A New Top-Down Context-Free Parsing for Syntactic Pattern Recognition

Mehrnoosh Bazrafkan
Department of Computer Engineering, Marvdasht Branch, Islamic Azad University, Marvdasht, Iran

Ali Broumandnia
Department of Computer Engineering ,Islamic Azad University-South Tehran Branch

ABSTRACT

The numerous different mathematical methods used to solve pattern recognition snags may be assembled into two universal approaches:the decision-theoretic approach and the syntactic (structural) approach.
In this paper,at first syntactic pattern recognition method and formal grammars are described and then has been investigated one of the techniques in syntactic pattern recognition called top –down tabular parser known as Earley's algorithm Earley's tabular parser is one of the methods of context -free grammar parsing for syntactic pattern recognition. Earley's algorithm uses array data structure for implementing, which is the main problemand for this reasontakes a lots of time,searching in array and grammar parsing,and wasting lots of memory. In order to solve these problems and most important,the cubic time complexity,in this article,a new algorithm has been introduced,which reduces wasting the memory to zero, with using linked list data structure.Also, withthe changes in the implementation and performance of the algorithm,cubic time complexity has transformed into O (n*R) order.

Keywords

Syntactic pattern recognition, tabular parser,context –free grammar,time complexity,linked list data structure.

1. INTRODUCTION

Syntactic pattern recognition worries the difficult of determining whether a string xbe appropriate to a language L(G) or not,if X is not deformed,this is called a recognition or a parsing problem. The two tabular parsing methods for context-free language, the cocke-kasami-younger (cky) and the Earley termed.In syntactic pattern recognition ,we assume that a pattern can be exemplified by a sequence of primitives and a class of patterns makes a language L(G).consider two grammars G_1 and G_2 which represent the class 1 and the class 2,respectively.If a sequence of primitives x belongs to $L(G_1)$,x is determined to be a pattern of the class 1.if x belongs to neither $L(G_1)$ and $L(G_2)$,x is rejected,[1]-[3].
In the syntactic approach ,formal grammars are used for pattern class representation.The productions of a grammar describe how complex (sub)patterns can be built up from simpler elements.The recognition procedure is based on the conceptof formal language parsing.The most fundumental concept is string grammars.They operateon strings of symbols,i.e words over a finite alphabet.Formal grammars operate on words over finite sets of symbols,[1]-[6].

Definition : A formal grammar is a four-tuple $G = (V_N, V_T, S, P)$ where V_N is a finite set of nonterminals symbols,V_T is a finite set of terminal symbols,P is a finite set of production set or rewriting rules and $S \in V_N$is the initial or starting symbol.It is essential that $V_N \cap V_T = 0$,the union of V_N and V_T is called the vocabulary $V = V_N \cup V_T$.
Definition 2:

(a) A grammar is called unrestricted or of type 0,iff any production is of the form $\alpha \rightarrow \beta$ where $\alpha \in V^+$and $\beta \in V^*$.
(b) A grammar is called context sensitive or of type 1,iff any production is of form $xAy \rightarrow xzy$ where $x, y \in V^*$; $A \in V_N$;$z \in V^+$.
(c) A grammar is called context –free or type 2,iff any production is of the form $A \rightarrow z$ where $A \in V_N$ and $z \in V^+$.
(d) A grammar is called regular or of type 3,iff any production is of the form $A \rightarrow aB \; or \; A \rightarrow a$ where $A, B \in V_N$ and $a \in V_T$.

Since context-free grammars play the most important in syntactic pattern recognition, we will limit considerations to this type of grammar,[13].
A parse problem is a problem to decide whether or not I is in L(G) and to make the syntactic tree(s) of I,if I is in L(G).A machine to parse a string is called parser,[1].
Parsers can be classified into the following four categories:
(a) Parsers with push down lists[11].
(b) Direct parser[12].
(c) Transition diagram parser[15]
(d) Tabular parser[11]
Tabular parser are separated into two groups:

(a) A bottom-up parser
(b) A top –down parser

Here we will focus on top-down tabular parser or Earley's algorithm of a general context free language.

