$, $1 \leq i \leq m-1$ and q_i rewrites the left-most non-terminal of α_{i-1}. The *language generated* by G is defined as $L(G) = \{x \in \Sigma^+ \mid S \overset{*}{\Rightarrow} x\}$.

A *Stochastic Context-Free Grammar* (SCFG) is defined as a pair (G, p), where G is a CFG and $p : P \rightarrow]0, 1]$ is a probability function of rule application such that $\forall A \in N : \sum_{i=1}^{n_A} p(A \rightarrow \alpha_i) = 1$, where n_A is the number of rules associated to A.

Let G_s be a SCFG. Then, we define the *probability of the derivation* d_x of the string x, $\Pr_{G_s}(x, d_x)$ as the product of the probability application function of all the rules used in the derivation d_x. Given that for some $x \in L(G)$ there can be more than one derivation, we also define the *probability* of the string x as: $\Pr_{G_s}(x) = \sum_{\forall d_x} \Pr_{G_s}(x, d_x)$.

The probability of generating a string, given G_s, can be computed from the well-known inside algorithm [5]. The inside algorithm is the stochastic version of the classical parsing algorithm of Cocke-Kasami-Younger.

In the following sections, we will present the inside algorithm in terms of the matrix product. To do this, the following concepts must be introduced.

Definition 1. *Given a SCFG G_s, we define a* stochastic non-terminal symbol vector *(SNTV) related to this SCFG as a vector C with $|N|$ components, where each* *component is associated with a non-terminal symbol such that* $0 \leq C[A] \leq 1, \forall A \in N$. *A special SNTV* $\overline{0}$ *can be defined:* $\overline{0}[A] = 0, \forall A \in N$.

We define a binary operation $\oplus$ on arbitrary SNTVs, a and b as follows:

$$(a \oplus b)[A] = a[A] + b[A] \qquad \forall A \in N$$

It can be seen that this operation is associative, commutative, and $a + \overline{0} = \overline{0} + a = a$. Note that this operation is not always an inner operation. However, we will see later that depending in the context of use, it can be an inner operation.

We also define another binary operation $\odot$ on arbitrary SNTVs, a and b as follows:

$$(a \odot b)[A] = \sum_{B,C \in N} \Pr(A \to BC) a[B] b[C] \qquad \forall A \in N \tag{1}$$

This is an inner operation because $0 \leq a[B], b[C] \leq 1, \forall B, C \in N$, and $\sum_{B,C \in N} \Pr(A \to BC) = 1$. Note that $a \odot \overline{0} = \overline{0} \odot a = \overline{0}$. This operation is neither associative nor commutative. However, operation $\odot$ distributes over operation $\oplus$.

Definition 2. *A SNTV* matrix *is a square matrix in which each element is a SNTV.*

Given two $n \times n$ square SNTV matrices, U and V, their sum is defined as:

$$(u + v)_{i,j} = u_{i,j} \oplus v_{i,j} \qquad 1 \leq i, j \leq n$$

and the product is defined as:

$$(u\ v)_{i,j} = \sum_{k=1}^{n} u_{i,k} \odot v_{k,j} \qquad 1 \leq i, j \leq n$$

where the sum is defined in terms of operation $\oplus$. We will see later that the conditions in which these operations are used guarantee that both of them are inner operations. The sum of SNTV matrices is associative and commutative and has unit element. The product is neither associative nor commutative. However, the product distributes over the sum.

From the previous definitions and following a notation very close to [11], the inside algorithm can expressed in terms of a $(n + 1) \times (n + 1)$ SNTV matrix t, such that:

$$t_{i,j}[A] = \Pr_{G_s}(A \overset{*}{\Rightarrow} x_{i+1} \ldots x_j)$$

Initializing $t_{i,j} = \overline{0}, \quad 0 \leq i, j \leq n$, and $\forall A \in N$:

$$t_{i,i+1}[A] = \Pr(A \to x_{i+1}) \qquad 0 \leq i < n,$$

$$t_{i,i+j}[A] = \sum_{B,C \in N} \Pr(A \to BC) \sum_{k=1}^{j-1} t_{i,i+k}[B] t_{i+k,i+j}[C] \quad 2 \leq j \leq n, 0 \leq i \leq n - j$$

and therefore,

$$t_{i,i+j} = \sum_{k=1}^{j-1} t_{i,i+k} \odot t_{i+k,i+j} \tag{2}$$

In this way, $\Pr_{G_s}(x) = t_{0,n}[S]$.

