Hands-on Project
CSC 8350 Advanced Software Engineering

Title: Sequential Cyclomatic Complexity over a Chain of Function Calls

Advisor: Xiaolin Hu

By
Bhavana Pallepati,
Venkata Sarath Kumar Galimelu

Abstract:

McCabe’s Cyclomatic Complexity (CC) is well known software metric which indicates the complexity of a software unit such as a function or an algorithm. It directly measures the number of linearly independent paths through a program's source code. In the existing method of evaluating Cyclomatic Complexity, chain of function calls will not be considered. However, it may not give the true complexity when a function includes other function calls. In this project, we are modifying conventional Cyclomatic Complexity for a chain of function calls. We consider that, this will reflect the actual complexity of the function. Further, we are extending this concept to evaluate the complexity of an entire program.

Introduction:

Definition of Cyclomatic Complexity: It is a software metric which indicates the complexity of a software unit such as a function or an algorithm. It is measured by the number of if, while, do, for, catch, switch, case statements (plus one) in the body of a constructor, method.

It is developed by Thomas McCabe's and Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program. The program with no decision points such as IF, WHILE, or FOR has Cyclomatic Complexity of 1.

Informal Definition: Cyclomatic Complexity =number of decision points + 1

- Cyclomatic Complexity directly measures the number of linearly independent paths through a program's source code.
- A set of paths in a strongly connected graph is **linearly independent** if no path in the set is a linear combination of any other paths in the set or it can be said that any path through the program ("complete path") that introduces at least one new edge that is not included in any other linearly independent paths.
- A path that is subpath of another path is not considered to be a linearly independent path.
- One testing strategy called Basis Path Testing by McCabe who proposed it, is to test each linearly independent path through the program.
- In this case, the number of test cases will equal the Cyclomatic Complexity of a program. The CC of a section of source code is the count of the number of linearly independent paths through the source.
- For instance, if the source code contained no decision points such as IF statements (or) FOR loops, the complexity would be 1, since there is only a single path through the code. If the code had a single IF statement containing a single condition there would be two paths through the code, one path where the IF statement is evaluated as true and one path where the IF statement is evaluated as FALSE.

CC for function calls can be calculated by using the following formula

CC=number of condition paths+1

CC is computed using the control flow graph of the program, the nodes of graph corresponds to individual group of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. It may also be applied to individual functions, modules, methods, and classes within a program.

But the calculation of CC for a function (f), is not considering the complexity of the functions which are called from f. In this project, we propose a new metric called Sequential Cyclomatic Complexity (SCC) to consider the chain of function calls to get the true complexity of the function.

Significance:

- Cyclomatic Complexity is related to number of coding errors in estimating the cost of a program.
- Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand.
- Helpful in Risk Prediction.
- Important in limiting the complexity during development.
- It checks the linearly independent path through the program which means test case will be equivalent to the Cyclomatic complexity of the program.
- Complexity of a unit is used to estimate the effort or cost of a program and is a key factor in project scheduling as an under estimated module may consume more time.

Following are the guidelines to interpret structure of the based on value of Cyclomatic Complexity:

- If value of Cyclomatic Complexity is more than 10, the structure of the module is overly complex. Thus ask the developer to re-construct the module.
- If value of Cyclomatic Complexity is more than 5 and less than 10, the structure of the module is complex indicating that the logic is difficult to test. Thus such a module shall be allocated to experience testers.
- If value of Cyclomatic Complexity is less than 5, the structure of the module is simple indicating that the logic is easy to test. Thus such a module may be allocated to less experienced testers.

Mathematical Representation:

Mathematically, it is set of independent paths through the graph diagram. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module. Control Flow depicts the flow of a program with nodes and Edges.

