A Fast Multiple Pattern Matching Algorithm using Context Free Grammar and Tree Model

G.Phanindra[1] K.V.V.N. Ravi Shankar[2] P.Deepak Sreenivas[3,]

[1]Computer Sciences Corporation India Pvt Ltd, Hyderabad, Andhra Pradesh, India
phani541@yahoo.co.in
[2]Infosys *Chennai, TamilNadu, India*
ravi523096@yahoo.com
[3] Tata Consultancy Services Limited , *Hyderabad, Andhra Pradesh,* India
deepaksreenivas.p@gmail.com

Summary

Multiple Substring-Pattern Matching Algorithm presented here is implemented in two phases. The first phase is preprocessing in which an n-ary tree like structure is constructed for the given text data and Griebach Normal Form is created for given Context Free Grammar. The second phase called search phase takes as input an n-ary tree structure and Griebach Normal Form of given Context free grammar constructed in phase 1 and outputs those strings that match both the text data and the context free grammar. The algorithm proposed here has the advantage that it can retrieve any number of patterns at the same time. This finds wide applications in Bio-informatics, information retrieval and requirements specification stage of software life cycle development.

Key words

Pattern Matching, preprocessing, n-ary tree, Context Free Grammar, Griebach Normal Form, Bio-informatics, information retrieval

I. Introduction

Pattern matching is the act of checking for the presence of the constituents of a given pattern. In the present work the query is given in the form of Context Free Grammar.

Context Free Languages are widely used in different areas, including programming languages, speech recognition, natural language processing [6], bioinformatics. and requirements specification stage of software life cycle development.

The complexity of parsing words according to context free grammars is usually considered as having theoretically two parameters: the length of input sequence and the size of grammar, namely the number of rules or sum of the length of the rule bodies. The strings in the context free language are accepted by a Push down automaton [2], in our present work we have bypassed the construction of a push down automaton for the acceptance of the strings generated by the context free language. The time taken by our algorithm is independent of the size of patterns, which overcomes the major limitation which is present in the previous algorithms.

The syntax rules of formal grammars are used to generate patterns. Natural English Language can also be derived from a context free grammar with appropriate productions. For a given grammar G, the number of strings generated by that grammar is infinite that means |L (G)| is ∞. Parsing is a fundamental concept related to the syntactic approach whose objective is to determine if the input pattern is syntactically well formed in the context of the given grammar. Parsing is generally accomplished by parsers. In the search algorithm presented here we find the intersection of the strings present in the text database and the language of the given grammar, for that we have not used any parser. Multiple substrings can be obtained at same time using the search algorithm presented here.

2. Preprocessing phase

2.1 Preprocessing of text data

The text data is preprocessed so that search time is optimized. Text data is nothing but the set of strings. The text data is represented with a structure similar to n-ary tree. The text data is taken, each string is passed to tree and for each string the tree is modified. The tree structure is similar to n-ary tree where n represents the number of distinct symbols in the given text data, in present work. The node structure for tree is given by Madhuri et al [4]

2.1.1 Node Structure

Each node of the tree has the following fields.

i) Information Field:

This field contains information.

ii) Pointers to the child nodes:
The pointers that hold the addresses of child nodes (the no of children is at max is n).

iii) Flag :
Flag is used to recognize whether the particular string obtained by concatenating the strings from the root up to the current node in the left to right sequence is present as an element in the database or not. The flag of a node is 1 if and only if the string up to the corresponding node is contained in the database.

The algorithm for the construction of database tree given by Madhuri et al [4]

2.2 Preprocessing of the context free grammars:

The query is given in the form of context free grammars. The given context free grammar is converted into a Griebach Normal Form which is also a context free grammar. The language generated by both the context free grammar and its Griebach normal form is same. The preprocessing of the context free grammars increases the efficiency of searching. The conversion takes place in five steps. They are

Step1: Elimination of Useless symbols
Step2: Elimination of Null productions
Step3: Elimination of Unit Productions
Step4: Conversion to Chomsky like normal Form
Step5: Conversion to Greibach normal form

The algorithm for the conversion of given context free grammar to Griebach Normal form is given by Madhuri et al [4]

3. Search Phase

A context-free grammar G consists of the following four entities:

1. The set of terminals or primitive symbols denoted by T. In many applications, the choice of the terminal set is difficult and has a large component of art as opposed to science. T is finite.

2. The set of non-terminal symbols or variables which are used as intermediate quantities in the generation of outcome consisting solely of terminal symbols. This set is denoted as V and it is also finite.

3. The set of productions or production rules that allow the previous substitutions. It is this set of productions coupled with terminal symbols that principally gives the grammar its structure. The set of productions is denoted by P

4. The starting or root symbol denoted by S S belongs to V.

The grammar G is denoted formally as G= (T, V, P, S)

A language L is said to be a Context-Free-Language (CFL) if its grammar is Context-Free. The production rules P, is used to generate sentences that consist of linear or 1-D strings of terminals. The length or number of symbols in string s is denoted by |s|. The empty string is denoted by ε. The size of empty string is 0. Intersection of the terminals and the variables is empty set ($V \cap T = \Phi$).