2. A TOP-DOWN PARSER

In this section,a parser for a context –free –language proposed by Earley is described,[14]. This parser usages an item of the form $(A \rightarrow \alpha.\beta, i)$ which means that α has been parsed. A list L[i] (i=0,1,…,n) is a set of items. The number "i" in $(A \rightarrow \alpha.\beta, i)$directs that the item is produced from an item in L[i].let λ be the null string [7]-[10],[14].
Define the operation $L[k] \cup \{(X \rightarrow \alpha.\beta, j)\}$ in the two ways as follows:

Define 1: If $(X \to \alpha.\beta, i) \in L[k] and\ i \leq j$,do nothing ,otherwise. Add $(X \to \alpha.\beta, j) to\ L[k]$.

Define 2: If $(X \to \alpha.\beta, i) \in L[k] and\ i = j$, do nothing. Otherwise, add $(X \to \alpha.\beta, j) to\ L[k]$.

Definition 1 suits for the recognition problem and the parse problem to construct only one parse tree of an input .If all parse trees of an input are necessary ,we must use definition 2.

In following, Earley parser is shown.

Algorithm :Top-Down Parser Earley

Input : CFG $G = (V_N, V_T, p, S)$ and input string $I = a_1 a_2 \ldots a_n$

Output :the parse list L[0],L[1],…,L[n];

Algorithm Earley

1 begin {∗ of the Earley ∗}

2 $L[0] := \{(\$ \to \cdot S, 0)\}$;

3 for i := 1 to n do

$L[i] := \emptyset$; (∅ is the null set)

4 i := 0;

5 while ($i \leq n$)do begin

6 for each $(X \to \alpha \cdot Y\beta, j) \in L[i]$do begin

7 if $Y \in V_N$ then begin {∗ predictor ∗}

8 for each $Y \to \gamma \in P$ do

9 $L[i] = L[i] \cup \{(Y \to \cdot \gamma, i)\}$;

10 end

11 else if ($Y \in V_T$ and i
≠ n) then begin {∗ scanner ∗}

12 if $Y = a_{i+1}$ then

13 $L[i+1] := L[i+1] \cup \{(X \to$
αY.β,I;

14 end

15 else if

$Y\beta = \Lambda$ then begin {∗ completer ∗}

16 for each $(A \to \delta.X\xi, k) \in L[j]$do

17 $L[i] := L[i] \cup \{(A \to \delta.X\xi, k)\}$

18 end

19 end {∗ of for i ∗};

20 if $\begin{pmatrix}\text{no new item has been generated} \\ \text{in L[i]from lines } 6-19\end{pmatrix}$

21 then i := i + 1;

22 end {∗ of while ∗}
23 if $((S \to \alpha.,0)$
$\in L[n]$then I is accepted with weight w

24 else I is rejected;

25 end{∗ of Earley ∗}.

The Earley has the following properties:

(1)If$(A \to \alpha \cdot \beta, i) \in L[j] then$

$$\alpha \overset{*}{\to} \alpha_{i+1}\alpha_{I+2} \dots \alpha_j$$

(2) The space and time complexities to make the parse lists are $O(n^2)$ and$O(n^3)$,respectively.The time complexity to create all syntactic trees of a string is $O(c^n)$[14].

In the following, how the parsing grammar and input string searching perform, it has been shown with an example.
Consider CFG $G = (V_N, V_T, p, S)$ where $V_N = \{S, T, A, B\}$,$V_T = \{a, b\}$, $P = \{S \to T, S \to AB, T \to aTb, T \to ab, A \to aA, A \to a, B \to bB, B \to b\}$ and $L(G) = \{a^n b^m | n, m \geq 0\}$ and input string I=aabb.Fig 1 shows the procedure of described algorithm.