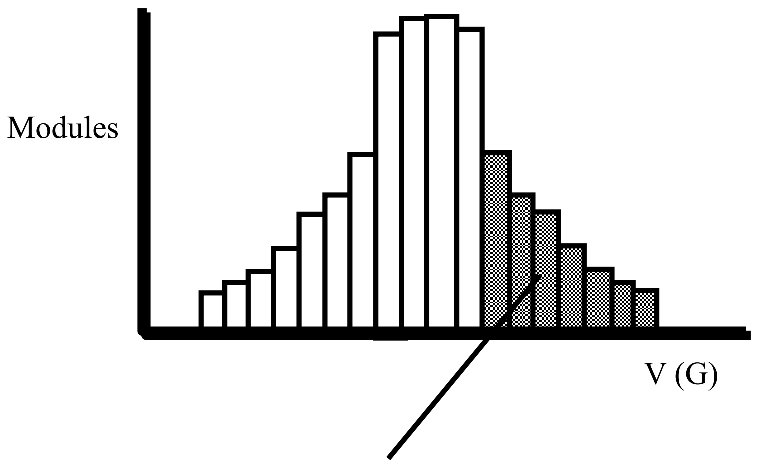
Cyclomatic Complexity V (G) = E - N + P where

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points (node that contains condition)

Graph Complexity (Cyclomatic Complexity):



Modules in this range are more error prone

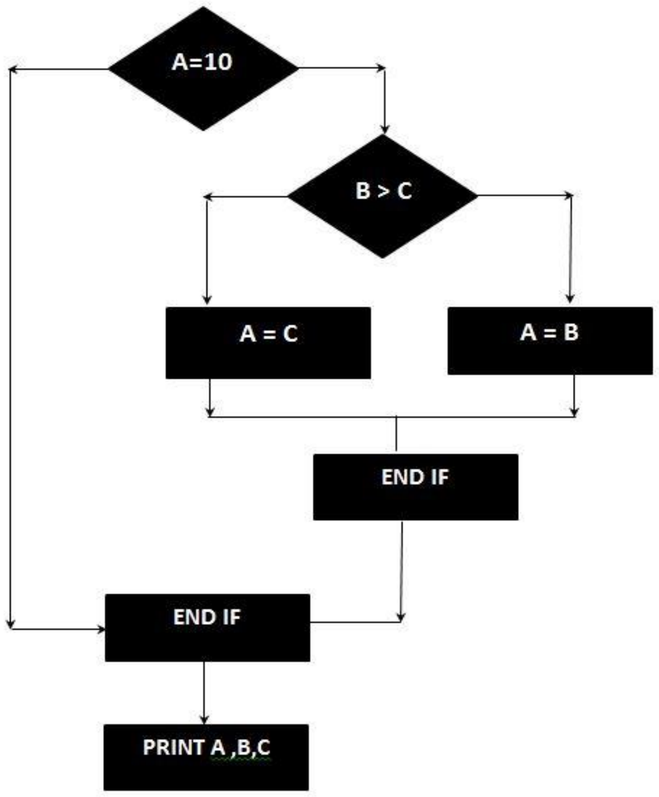
Example

```
if a = 10 then
  if b > c then
    a = b
  else
    a = c
 endif
endif
```

print a

print b

print c



Here the number of control flow is 8, number of node is 7 and the number of condition boxe is 2.
Cyclomatic Complexity V (G) = 8 - 7 + 2 = 3.

Tools for Calculating CC:

Different commercial tools were developed for calculating McCabe's Cyclomatic Complexity.
CCM is a tool that analyzes C, C++, and C #and javascript code and produces a list of the most complex methods or functions in the analyzed code base

NDepend is a static analysis tool for .NET managed code.

CheckStyle plug-in for eclipse provides checking of code layout issues, and also provides checks that find class design problems, duplicate code, or bug patterns like double checked locking.

devMetrics is a C# code analysis tool that gathers software metrics , so that developers, leads and software managers can quickly identify potentially problematic and high-risk areas of their .NET code.

All these tools are not considering the chain of function calls in calculation of CC.

Proposed System: In this project we are extending this concept to evaluate the complexity of an entire program. It is a new metric called Sequential Cyclomatic Complexity (SCC) to consider the chain of function calls inside a function to get the true complexity of the function.

Sequential Cyclomatic Complexity over a Chain of function calls:

We propose Sequential Cyclomatic Complexity (SCC), which considers chain of function calls while calculating the Cyclomatic complexity. SCC assumes CC of the function and CC of functions which are called from it. So,

SCC (fn) = CC(fn$_{cg}$) + $\sum_{i=1}^{n} \boldsymbol{CC(fn_{cd}(i))}$

Sequential Cyclomatic Complexity of the program is equal to sum of Sequential Cyclomatic Complexity of all the functions of the program.

SCC (program) = $\sum_{j=1}^{m} \boldsymbol{SCC(fn(j))}$

where m is the number of functions.

