Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Procedural Cognitive Complexity Measure of Tree Search Algorithms

Adedapo O. A., Ganiyu R. A.*, Olabiyisi S. O., Omidiora E. O., Sijuade A. A.
Department of Computer Science &Engineering, Ladoke Akintola University of Technology
Nigeria

Abstract*— Complexity of a computer software determines its efficiency while software metrics reveals its quality. However, many conventional software metrics like Lines of Code, Halstead Complexity Metrics and MCabe Cyclomatic Complexity Metrics focus on the number of variables with less emphasis on the cognitive structure of programming. Therefore, this paper uses procedural cognitive complexity measure (PCCM) to analyze four selected tree search algorithms, which include Breadth-first search algorithm, Depth-first search algorithm, Depth-limited search algorithm and A* search algorithm. Each of the algorithms is implemented using C, Pascal and Visual BASIC programming languages. Evaluation of the complexity of each implementation is carried out using PCCM. The complexities obtained are compared to determine the best programming language for implementing each of the tree search algorithms. The results of the complexity analysis show that breadth-first search algorithm assumes PCCM value of 534, 460 and 550 using C, Pascal and Visual BASIC programming languages, respectively. Also, depth-first search algorithm yields PCCM value of 427, 442 and 549 when implemented in C, Pascal and Visual BASIC programming languages, respectively. Similarly, depth-limited search algorithm assumes PCCM value of 567, 563 and 726 when implemented in C, Pascal and Visual BASIC programming languages, respectively while A* search algorithm yields PCCM value of 886, 913 and 1092 using C Pascal and Visual BASIC programming languages, respectively. The results of this study imply that the choice of programming languages affect the complexities of programs of the tree search algorithms. The less the number of arbitrarily named variables the less complex the implementation of the tree search algorithms.

Keywords*— Procedural cognitive complexity measure (PCCM), Breadth-first search algorithm, Depth-first searchalgorithm, Depth-limited search algorithm and A* search algorithm.

I. INTRODUCTION

The analysis of algorithm is a major task in computing. A computer scientist, most especially a programmer who is faced with the problem of choosing the appropriate algorithm to solve his problem from myriad of available ones, may have his problem solved by analyzing the complexity of each of these algorithms in order to know the best. Algorithms are frequently assessed by the execution time and by the accuracy or optimality of the results. For practical use, a third important aspect is the implementation complexity. An algorithm which is complex to implement requires skilled developers, longer implementation time, and has a higher risk of implementation errors. Moreover, complicated algorithms tend to be highly specialized and they do not necessarily work well when the problem changes [1].

Algorithms analysis is an important part of a broader computational complexity theory, which provides estimate for the resources needed by any algorithm which solves a given computational problem. This estimate provides an insight into reasonable direction of search of efficient algorithms. Complexity of an algorithm is the determination of the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with input of arbitrary length. Usually, the efficiency or complexity of an algorithm is stated as a function relating the input length to the number of step (time complexity) or store locations required to execute the algorithms.

Algorithm can be studied theoretically. Theoretical analysis allows mathematical proofs of the execution time of algorithms but can typically be used for worst case analysis only. Therefore, empirical analysis is often necessary to study how an algorithm behaves with typical inputs. In computer science, tree search (also known as tree traversal) is a form of graph traversal and it refers to the process of visiting (examining and/or updating) each node in a tree data structure, exactly once, in a systematic way. Such traversals are classified by the order in which the nodes are visited. Complexity of tree search algorithms have been mostly evaluated either mathematically or by computing the computer execution time. Neither of the two approaches is good enough for practical and realistic purposes especially in the situation where more than one algorithm, exists for solving a given problem or class of problems. Therefore there is a need therefore to seek for pragmatic means of computing complexity of algorithms. The evolutionary gradient in both the hardware and software would greatly be influenced by the effort since the development of the hardware and software revolves around the concept of algorithm.