3.1 Algorithm for searching the database tree using Context Free Grammars

Input: 1. n-ary tree representation of text data
2. Griebach Normal Form of given context free grammar
Output: List of data base tree pointers that matches the given context free grammar

```
struct list *search_cfg(char *prod,struct list *db_list)
//Searching CFG which is in GNF
{
  // 'prod' is a string that is in the form of terminal followed by non terminals
  // 'db_list' is the list of database pointers in the format of single linked list
   if(is_terminal(prod[0]))
   {
     match the terminal in 'prod[0]' to the present node in the 'db_list'
      if(there is no match)
      {
        return(NULL)
      }
     else
     {
        Make the present node in the 'db_list' to
```

```
        the next node to be searched in the
          database
        if(the node in the 'db_list' to be searched
        is NULL)
        {
            if ((production+1) = =NULL)
            {
                return (db_list);
            }
            else
            {
                Return(NULL);
            }
        }
        else
        {
            Temp_list=search_cfg(production+1,db
            _list)
            return(Temp_list);
        }
    }
}
else
{
    Temp_list1=NULL;
    for each right hand side production
    'temp_prod' of non-terminal 'prod[0]'
    {
        Append the returned list from
        'search_cfg(temp_prod,db_list)' to
        Temp_list1
    }
    Temp_list3=NULL;
    for each 'Temp_list2' in 'Temp_list1' linked
    list
    {
        Append the returned list from
        'search_cfg(prod+1,Temp_list2)' to
        Temp_list3
    }
    return(temp_list3);
}

}

void CFG_substring_matching()
{
    db_list db_ls_2, db_ls_3;
for each node 'node_1' in data base tree
{
    for each character position 'pos_1' in 'node1'
    {
        db_list db_ls_1;
        db_ls_1 = create new db_list;
        db_ls_1->pos = pos_1;
        db_ls_1->data = node_1;
        db_ls_3 =
            search_cfg(db_ls_1,Starting_charact
            er_of_CFG);
        Append 'db_ls_3' to 'db_ls_2';
        return(db¬_ls_2);
    }
}
}
```

This is the algorithm that finds out all the substrings for the given database that satisfies the Context Free Grammar. The routine 'search_cfg' for finding the Complete-strings is used in this algorithm internally. For each and every node in the database, we will call the 'search_cfg' routine so that all the substrings also retrieved.

4. Applications

Requirements are the basis of the systems engineering life cycle activities[5] but creating a good set of requirements is really difficult task. Some difficulties can be reduced through the application of a context- free grammar for requirements to reduce the complexity of requirements elicitation [7]. Developing the grammar involved a melding of computer science and natural language that yielded useful insights into the nature of requirements. The grammar was developed to empower a case-based assessment system for requirements.

Bioinformatics involves the use of techniques including applied mathematics, informatics, statistics, computer science, artificial intelligence, chemistry and biochemistry to solve biological problems usually at a molecular level. The algorithm presented here can be used to identify specific patterns of amino acids in the DNA sequence [3] and thereby help in the diaganosis of certain diseases [1]. In this case, the terminal set consists of {a,c,g,t}.

5. Conclusions

The algorithm proposed here is very helpful in retrieving the substrings of given context free language in an easy and efficient manner. The specialty of the algorithm proposed here is that it do not require any Push down automata for the acceptance of the strings generated by given context free grammar. The time taken to retrieve the substrings is same for same size of text database and same context free grammar that means search time independent of the content of text database.

6. References:

[1] Abarbanel, R.M., Wieneke, P.R., Mansfield, E., Jaffe, D.A., and Brutlag, D.L. (1984), ''Rapid searches for

complex patterns in biological molecules,'' *Nucleic Acids Research*, Vol. 12, No. 1, pp. 263-280.

[2] Aleksander, I. & Hanna F.H., 1976 *Automata Theory: An Engineering Approach,* Crane,Russak & Company, Inc.

[3] Hawley, D.K., and McClure, W.R. (1983), ''Compilation and analysis of Escherichia coli promoter DNA sequences,'' *Nucleic Acids Research*, Vol. 11, No. 8, pp. 2237-2255.

[4] IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.3, March 2007

[5] INCOSE, 2000 *Systems Engineering Handbook,* (Online 20 October, 2003)

[6] Luger, G. F., & Stubblefield, W. A., 1998, *Artificial Intelligence, Structures and Strategies For Complex Problem Solving,* Addison Wesley Longman Inc

[7] Martin, J., 2000, 'Requirements Mythology: Shattering Myths About Requirements and the Management Thereof', *Proceedings of the 2000 INCOSE Symposium*

Author Biographies:

G.Phanindra resident of Vizianagram born on 22-12-1985.He received B-Tech degree in Computer Science and Engineering from Gayatri Vidya Parishad College of Engineering in Visakhapatnam, Andhra Pradesh, India, and the year 2007. Currently working as a Software Engineer in Computer Sciences Corporation India Pvt Ltd (CSCI). He received Best Outgoing Student Award from Tata Consultancy Services. He received Best All Rounder Award from Naval Science and Technological Laboratories. He presented 10 national level paper presentations. His current research interests include datamining, natural language programming and compiler design.

Ravi Shankar Kalla resident of Visakhapatnam. He received B-Tech degree in Computer Science and Engineering in the year 2007 From Gayatri Vidya Parishad College of Engineering (India). Won prizes for student projects presented in Jawaharlal Nehru Technological University and Andhra University. His research interests include Computer graphics, DBMS, Data Warehousing, Compiler design. Currently working in Infosys Technologies Ltd. Chennai (INDIA).

Deepak Sreenivas Pemmaraju resident of Visakhapatnam. He received B-Tech degree in Computer Science and Engineering from Gayatri Vidya Parishad College of Engineering in Visakhapatnam, Andhra Pradesh, India in the year 2007. Currently working as a Software Engineer in Tata Consultancy Services Limited (TCSL) India. He presented 10 national level paper presentations. His current research interests include software methodology and engineering, compiler design and natural language programming.