Note that in expression (2), the sum is defined in terms of $\oplus$. It can be observed that by definition, $t_{i,j}[A] \leq 1$, and because the combinations of probabilities are always bounded by one then the $\oplus$ operation is only used as an inner operation.

3 Reducing the Inside Probability to the Transitive Closure

Following a presentation that is similar to the one in [12], we now explain how the SNTV matrix t can be computed by means of square SNTV matrix operations.

Definition 3. *Given a SCFG G_s and a string $x_1 \cdots x_n \in \Sigma^*$, we define E as a square SNTV matrix with dimension $n+1$ such that $\forall A \in N$:*

$$\begin{aligned} e_{i,i+1}[A] &= \Pr(A \rightarrow x_{i+1}) && 0 \leq i \leq n-1, \\ e_{i,j} &= \overline{0} && \text{otherwise.} \end{aligned}$$

From this definition, E^2 can be computed in the following way:

$$e^{(2)}_{i,j} = \sum_{k=0}^{n} e_{i,k} \odot e_{k,j} \quad , \quad 0 \leq i,j \leq n-2$$

Given that $e_{i,k} = \overline{0}$ if $k \neq i+1$ and, consequently, $e_{k,j} = \overline{0}$ if $j \neq i+2$, then the only element that is different from zero in row i is:

$$e^{(2)}_{i,j} = e_{i,i+1} \odot e_{i+1,i+2}$$

Therefore, $e^{(2)}_{i,i+2} \neq \overline{0}$, $0 \leq i < n-2$, and $e^{(2)}_{i,j} = \overline{0}$, otherwise.

For E^3: $E^3 = E^2E + EE^2$, given that the product between square SNTV matrices is not commutative. Thus:

$$e^{(3)}_{i,j} = \sum_{k=0}^{n} e^{(2)}_{i,k} \odot e_{k,j} + \sum_{k=0}^{n} e_{i,k} \odot e^{(2)}_{k,j} = e^{(2)}_{i,i+2} \odot e_{i+2,i+3} + e_{i,i+1} \odot e^{(2)}_{i+1,i+3}$$

Therefore, $e^{(3)}_{i,i+3} \neq \overline{0}$, $0 \leq i < n-3$, and $e^{(3)}_{i,j} = \overline{0}$, otherwise.

This result can be easily extended for l, $1 \leq l \leq n$:

$$e^{(l)}_{i,i+l} \neq \overline{0} \qquad\qquad 0 \leq i \leq n-l \ .$$

Lemma 1. *The positive closure of a square SNTV matrix E is defined as: $E^+ = \sum_{i=1}^{n} E^i$.*

From the result of the previous definition, it can be seen that, for $l = n$, the only no null value in E^n is $e^{(n)}_{0,0+n}$ and, for $l > n$, all values of E^l are null.

Theorem 1. *Let G_s be a SCFG and let $x \in \Sigma^+$ be a string. Let E^+ be the positive closure of matrix E which was defined previously. Then $E^+ = E^n = t$, and $e^{(l)}_{i,i+l} = t_{i,l}$, $1 \leq l \leq n$, $0 \leq i \leq n-l$.*

The demonstration is by induction on l. For $l = 1$, by definition of E and t:

$$e_{i,i+1}[A] = \Pr(A \rightarrow x_{i+1}) = t_{i,i+1}[A] \qquad 0 \leq i \leq n-1, \forall A \in N.$$

Suppose that for j, $1 < j < l$, then: $e^{(j)}_{i,i+j}[A] = t_{i,i+j}[A]$ By expression (2) in the definition of t:

$$t_{i,i+j}[A] = \sum_{B,C \in N} \Pr(A \to BC) \sum_{k=1}^{j-1} t_{i,i+k}[B] t_{i+k,i+j}[C] = \sum_{k=1}^{j-1} t_{i,i+k} \odot t_{i+k,i+j}$$