Example: Sequential Cyclomatic Complexity over a Chain of function calls

For example, consider the following C program fragment.

```
void fun1()
{
 if(condition)
 fun2();
 else
 fun3();
}
void fun2()
{
 printf("In fun2");
}
void fun3()
{
 printf("In fun3");
}
void main()
{
 fun1();
}
```

As per the McCabe's approach, the Cyclomatic Complexity = number of predicate nodes +1.
So, the Cyclomatic Complexity of
main is 1, fun1 is 2, fun2 is 1, fun3 is 1.
But the function fun1 is calling fun2 and fun3, and main is calling fun1
So, Sequential Cyclomatic Complexity is calculated over chain of function calls.
fun3 is 1, fun2 is 1,
fun1 is (CC(fun1) + CC(fun(2) + CC(fun(3)) = 2+1+1=4, and
main is CC(main) + CC(fun1) = 1+ 4 = 5.
Now, the Sequential Cyclomatic Complexity of the entire program is 11.

Implementation

We have developed a .NET application for the implementation of this Project using C# language. We have developed this project for calculating the complexity of the C Programs as of now. Generally the Source code is written in four files and the approximate the number of lines of Code is 900.

We have used the Microsoft Access Database to store the details of the C program for evaluating the Sequential Cyclomatic Complexity and created four tables named Keywords, funcalls, complexity and chaincomplexity.

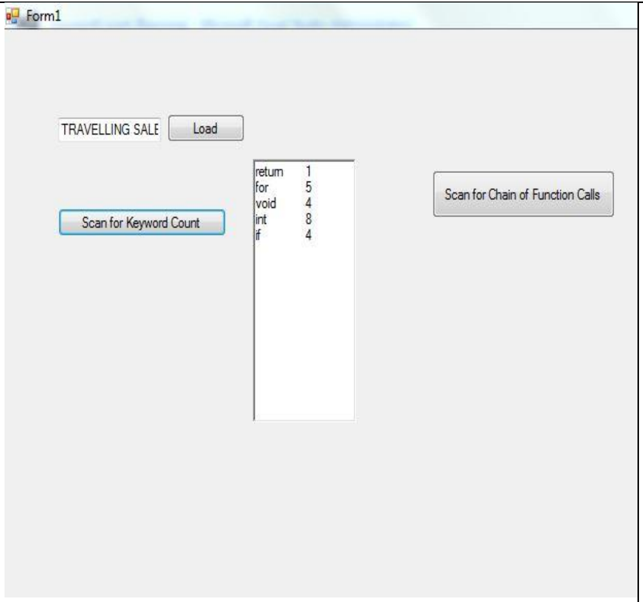
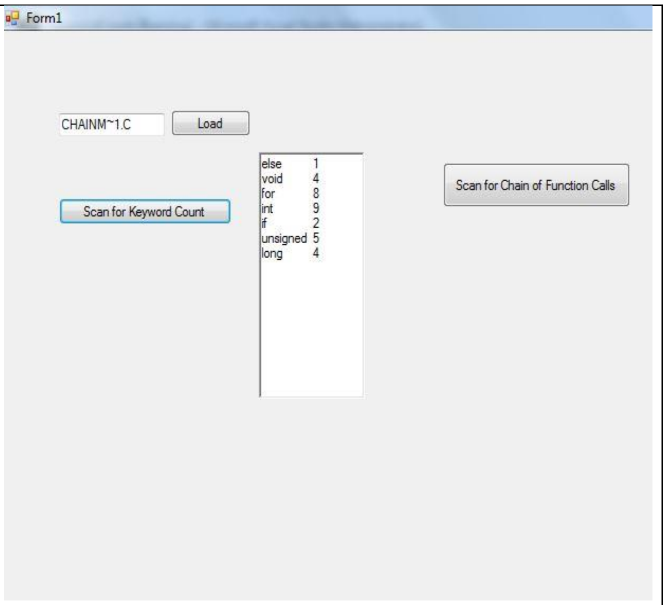
The Program is written such that all the keywords, built in functions, header files and library files of the C language are stored in a separate text files and each time a C program is loaded to evaluate the complexity, all these files are also loaded to check with the C program the count of all the keywords or decision points present in the program i.e. number of FOR loops, IF loops, WHILE etc. All these Keyword count information is stored in the keywords table.