II. LITERATURE REVIEW

Given a problem, how can we find an efficient algorithm for its solution? How can we compare this algorithm with other algorithms? Questions of these types are of interest to both programmers and theoretically oriented computer scientists. To compare the efficiency of algorithms; measure of the degree of the difficulty of an algorithm called computational complexity was developed [2]. Algorithm can be studied theoretically and empirically. Theoretical analysis allows mathematical proves of execution time of algorithms by calculating the rate at which the storage or time grows as a function of the problem size. Theoretical analysis concentrates on a proportionality approach, expressing the complexity in terms of its relationships to some known functions. There are a large number of implementation complexity measures available, such as: Lines of Code (LOC), Halstead metrics, MCabe Cyclomatic number and PCCM (to mention a few). The advantage of PCCM over others is that PCCM measures the cognitive aspect of programming language while others do not.

Software complexity is defined as ‘’the degree to which a system or component has a design or implementation that is difficult to understand and verify’’ [3]. That is, the complexity of a code is directly dependent on the understandability. All the factors that make a program difficult to understand are responsible for its complexity. Software complexity also is an estimate of the amount of effort needed to develop, understand or maintain the code and the more complex the code is the higher the effort and time needed to develop or maintain this code [4]. There are many software complexity measures that can be used to analyze the complexities of algorithms. These includeLines of Code, Halstead metrics and MCabe Cyclomatic number and PCCM.

[5]carried out performance evaluation of Procedural Cognitive Complexity Metric and Other Based Complexity Metrics. The total complexity of a procedural language is given by the formula: Procedural Cognitive Complexity Measure (PCCM)

$$PCCM = \sum_{i=1}^{n} \sum_{j=1}^{mi} \left((4 * ANV + MNV) + Operators + Constants\right) * CWU_{ij} \quad (1)$$

Where: ANV is number of arbitrarily named variables;
MNV is meaningfully named variables;
Operator is number of operators;
Constant is number of constants;
CWU is cognitive weight unit.

Definitions of complexity imply that all the factors which make code difficult to understand are responsible for complexity. Accordingly, the factors which are responsible for the complexity of a procedural code should be identified. When procedural codes are analysed, it is found that the following factors are responsible for cognitive complexity:
Number of Arbitrarily Named Variables (ANV) [6].
Number of Meaningfully Named Variables (MNV) [6].
Number of Operators [7].
Cognitive weight of basic control structures (BCS) [8].
Tree search algorithms are the heart of searching techniques. The basic principle is that a node is taken from a data structure, its successors examined and added to the data structure. By manipulating the data structure, the tree is explored in different orders, level by level (Breadth–first search) or reaching a leaf node first and backtracked (Depth-first search). A number of tree search algorithms were reported in literature. These include:

- Breadth-first search
- Depth-first search
- Depth-limited search
- A* search algorithm

Breadth-first search (BFS) is an algorithm that begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, it explores their unexplored neighbour nodes, and so on, until it finds the goal. BFS is an uninformed search method that aims to expand and examine all nodes of a graph systematically in search of a solution. In other words, it exhaustively searches the entire graph without considering the goal until it finds it. It does not use a heuristic [9]. From the standpoint of the logarithm, all child nodes obtained by expanding a node are added to a FIFO queue. In typical implementations, nodes that have not yet been examined for their neighbors are placed in some container (such as a queue or linked list) called “open” and then once examined are placed in the container “closed”.

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one starts at the root and explores as far as possible along each branch before backtracking. Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it had not finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a Last in-First out (LIFO) stack for expansion [9].

Space complexity of DFS is much lower than BFS (breadth-first search). It also lends itself much better to heuristic methods of choosing a likely-looking branch. Time complexity of both algorithms are proportional to the number of vertices and the number of edges in the graphs they traverse. The simple solution of “remember which nodes I have already seen” does not always work because there can be insufficient memory. This can be solved by maintaining an increasing limit on the depth of the tree, which is called iterative deepening depth-first search [9]. Like the normal depth-

first search, depth-limited search is an uninformed search. It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search. Even if the search could still expand a vertex beyond that depth, it will not do so and thereby it will not follow infinitely deep paths or get stuck in cycles. Therefore depth-limited search will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs [9].

A* (Pronounced 'A star') is a tree search algorithm that finds a path from a given initial node to a given goal node. It employs a heuristic estimate that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. The A* algorithm is therefore an example of a best-first search [10].

III. RESEARCH METHODOLOGY

In this study the following approaches were used:

(i) Extensive study of Breadth-first search, Depth-first search, Depth-limited search, and A* search algorithms were carried out.
(ii) The use of C, Pascal and Visual BASIC programming languages to code the selected tree search algorithms.
(iii) Evaluation of the complexities of the algorithms using Procedural Cognitive Complexity Measure (PCCM).
(iv) Analysis and comparative study of the complexities of these set of tree search algorithms.