L[0]	(1) $(\$ \to .S, 0)$ *initial setting* (2) $(S \to .T, 0)$ $predict(1)$ (3) $(S \to .AB, 0)$ $predict(1)$ (4) $(T \to .aTb, 0)$ $pred(2)$ (5) $(T \to .ab, 0)$ $pred(2)$ (6) $(A \to .aA, 0) pred$ (3) (7) $(A \to .a, 0)$ $pred(3)$
L[1]	(8) $(T \to a.Tb, 0) scan$ (4) (9) $(T \to a.b, 0) scan(5)$ (10) $(A \to a.A, 0) scan$ (6) (11) $(A \to a.,0) scan(7)$ (12) $(T \to .aTb, 1) pred(8)$ (13) $(T \to .ab, 1) pred(8)$ (14) $(A \to .aA, 1)$ $pred(10)$ (15) $(A \to .a, 1)$ $pred(10)$ (16) $(S \to A.B, 0) complete(11,3)$ (17) $(B \to .bB, 1) pred(16)$ (18) $(B \to .b, 1) pred(16)$
L[2]	(19) $(T \to a.Tb, 1) scan$ (12) (20) $(T \to a.b, 1) scan(13)$ (21) $(A \to a.A, 1) scan(14)$ (22) $(A \to a.,1) scan(15)$ (23) $(T \to .aTb, 2) pred(19)$ (24) $(T \to .ab, 2) pred(19)$ (25) $(A \to .aA, 2) pred(21)$ (26) $(A \to .a, 2) pred(21)$ (27) $(S \to A.B, 0) comp(22,3)$ (28) $(B \to .bB, 2) pred(27)$ (29) $(B \to .b, 2) pred(27)$
L[3]	(30) $(T \to ab.,1) scan(20)$ (31) $(B \to b.B, 2) scan(28)$ (32) $(B \to b.,2) scan(29)$ (33) $(B \to .bB, 3) pred(30)$ (34) $(B \to .b, 3) pred(30)$ (35) $(S \to AB.,0) comp(32,27)$ (36) $(\$ \to S.,0) comp(35,1)$ (37) $(T \to aT.b, 1) comp(30,19)$ (38) $(S \to T.,0) comp(30,2)$ (39) $(\$ \to S.,0) comp(38,1)$
L[4]	(40) $(B \to b.B, 3) scan(32)$ (41) $(B \to b.,3) scan(33)$ (42) $(B \to .bB, 4) pred(36)$ (43) $(B \to .b, 4) pred(36)$
	(44) $(S \to AB.,0) comp(27,37)$ (45) $(\$ \to S.,0) comp(40,1)$ (46) $(T \to aTb.,1) scan(37)$ (47) $(T \to aT.b, 1) comp(46,19)$ (48) $(S \to T.,0)$comp(46,2) (49) $(\$ \to S.,0) comp(48,0)$

Fig 1. Earley algorithm for I=aabb

3. THE EARLEY ALGORITHM'S DISADVANTAGES

As it has been seen, this algorithm performs the parsing grammar and searching input string, in three steps predict, scan and completing. In order to prevent having the repeating items each time, when it expects, new item adds to the list, comparing that with all items exist at the list, happens. Also in each completing performance all items at each rows should checked out, to find the specific item and all these takes lots of time at the grammars with lots of rules and will have the very high time complexity, which will be at the $O(n^2)$order in the best case[15]-[16].
This algorithm performs, recognizing and grammar parsing in the very complicated way and takes lots of time.Another problem is using array data structure at implementing. One of the array data structure problem is the fixed length which has to be very definite from the beginning, so in order to implement the algorithm we have to consider the array bigger than usual therefore it won't having the problem when new item produces, and perform well for different grammars with the different rules,but it might lots of the memory spaces stay vacant and wastelots of memories.

