The Cocke-Younger-Kasami Algorithm for Cyclic Strings *

Jose Oncina
Depto. de Lenguajes y Sistemas Informáticos
Universidad de Alicante
E-03080 Alicante (Spain)
e-mail: oncina@dlsi.ua.es

Abstract

The chain-code representation of contours is widely used in syntactic shape recognition. However the ignorance of a starting point makes the time complexity of traditional parsers increase in one order. This paper describes a technique for adapting any Cocke-Younger-Kasami context-free parser in order to use it with cyclic strings. The parsers obtained with this technique have the same time and space complexity as the original one (usually $\mathcal{O}(n^3)$).

1. Introduction

The recognition of two-dimensional shapes has been intensively studied in pattern recognition and computer vision in the past and many different approaches have been proposed. One technique consists in describing the contour of the shape by an unidimensional string of primitives. Each primitive represents the appropriate local feature of the corresponding object shape. A number of well-known techniques exists to obtain these representations [1].

Due to the cyclic nature of a contour description, it must be invariant with respect to the starting point. For instance, if we have a set of planar figures and the chosen primitives are the symbol a representing a unit length straight segment and the symbol b representing a corner, the strings $ababaab$, $babaaba$ and $abaabab$ represent the same shape.

In order to learn a model, it is useful to write all strings representing the shape in a normal form. For instance, when identifying isosceles triangles we can assume that the strings are written such that the two borders of the same length appears first. Then a string represents an isosceles triangle if it fits the pattern $a^nba^nba^*b$.

During the learning phase one can spend some time in order to put the strings in the normal form, doing it by hand or by some automatic (and usually expensive) way. However, it is very interesting to find a fast way of knowing if there exists a cyclic rotation of the string such that it fits the model in order to use it at test time.

Maes [4], has proposed an algorithm to calculate the edit distance between two cyclic strings with length n and m which works in time $\mathcal{O}(nm \log m)$. Gregor and Thomason [2] propose an algorithm with cubic time complexity, although it proves to be much faster in practice.

In this paper, a technique is described for adapting any context-free parser based on the Cocke-Younger-Kasami algorithm in order to use it for cyclic strings without increasing its time and space complexity.

2. Mathematical background

Let Σ be a finite non-empty set called *alphabet*. A *word* or *string* over the alphabet Σ is a finite Σ-sequence. A typical word can be written as $a_1 \cdots a_n$, with $n \geq 0, a_i \in \Sigma$. We allow $n = 0$, which gives the *null* (or *empty*) *word*, which is denoted by λ. n is called the *length* of x, written $|x|$, and is the number of occurrences of elements of Σ in x. Let Σ^* denote the set of all words and let $\Sigma^+ = \Sigma^* - \{\lambda\}$. If x and y are in Σ^*, then xy denotes a new word called the *concatenation* of x and y.

A *cyclic shift* is a mapping $\sigma : \Sigma^* \rightarrow \Sigma^*$, defined by $\sigma(a_1 \cdots a_n) = a_2 a_3 \cdot a_n a_1$. Let σ^k denote the composition for all $k \in N$ and let σ^0 denote the identity. Two strings x and y in Σ are *equivalent* if $x = \sigma^k(y)$ for some $k \in N$. Clearly, this defines an equivalence relation in Σ^*. The equivalence class of a string x will be denoted with $[x]$, and will be called a *cyclic string*. Let $A \subset \Sigma^*$, then $[A] = \bigcup_{x \in A}[x]$.

A *phrase-structure grammar* is a 4-tuple $G = (N, \Sigma, P, S)$ where N is a finite non-empty set called the *non-terminal alphabet*, Σ is a finite non-empty set called the *terminal alphabet*, $S \in N$ is the *start symbol*, and P is a finite set of *rules* (or *productions*) of the form $\alpha \rightarrow \beta$, where $V = N \cup \Sigma$, $\alpha \in V^*NV^*$ and $\beta \in V^*$.

*This work has been partially supported by the Spanish CICYT under contract TIC93-0633-C022

1015-4651/96 $5.00 © 1996 IEEE
Proceedings of ICPR '96

Let $\alpha', \beta' \in V^*$. String α' is said to *directly generate* β', written $\alpha' \Rightarrow \beta'$, if there exist $\alpha_1, \alpha_2, \alpha, \beta \in V^*$, such that $\alpha' = \alpha_1\alpha\alpha_2$, $\beta' = \alpha_1\beta\alpha_2$ and $\alpha \to \beta \in P$. We write $\stackrel{+}{\Rightarrow}$ for the transitive closure of $\Rightarrow$. The *language generated by* G, written $L(G)$, is the set $L(G) = \{w \in \Sigma^* | S \stackrel{+}{\Rightarrow} w\}$. If G and G' are phrase-structure grammars, then G is *equivalent to* G' if $L(G) = L(G')$.