The Breath-First search algorithm under consideration consists of the following sequential steps:

```
procedurebfs (v)
        q : = make _ queue ( )
        enqueue (q, v )
        mark v as visited
        while q is not empty
              v = dequeue (q)
        process  v
        for all unvisited vertices v' adjacent to v
              mark v' as visited
              enqueue (q,v')  [9].
```

The Depth-First search algorithm under consideration consists of the following sequential steps:

```
dfs (graph G)
{
list L = empty
tree T = empty
choose a starting vertex x
search (x)
while (L is not empty)
        remove edge (v, w) from end of L
ifw not yet visited
          {
add (v,w) to T
search (w)
          }
        search (vertex v)
{
visit v
for each edge (v,w)
add edge (v,w) to end of L
}   [9].
```

The Depth-Limited search algorithm under consideration consists of the following sequential steps:

```
DLS (node, goal, depth)
{
if (node = = goal)
return node;
else
  {
stack ;= expand (node)
while (stack is not empty)
     {
node' := pop (stack);
if (node' . depth ( ) < depth);
DLS(node', goal, depth);
```



```
else
        ; / / no operation
       }
    }
} [9].
```

The **A*** search algorithm under consideration consists of the following sequential steps:

```
function A* (start, goal)
    var closed := the empty set
    var q : = make_queue (path (star))
    While q is not empty
        var p := remove_ first (q)
        var x := the last node of p
        If x in closed
            Continue
        If x = goal
            return p
        add x to closed
        for each  y in successors (p)
            If the last node of y not in closed
                enqueue (q,y)
    Return failure  [9].
```

The performance evaluation of breadth-first search algorithm when implemented in C, Pascal and Visual VASIC programming languages was determined using equation (1). Similarly, through equation (1), the perfomance of each of depth-first search algorithm, depth-limited search algorithm and A* search algorithm was obtained when implemented in C, Pascal and Visual BASIC programming languages.

IV. DISCUSSION OF RESULTS

Having implemented all the four tree search algorithms in C, Pascal and Visual BASIC programming languages, the results obtained are presented in Tables 1 - 4. Table 1 shows the various PCCM values of breadth-first search algorithm when implemented in C, Pascal and Visual BASIC languages. It is explicitly shown that breadth-first search algorithm has the least PCCM (cwu) when programmed in Pascal language, followed by C language and Visual BASIC language. This means that breadth-first search algorithm is best implemented in Pascal language and worst implemented in Visual BASIC language. Also, Table 2 shows the various PCCM values of depth-first search algorithm when implemented in C, Pascal and Visual BASIC languages. It is explicitly shown that depth-first search algorithm has the least PCCM (cwu) when programmed in C language, followed by Pascal language, and Visual BASIC language. This means that depth-first search algorithm is best implemented in C language and worst implemented in Visual BASIC language. Similarly, Table 3 shows the various PCCM values of depth-limited search algorithm when implemented in C, Pascal and Visual BASIC languages. It is explicitly shown that depth-limited search algorithm has the least PCCM (cwu) when programmed in Pascal language, followed by C language and Visual BASIC language. This means that depth-limited search algorithm is best implemented in Pascal language and worst implemented in Visual BASIC language. Furthermore, Table 4 shows the various PCCM values of A* search algorithm when implemented in C, Pascal and Visual BASIC languages. It is explicitly shown that A* search algorithm has the least PCCM (cwu) when programmed in C language, followed by, Pascal language and Visual BASIC language. This means that A* search algorithm is best implemented in C language and worst implemented in Visual BASIC language.