4. THE PROPOSED ALGORITHM

Cause of the problems have been mentioned, we introduce an algorithm which won't have most of the previous problems. In this algorithm at first in order to solve the array's problem, use of the linked list has been suggested reason for that: this data structure is flexible for the length changing during performance, also insertion and deletion of the element at linked list is doable with O (1) easily. Therefore at this algorithm we allow all the nodes to be added in the list and it won't be needed tocompare anymore. Another thing about this suggested algorithm is : with the changes at the Earley's performance, would haven't been need to compare so would have been deleted lots of comparing. Suggestedalgorithm, does its own performance in n steps (n is the length of the input string) and maximum at each steps will produce nodes equal to the rules exist in rules set (called R) therefore n*R nodes produce in order to recognize the input string, in the other way R represents the number of the repeating while(p!=null)loop. At each step three comparison happensso n*R*3comparison are needed for all nodes exist at linked list .All these are less than cubic order at the Earley algorithm. Also cause using the linked list the quantity wasting memory has become zero. This algorithm won't need to completed operation. So lots of comparisons cause of completed operations will be eliminated. In order to use of the suggested algorithm shouldn't exist left-recursion in

grammar.Suggested algorithm has the time complexity of the O(n*R)(n is length of the input string and R is the number of rules in the rule set).fig 2 shows the proposed algorithm as described up.

```
1 new link list*p,*q;
2 p → data = ($ →.S)
3p → next = null;
4q = p;
6  i = 0;
7  while (i ≤ n)
8         new link list *r,*h;
9         r → data = header;//list is not empty //
9          h = r;
10         While(p!=null)
11           if(p → data = ($ → α.Yβ) )
12             if (Yβ = λ)
13                        if(i ≠ n)
14                           Break;
15                        elseif(i == n)
16
(print  input string is accepted);
17            else if(Y is nonterminal)
18                        if(i ≠ n)
19                    foreach(Y → ψ ∈
production set){*predictor*}
20                           New node  * temp;
21                            temp → data = ($ →
α.ψβ)
22                            temp → next = null;
23                            q → next = temp;
24                      else if(i == n)
25                            break;//go to line 40 //
26          else if(Yis terminal)
27                        if(i ≠ n)
28                        if(Y = a_{i+1}){*scanner*}
29                           new node ∗ temp;
30                           temp → data = ($ → αY. β);
31                           temp → next = null;
32                          r → next = temp
33                          r = temp;
34                           else
35                             Break;//go to line42//
36
37                    else
38                             break;
39               end
40         p = p → next;
41         }
42  p → next = h;
43  q = r;
44  i = i + 1;
45  }
```

Fig 2. Suggested algorithm

At the following, the example of the part 2, with getting help from the suggested algorithm, for the input string (I=aabb)has been represented. Fig 3 represents function of the suggested method.