This will help in calculating the Cyclomatic Complexity of the Program.

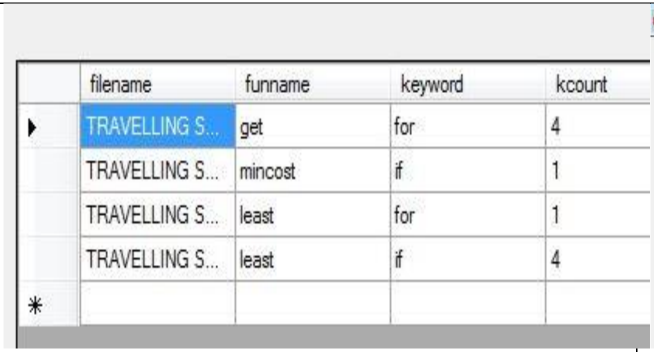
Moreover we identify the User defined functions and the function calls within a function and all these are stored in another table named funcalls table used for identifying the number of function calls within another function.
Then we have the complexity and chaincomplexity tables which stores the information of functions and the corresponding complexity and sequential complexity over function calls respectively.

Experimental Results

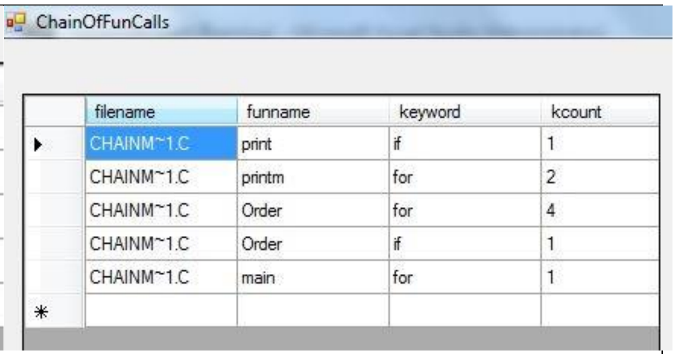
Here let us see the Sequential Cyclomatic Complexity of the Travelling Salesman Problem and Chain of Matrix Multiplication program with no function calls. The complexity of a simple program will have the complexity and sequential complexity the same. Hence the Complexity increases if the function call is considered.

 Form1 TRAVELLING SALE Load Scan for Keyword Count Scan for Chain of Function Calls return 1 for 5 void 4 int 8 if 4	 Form1 CHAINM~1.C Load Scan for Keyword Count Scan for Chain of Function Calls else 1 void 4 for 8 int 9 if 2 unsigned 5 long 4
Travelling Salesman Problem	Matrix Multiplication program

Form1 loads the file and calculates the total keyword count int the program.



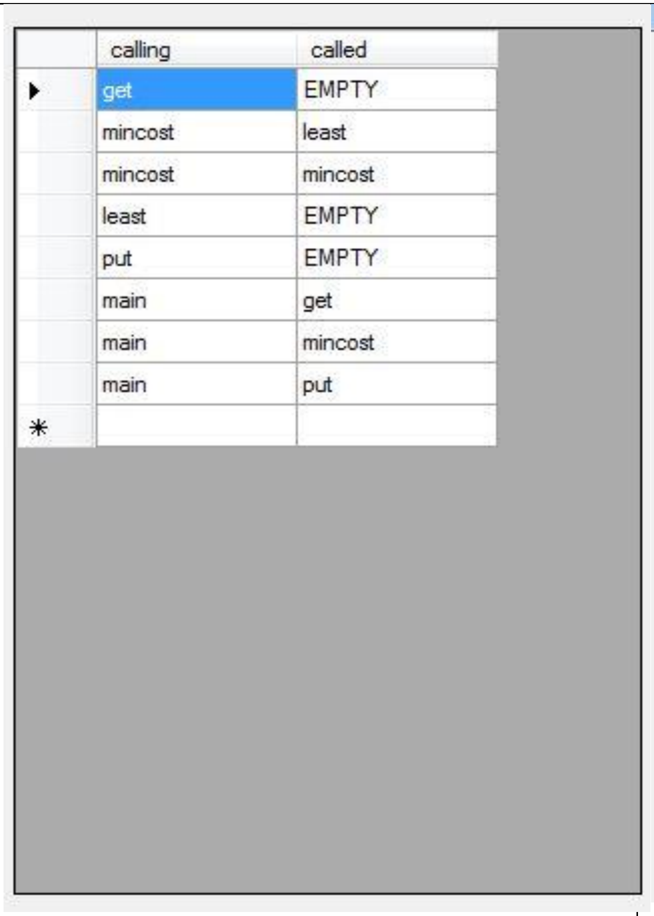
filename	funname	keyword	kcount
TRAVELLING S...	get	for	4
TRAVELLING S...	mincost	if	1
TRAVELLING S...	least	for	1
TRAVELLING S...	least	if	4