For the comparative analysis of the software complexity of the selected set of tree search algorithms, the four tree search algorithms had been implemented in C, Pascal and Visual BASIC programming language. Having evaluated the complexity of each implementation using Procedural Cognitive Complexity Measure (PCCM), it was discovered that the complexities of each of the selected tree search algorithm codes are significant in number for comparison since they include different structures. It was also discovered that out of the parameters that determines how complex an implementation can be, such as the number of meaningfully named variables (MNV), arbitrarily named variables (ANV), operators, constants and the cognitive weight unit (CWU) of each line in the code. The number of arbitrarily named variables (ANV) has the highest determining factor, because it's a multiple of four, followed by the cognitive weight unit of each line in the code because it is also a multiplier, while the meaningfully named variables, operators and constants are just being added. The results presented in Tables 1 - 4 showed that all the four tree search algorithms that that were considered were worst implemented in Visual BASIC language. Even from the study of all the implementation of the algorithms, it was discovered that Visual Basic language has the highest number of arbitrarily named variables (ANV), which highly determines the complexity of an algorithm. Breath-first search algorithm and depth-limited search algorithm are best implemented in Pascal language due to the very low number of arbitrarily named variables (ANV). While depth-first search algorithm and A* search algorithm are best implemented in C language because of the low cognitive weight unit of each line in the code.

V. CONCLUSION

In this paper, Procedural Cognitive Complexity Measure had been used to evaluate and compare the complexities of a set of tree search algorithms implemented in different languages in other to know the best programming language for implementing each of the algorithms. The results obtained implied that Pascal programming Language is the best language for implementing breadth-first search algorithm and depth-limited search algorithm. While C language is the best implementation language for implementing depth-first search algorithm and A* search algorithm. Also, it can be inferred that the choice of programming languages affects the complexities of programs of the tree search algorithms.

REFERENCES

[1] Akkanen, J and Nurminen, J.K. (2000): "Case – Study of the Evolution of Routing Algorithms in a Network Planning tool." Journal of System Software, 58: 181-198.

[2] Alfred V.A., John E., and Jeffrey P.U. (1974): "The Design and Analysis of Computer Algorithms", Addison Wesley Publishing Company.

[3] IEEE Standard (1998): Standard for Software Quality Metrics Methodology. Revision IEEE Computer Society 1061-1998.

[4] Li and Henry (1993): ''Object Oriented Metrics that Predict Maintainability'', Journal of Systems and Software, 23(2): 111-122.

[5] Olabiyisi S. O., Omidiora E. O., Isola E. O. (2012). "Performance Evaluation of Procedural Cognitive Complexity Metric and Other Based Complexity Metrics". International Journal of Scientific and Engineering Research, vol 3, issue 9, ISSN 229-5518 IJSER.

[6] Kushwaha, D.S. and Misra, A.K. (2006): Improved Cognitive Information Complexity Measure: A Metric that Establishes Program Comprehension Effort, Software Engineering Notes, 31(5): 206-208.

[7] Misra, S. and Akman, I., (2008): A Complexity Metric based on Cognitive Informatics, Lecture Notes in Computer Science, Vol. 5009, pp.620-627.

[8] Wang, Y. and Shao, J.A,. (2003): New Measure of Software Complexity Based on Cognitive Weights. Can. J. Elec. Computer Engineering, pp. 69-74.

[9] Thomas H. C, Charles E. L, Ronald L.R and Clifford S. (2001): "Introduction to Algorithms", Second Edition. MIT Press and McGraw-Hill. ISBN 0-263-03293-7 Section 22.2: Breadth-First Search, pp.531-539.

[10] Hart, P.E., Nilsson, N.J., Raphael, B. (2008): "A Formal for the Heuristics Determination of Minimum Cost Paths," IEEE Transaction on Systems Science and Cybernetics' SSC4 (2): PP. 100 – 107.

Table 1: Breadth-first search algorithm complexity measures for three different implementation languages

Algorithm	Language	PCCM (cwu)
Breadth-First search	C	534
Breadth-First search	Pascal	460
Breadth-First search	Visual BASIC	550

Table 2: Depth-first search algorithm complexity measures for three different implementation languages

Algorithm	Language	PCCM (cwu)
Depth-First search	C	427
Depth-First search	Pascal	442
Depth-First search	Visual BASIC	549

Table 3: Depth-limited search algorithm complexity measures for three different implementation languages

Algorithm	Language	PCCM (cwu)
Depth-Limited search	C	567
Depth- Limited search	Pascal	563
Depth- Limited search	Visual BASIC	726

Table 4: A* search algorithm complexity measures for three different implementation languages

Algorithm	Language	PCCM (cwu)
A* search	C	886
A* search	Pascal	913
A* search	Visual BASIC	1092