A phrase-structure grammar is said to be *context-free* if every rule is of the form $A \to \alpha$, where $A \in \Sigma$ and $\alpha \in V^*$. A context-free grammar is said to be in *Chomsky normal form* if all rules are of the form $A \to BC$, $A \to a$ or $S \to \lambda$, where $A, B, C \in N$, and $a \in \Sigma$. It can be shown that any context-free language may be generated by a grammar in Chomsky normal form (CNF) [3].

3. The Cocke-Younger-Kasami algorithm

This algorithm requires grammars to be in Chomsky normal form [3]. The string λ is in a language if and only if the Chomsky normal form grammar for the language has a rule $S \to \lambda$, where S is the start symbol. Furthermore, the rule $S \to \lambda$ cannot be used in the derivation of any non-null strings. Consequently, without loss of generality, we can restrict our attention to non-null input strings and λ-free grammars in Chomsky normal form.

The method works as follows. Let $G = (N, \Sigma, P, S)$ be a Chomsky normal form CFG with no λ-production. Let $w = a_1 \cdots a_n$ be the input string to be parsed according to G. We assume that each a_i is in Σ for $1 \leq i \leq n$. The essence of the algorithm is the construction of a triangular parse table T whose elements are denoted $t_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq n - i + 1$. Each $t_{i,j}$ is a subset of N. Non-terminal $A \in t_{i,j} \iff A \stackrel{+}{\Rightarrow} a_i \cdots a_{i+j-1}$, that is, if A derives the j input symbols following position i. In particular, the input w is in $L(G)$ if and only if S is in $t_{1,n}$.

The algorithm is shown in fig. 1 and works in a time $\mathcal{O}(n^3)$.

4. The Cocke-Younger-Kasami algorithm for cyclic strings

Let $w = a_1 \cdots a_n$ be a string, and let $G = (N, \Sigma, P, S)$ a Chomsky normal form context-free grammar with no λ-production. We want to know if $w \in [L(G)]$, that is, if exists i, such that $1 \leq i \leq n$ and $S \stackrel{+}{\Rightarrow} a_i \cdots a_n a_1 \cdots a_{i-1} = \sigma^{i-1}(w)$. A brute force approach (running a new parse for all i) takes a time $\mathcal{O}(n^4)$.

This complexity can be reduced by using some byproducts of the Cocke-Younger-Kasami algorithm. Suppose that the parsing table for the string ww is built (it takes a time $\mathcal{O}(n^3)$). Let $1 \leq i \leq n$, then, $S \in t_{i,n}$ if and only if $S \stackrel{+}{\Rightarrow} a_i \cdots a_{i+n-1}$. By construction of the string, $a_{j+n} = a_j$ $1 \leq j \leq n$ and therefore $a_i \cdots a_n a_{n+1} \cdots a_{i+n-1} = a_i \cdots a_n a_1 \cdots a_{i-1}$. Thus, $S \in t_{i,n}$ means $S \stackrel{+}{\Rightarrow} a_i \cdots a_n a_1 \cdots a_{i-1} = \sigma^{i-1}(w)$ and then $w \in [L(G)]$.

This technique can be speeded up if one notes than:

- No element $t_{i,j}$ such that $j > n$ are needed in order to compute the elements $t_{i,n}$.
- The computation of the elements $t_{i,j}$ with $i > n$ can be avoided because $t_{i+n,j} = t_{i,j}$. Indeed, $a_{i+n} \cdots a_{i+n+j-1} = a_i \cdots a_{i+j-1}$ for $1 \leq i \leq n - j + 1$.

The new algorithm, incorporating the suggestions above, is shown in fig. 2. The algorithm essentially fills a square parse table instead of a triangular one. Once the table is

Cocke-Younger-Kasami Algorithm

INPUT: A Chomsky normal form CFG $G = (N, \Sigma, P, S)$ with no λ-production and an input string $w = a_1 \cdots a_n \in \Sigma^+$

OUTPUT: The parse table T for w.

METHOD:

for $i = 1$ **to** n

$t_{i,1} = \{A | A \to a_i \in P\}$

for $j = 2$ **to** n

for $i = 1$ **to** $n - j + 1$

$t_{i,j} = \{A | A \to BC \in P, B \in t_{i,k}, C \in t_{i+k,j-k}, 1 \leq k \leq j\}$

end

Figure 1. The Cocke-Younger-Kasami Algorithm

Cocke-Younger-Kasami Algorithm for cyclic strings

INPUT: A Chomsky normal form CFG $G = (N, \Sigma, P, S)$ with no λ-production and an input string $w = a_1 \cdots a_n \in \Sigma^+$

OUTPUT: The parse table T for w.