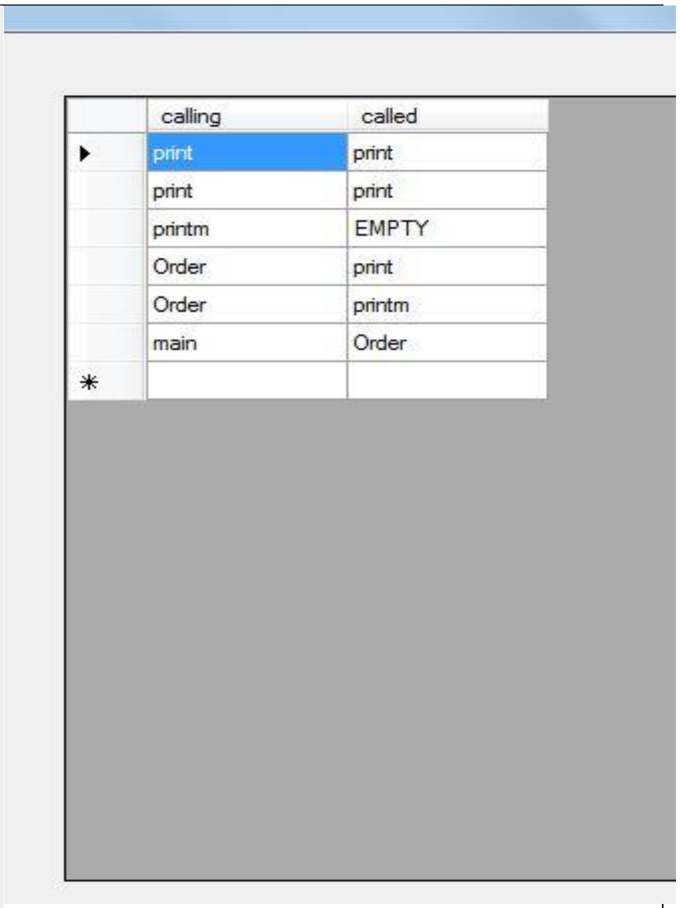
filename	funname	keyword	kcount
CHAINM~1.C	print	if	1
CHAINM~1.C	printm	for	2
CHAINM~1.C	Order	for	4
CHAINM~1.C	Order	if	1
CHAINM~1.C	main	for	1

Travelling Salesman Problem	Matrix Multiplication program

Calculates the keyword count for each function in the program.



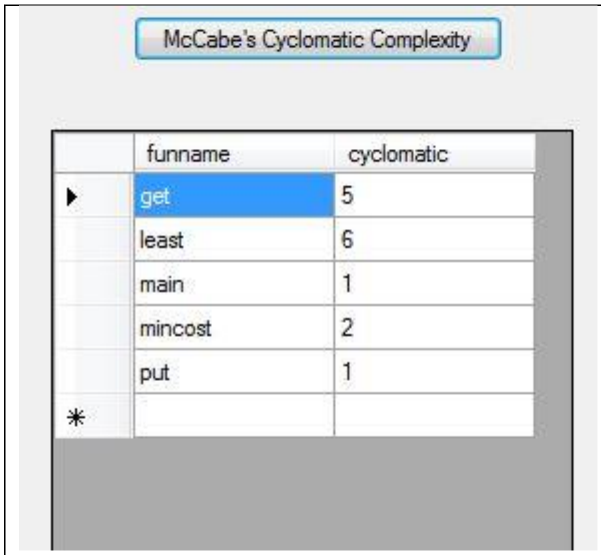
calling	called
get	EMPTY
mincost	least
mincost	mincost
least	EMPTY
put	EMPTY
main	get
main	mincost
main	put



calling	called
print	print
print	print
printm	EMPTY
Order	print
Order	printm
main	Order