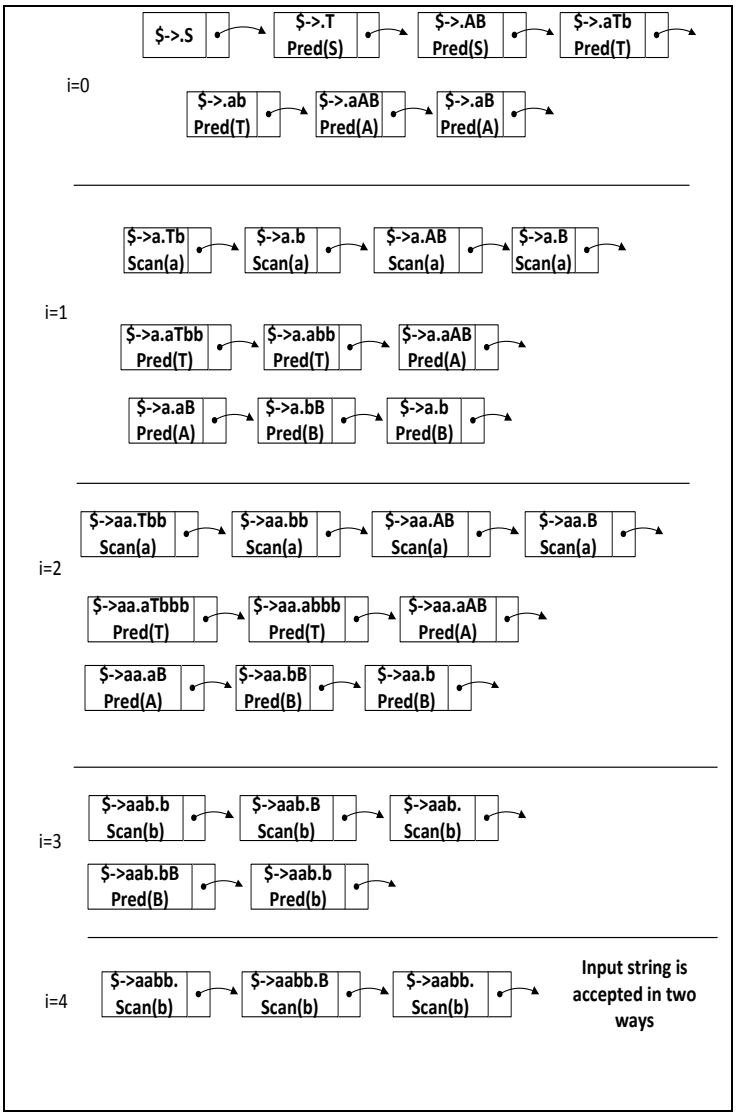

Fig.3 suggested algorithm for I=aabb

5. CONCLUSION

At this article , At first ,we have reviewed one of the tabular parsers which has been used at recognizing syntactic pattern ,called Earley ,and then we discussed about advantages and disadvantages. Then we suggested an algorithm which decreased the cubic time complexity of the Earley algorithm to the O(n*R)time complexity with using the linked list and changing at parsing performance. Alsodecrease the amount of wasting memory to the zero with the linked list.

6. REFERENCES

[1] Bunke, Horst. Syntactic and structural pattern recognition: theory and applications. Vol. 7. World Scientific Publishing Company Incorporated, 1990.

[2] K.S.Fu ,Syntactic Pattern Recognition And Application,Chapter 5,Prentice hall ,Engelwood Cliffs,NJ,1982

[3] Jain, Anil K., Robert P. W. Duin, and Jianchang Mao. "Statistical pattern recognition: A review." Pattern Analysis and Machine Intelligence, IEEE Transactions on 22.1 (2000): 4-37.

[4] Kandel, Abraham. Introduction to Pattern Recognition, Sta: Statistical, Structural, Neural and Fuzzy Logic Approaches. Vol. 32. World Scientific Publishing Company, 1999.

[5] Bunke, Horst. "Recent advances in structural pattern recognition with applications to visual form analysis." Visual Form 2001. Springer Berlin Heidelberg, 2001. 11-23.

[6] Suganthan, Ponnuthurai N. "Structural pattern recognition using genetic algorithms." Pattern Recognition 35.9 (2002): 1883-1893.

[7] Mitra Basu, Horst Bunke, and Alberto Del Bimbo ,Guest Editors' Introduction to the Special Section on Syntactic and Structural Pattern Recognition IEEE TRANSACTIONS VOL. 27, NO. 7, JULY 2005.

[8] Suganthan, Ponnuthurai N. "Structural pattern recognition using genetic algorithms." Pattern Recognition 35.9 (2002): 1883-1893.

[9] Civera, Jorge, et al. "A syntactic pattern recognition approach to computer assisted translation." Structural, Syntactic, and Statistical Pattern Recognition. Springer Berlin Heidelberg, 2004. 207-215.

[10] Conte, Donatello, et al. "Thirty years of graph matching in pattern recognition."International journal of pattern recognition and artificial intelligence 18.03 (2004): 265-298.

[11] A.V.Aho and J.D.Ullman,The theory of parsing ,Translation , and compiling ,Vol 1:parsing,(prentice –hall, Englewood cliffs ,NJ ,1972).

[12] E.Tanaka, M.Ikeda and k.esure,"direct parsing",pattern recognition 1986. 315-323

[13] Miclet, Laurent. Grammatical inference. World Scientific, 1990.

[14] Earley, J.: An Efficient Context-flee Parsing Algorithm Ph. D. Thesis (Carnegie- Mellon University, 1968).

[15] Aho, A.V. and Ullman, J. D ,The Theory of Parsing, Translation, and Compiling, Parsing, vol. I (Prentice-Hall, 1972).

[16] L.Miclat,Structural Methods in Pattern recognition (Springer-verlag,New York 1986)