For l:

$$e^{(l)}_{i,i+l} = \sum_{j=1}^{l-1} e^{(j)}_{i,i+j} e^{(l-j)}_{i+j,i+l} = \sum_{j=1}^{l-1} t_{i,i+j} t_{i+j,i+l} = t_{i,i+l}$$

$\square$

Corollary 1. *Let G_s be a SCFG and let $x \in \Sigma^+$ be a string. Then* $\Pr_{G_s}(x) = t_{0,n}[S] = e^{(n)}_{0,0+n}$.

4 Reducing the Transitive Closure to a Matrix Multiplication

In this section, we describe how to compute the positive closure of a square SNTV matrix E by reducing the computation to the multiplication of a square SNTV matrix. Following Valiant's work [6] and taking into account the properties of the operation for square SNTV matrices defined in Section 2, the following lemma is proposed:

Lemma 2. *Let E be a $(n \times n)$ SNTV matrix, and suppose that, for some $r > n/2$, the transitive closure of the partitions $[1 \leq i, j \leq r]$ and $[n - r < i, j \leq n]$ is known. Then the closure of E can be computed by*

(i) performing a single matrix multiplication, and
(ii) finding the closure of a $2(n - r) \times 2(n - r)$ upper triangular matrix for which the closure of the partitions $[1 \leq i, j \leq n - r]$ and $[n - r < i, j \leq 2(n - r)]$ is known.

The demonstration of the lemma is similar to the one that is presented in [6]. The lemma shows that the only part of E^+ that needs to be calculated is the top-right partition. Figure 1 shows in bold the parts of the upper triangular matrix E of the lemma that are known. The parts of E in the squares are multiplied in step (i) of the lemma, and an $(n - r) \times (n - r)$ matrix C is obtained. This gives a partial computation of the closure of $e_{i,j}$, $1 \leq i \leq n - r$, $r < j \leq n$. The computation is completed in step (ii), which is explained below.

$$\left(\begin{array}{ccc|ccc|ccc}
\mathbf{e_{1,1}} & \cdots & \mathbf{e_{1,n-r}} & \mathbf{e_{1,n-r+1}} & \cdots & \mathbf{e_{1,r}} & e_{1,r+1} & \cdots & e_{1,n} \\
 & \ddots & & & & & & & \\
 & & \mathbf{e_{n-r,n-r}} & \mathbf{e_{n-r,n-r+1}} & \cdots & \mathbf{e_{n-r,r}} & e_{n-r,r+1} & \cdots & e_{n-r,n} \\
\hline
 & & & \mathbf{e_{n-r+1,n-r+1}} & \cdots & \mathbf{e_{n-r+1,r}} & \mathbf{e_{n-r+1,r+1}} & \cdots & \mathbf{e_{n-r+1,n}} \\
 & & & & \ddots & & & & \\
 & & & & & \mathbf{e_{r,r}} & \mathbf{e_{r,r+1}} & \cdots & \mathbf{e_{r,n}} \\
\hline
 & & & & & & \mathbf{e_{r+1,r+1}} & \cdots & \mathbf{e_{r+1,n}} \\
 & & & & & & & \ddots & \\
 & & & & & & & & \mathbf{e_{n,n}}
\end{array}\right)$$

Fig. 1. Upper triangular matrix E

Let D be the $2(n-r) \times 2(n-r)$ matrix obtained from E by replacing its top-right part by C, and eliminating all the ith rows and columns for $n - r < i \leq r$. In the step (ii), the closure of D is computed and the top-right partition of E^+ is obtained.

Theorem 2. *Let $M(n)$ be the time complexity of a matrix multiplication algorithm that is well behaved in the sense that there is a constant $\gamma \geq 2$ such that for all m, $2^\gamma \cdot M(2^m) \leq M(2^{m+1})$. Then, there is a transitive closure algorithm of complexity $T(n)$ such that $T(n) \leq M(n) \cdot f(n)$, where $f(n)$ is a constant function if $\gamma > 2$, and $f(n) = O(\log n)$ in any case.*

The demonstration of this theorem can be seen in [6].