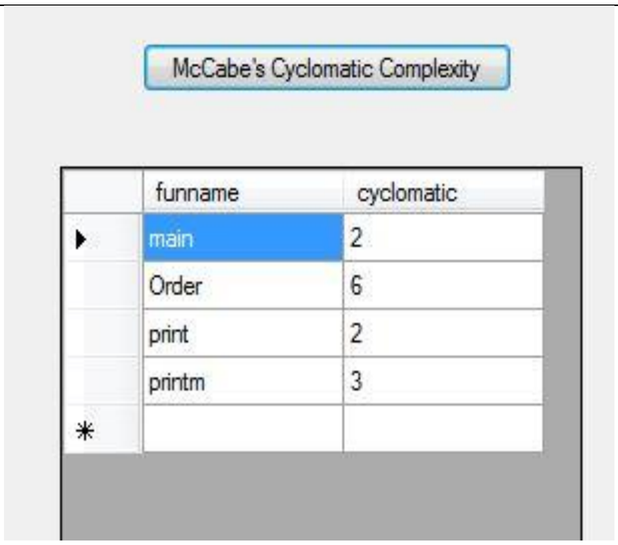
Travelling Salesman Problem	Matrix Multiplication program

Calling and Called Functions(EMPTY if there is no Function called)



funname	cyclomatic
get	5
least	6
main	1
mincost	2
put	1

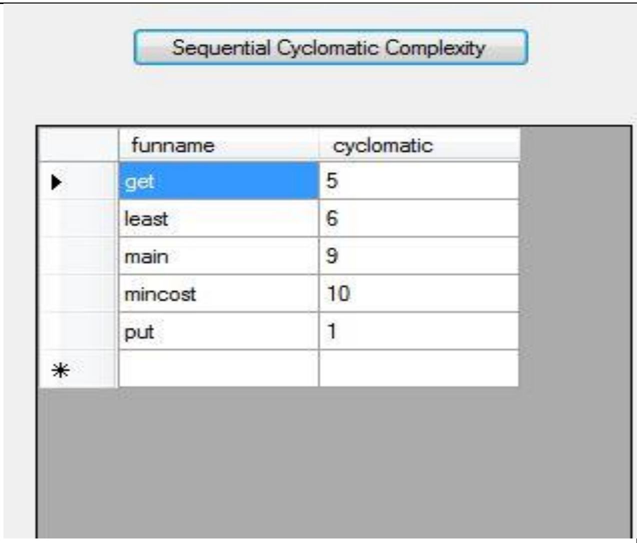
Travelling Salesman Problem



funname	cyclomatic
main	2
Order	6
print	2
printm	3

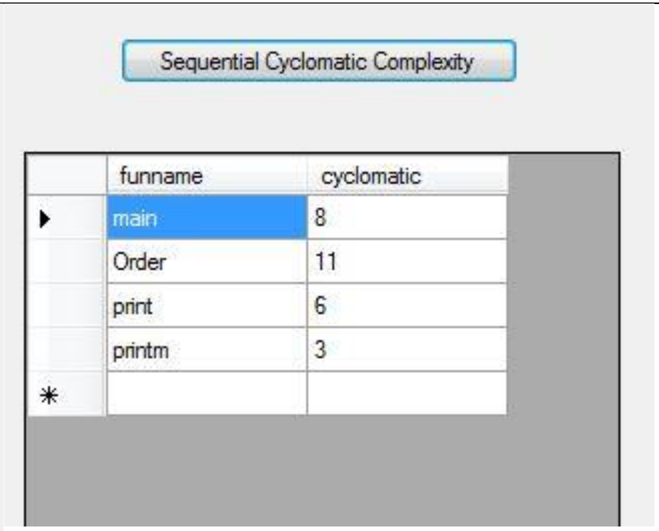
Matrix Multiplication program

Cyclomatic Complexity of each of the functions based on the number of calls



funname	cyclomatic
get	5
least	6
main	9
mincost	10
put	1

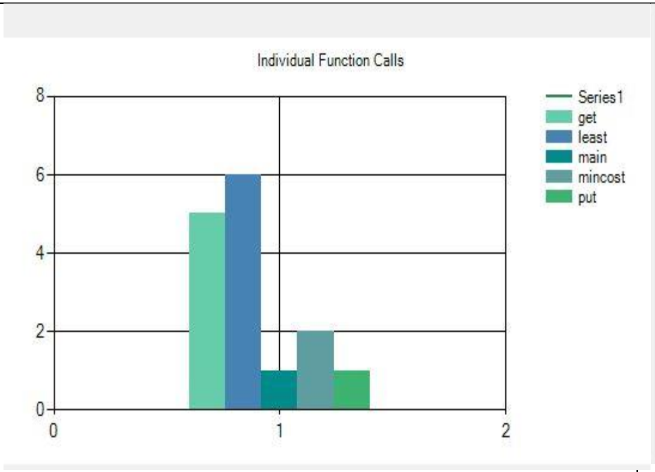
Travelling Salesman Problem



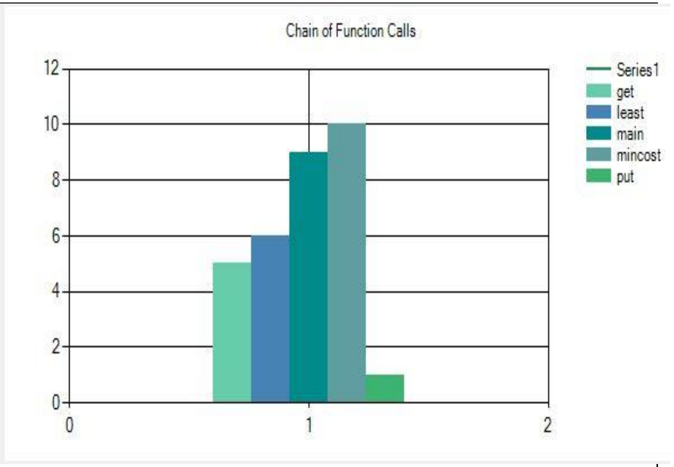
funname	cyclomatic