METHOD:

for $i = 1$ **to** n

$t_{i,1} = \{A | A \to a_i \in P\}$

for $j = 2$ **to** n

for $i = 1$ **to** n

$t_{i,j} = \{A | A \to BC \in P, B \in t_{i,k}, C \in t_{(i+k-1) \bmod n+1, j-k}, 1 \leq k < j\}$

end

Figure 2. The Cocke-Younger-Kasami algorithm for circular strings

$$
\begin{aligned}
G \quad = (\quad & \{A, B, C, D, E, F, S\}, \\
& \{a, b\}, \\
& \{S \rightarrow CD, C \rightarrow AE, E \rightarrow CA, D \rightarrow BF, \\
& F \rightarrow AF, C \rightarrow b, F \rightarrow b, B \rightarrow b, A \rightarrow a\}, \\
& S \\
) &
\end{aligned}
$$

Figure 3. Chomsky normal form context free grammar for the language $\{a^n ba^n ba^* b | n \geq 0\}$.

		S			S			S
		C			C			C
E	D		E	D		E	D	
C		F	C		F	C		F
F	E		F	E		F	E	
A	B,C,F	A	A	B,C,F	A	A	B,C,F	A
a	b	a	a	b	a	a	b	a

Figure 4. parse table for the string *abaabaaba*

computed we have to go across the top row, $t_{i,n}$ for $i = 1$ to n, in order to see if the start symbol is in a cell.

The technique described here can be applied to all versions of the Coke-Younger-Kasami algorithm. For instance, the error-correcting parser version due to Tanaka and Fu [5] stores in each cell $t_{i,j}$ an array, indexed by the symbols in N, containing the weight of the error correcting operation needed in order to accept the string $a_i \cdots a_{i+j-1}$ from the non-terminal symbol. The version for cyclic strings will work using the same rules in order to fill in the cells but filling a square table and paying attention to reading the contents of $t_{i,j}$ when the content of the cell $t_{i+n,j}$ is required. Once the parse table is filled, the minimum weight of the element indexed with S of the arrays in $t_{i,n}$ $1 \leq i \leq n$ is returned.

Similar comments apply to the version for stochastic grammars, that stores in each cell an array, indexed by the symbols in N, containing the probability of producing the corresponding string. In this case, once the parse table is filled in, one has to evaluate the sum for $i = 1$ to n of the elements S of the arrays $t_{i,n}$.

In all these examples the time complexity is $\mathcal{O}(n^3)$.

5. Examples

Let us go back to the example presented in the introduction. Assume that we have a set of shapes and we want to identify the ones that are isosceles triangles. In order to describe the different shapes, we use the contour chains made of two primitives represented by, the symbol a for each straight line of length one, and symbol b for each corner. As said in the introduction, all isosceles triangles are described by a string in the context-free language $\{a^n ba^n ba^* b | n \geq 0\}$. It can be shown that the Chomsky normal form context-free grammar shown in fig. 3 accepts exactly this language.

Usually, the algorithm to extract the contour chains has no reason to begin in the first of the two equal borders of the isosceles triangle. For instance, the same isosceles triangle can be represented by the strings *aaabaaabaab*, *abaaabaabaa* or *baabaaabaaa*. However, the strings *aabaabaabaab* and *abaabaaab* does not represent isosceles triangles.

Let us suppose that the input string is $w = abaabaaba$. The parse table that generates w can be seen in fig. 4. Because some cells at the top row contain the start symbol there exists a shift of the string *abaabaaba* with belongs to the context-free language, that is, the input string represents an isosceles triangle. In fact, as the string represents a equilateral triangle there exist three different valid shifts $(i = 3, i = 6, i = 9)$.

6. Conclusions

We have shown that any Cocke-Younger-Kasami-style context-free parser can be adapted for cyclic strings without increasing the intrinsic complexity of the algorithm. The time complexity remains $\mathcal{O}(n^3)$ and the space complexity $\mathcal{O}(n^2)$. Applications can be found, for instance in shape recognition with contour chains.

7. Acknowledgments

The author wishes to thank Rafael C. Carrasco, Mikel L. Forcada, Paco Moreno and Luis F. Gomis for their comments.

References

[1] K. Fu and T. Young. *Handbook of pattern recognition and image processing*. Academic-Press, 1986.

[2] J. Gregor and T. M.G. Dynamic programming alignment of sequences representing cyclic patterns. *IEEE Trans. Pattern Analysis Mach. Intell.*, (15), 1993.

[3] M. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company, 1978.

[4] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, (35), 1990.

[5] E. Tanaka and K. Fu. Error-correcting parsers for formal languages. *IEEE Transactions on Computers*, C-27(7), July 1978.