Note that the elements of the matrices are SNTVs, and the matrix product is stated in terms of operation (1). Therefore, the size of the grammar affects the time complexity of the most inner operation of the matrix product. A fast matrix multiplication algorithm that decreased the number of operations could lead to important improvement in real tasks. In the following section, we explore this idea.

5 Experiments

In this section, we describe the experiments that were carried out to test the stochastic parsing algorithm described in the previous section. Two different experiments were carried out. In the first experiment, we tested the parsing algorithm with synthetic tasks. In the second experiment, we tested the parsing algorithm with a SCFG obtained from the UPenn treebank corpus.

For these experiments, we implemented both the classical version of the inside algorithm and the new version of the parsing algorithm that is based on matrix multiplication. Note that both algorithms produce exactly the same results, that is, the probability of the input string (see Corollary 1). We chose a simple fast matrix multiplication algorithm, since the emphasis of these experiment was on the reduction of the parsing problem to the matrix multiplication problem. In these experiments, we used the classical version of the Strassen algorithm [9]. In all the experiments, we measured the time complexity for both algorithms. The memory consumption of the matrix multiplication based parsing algorithm increased a logarithmic factor, depending on the size of the input string, with respect to the classical algorithm.

All the software was implemented in C language and the `gcc` compiler (version 3.3.5) was used. All experiments were carried out on a personal computer with an Intel Pentium 4 processor of 3.0GHz, with 1.0 GB of RAM and with a Linux 2.6.12 operating system.

5.1 Experiments with Synthetic Tasks

In this first experiment, we considered a very simple task. We parsed strings of increasing length with only one terminal symbol, given that the number of terminal symbols is not relevant for the parsing algorithm. We parsed the strings with grammars of increasing size. We used ergodic grammars with different numbers rules. Note that since the probabilities of the rules are not relevant for the parsing algorithm, they were randomly generated. In all the experiments, we measured the time in the number of seconds that were necessary to parse each string. Figure 2 shows the ratio between the time needed by the classical algorithm and the fast algorithm for strings of different length with grammars of increasing size. Note that in all cases the ratio was between two and three. The ratio decreased for small strings and large grammars due to the amount of overhead. However, the ratio kept over two for large grammars and large strings.

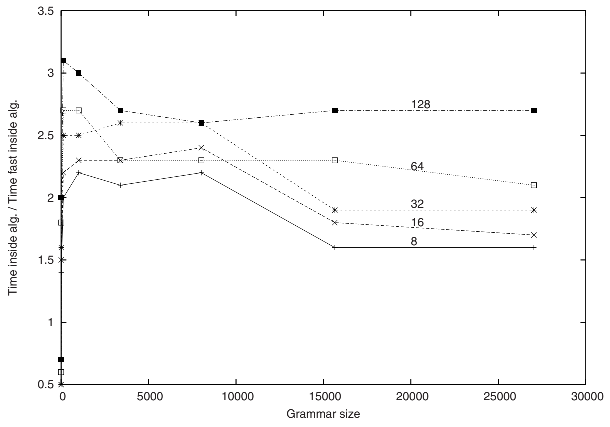

Fig. 2. Ratio between the time needed by the classical algorithm and the fast algorithm for strings of different length (8, 16, 32, 64, 128) with grammars of increasing size

5.2 Experiments with a Real Task

In this section, we describe the experiment that was carried out with a real task. The corpus used in the experiment was the UPenn Treebank corpus [13]. This corpus was automatically labeled, analyzed, and manually checked as described in [13]. There are two kinds of labeling: a POStag labeling and a syntactic labeling that is represented by brackets. The POStag vocabulary is composed of 45 labels. For this experiment, we used only the tagged part of the corpus.

We constructed an initial ergodic SCFG with the maximum number of rules that can be composed with 45 terminal symbols (the POStag set) and 35 non-terminal symbols. Therefore, the initial grammar had 44,450 rules. We trained this ergodic SCFG with the *inside-outside* algorithm [5]. After the estimation process, most of the rules disappeared due to underflow issues, and the final estimated grammar had 35 non-terminal symbols, 45 terminal symbols, and 1,741 rules.

Then, we parsed a string of the corpus of size 2^n for $n \geq 2$ with this SCFG. The results obtained in this experiment are shown in Figure 3.

Note that this figure corresponds to one point in the $x-$axis of the Figure 2. This figure shows that in this experiment the fast parsing algorithm parsed the strings in less time than the classical parsing algorithms. For strings with length near the average length (24 words in this corpus) the fast algorithm was twice quicker than the classical algorithm. This result shows that the fast parsing algorithm can be used in real problems.

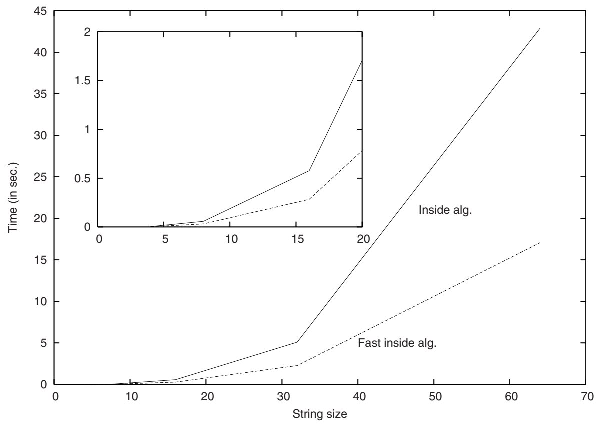

Fig. 3. Results obtained from the SCFG estimated with the UPenn Treebank corpus. The SCFG had 1,741 rules.

6 Conclusions

In this work, a stochastic version of the Valiant parsing algorithm has been presented. It has been shown that, in real tasks, the new algorithm notably improves the parsing time with respect to the classical inside algorithm. For future work, we intend to study the improvement of the time complexity of the estimation algorithms for SCFGs. In addition, we plan to use other fast matrix multiplication algorithms.

Acknowledgment

This work has been partially supported by the EC (FEDER) and Spanish MEC under contract (TIN2006-15694-C02-01) and by the UPV with the ILETA project.

References

1. Stolcke, A.: An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics 21(2), 165–200 (1995)
2. Benedí, J., Sánchez, J.: Estimation of stochastic context-free grammars and their use as language models. Computer Speech and Language 19(3), 249–274 (2005)
3. Ney, H.: Stochastic grammars and pattern recognition. In: Laface, P., Mori, R.D. (eds.) Speech Recognition and Understanding. Recent Advances, pp. 319–344. Springer, Heidelberg (1992)
4. Sakakibara, Y., Brown, M., Hughey, R., Mian, I., Sjölander, K., Underwood, R., Haussle, D.: The application of stochastic context-free grammars to folding, aligning and modeling homologous rna. Computer and Information Science UCSC-CRL-94-14, Univ. of California, Santa Cruz, Ca (1993)
5. Baker, J.: Trainable grammars for speech recognition. In: Klatt, Wolf. (eds.) Speech Communications for the 97th Meeting of the Acoustical Society of America, Acoustical Society of America (1979) 31–35

6. Valiant, L.: General context-free recognition in less than cubic time. Journal of computer and system sciences 10, 308–315 (1975)
7. Lee, L.: Fast context-free grammar parsing requires fast boolean matrix multiplication. Journal of the ACM 49(1), 1–15 (2002)
8. Rytter, W.: Context-free recognition via shortest paths computation: a version of valiant's algorithm. Theoretical Computer Science 143, 343–352 (1995)
9. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik 13, 354–356 (1969)
10. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. 9(3), 251–280 (1990)
11. Goodman, J.: Semiring parsing. Computational Linguistics 25(4), 573–605 (1999)
12. Harrison, M.: Introduction to Formal Language Theory. Addison-Wesley, London, UK (1978)
13. Marcus, M., Santorini, B., Marcinkiewicz, M.: Building a large annotated corpus of english: the penn treebank. Computational Linguistics 19(2), 313–330 (1993)