main	8
Order	11
print	6
printm	3

Matrix Multiplication program

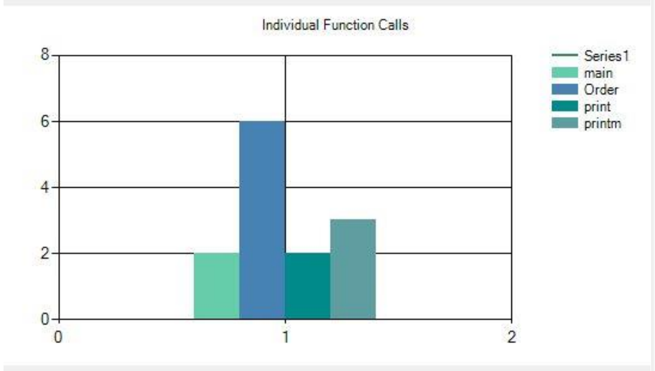
Now we have also depicted the graphical representation of complexity of the Functions over MacCab'es Cyclomatic Complexity and Sequential Cyclomatic Complexity over a chain of function calls using this proposed approach.



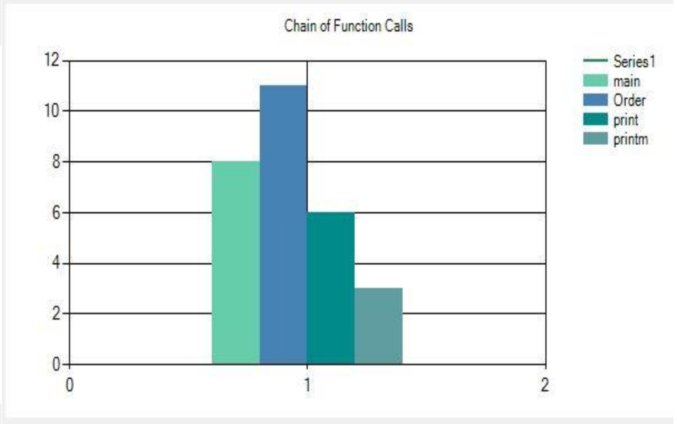
Travelling Salesman Problem Individual Function Calls



Travelling Salesman Problem Chain of Function Calls



Matrix Multiplication program Individual Function Calls



Matrix Multiplication program Chain of Function Calls

Results: The results shows that the calculation of cyclomatic complexity by considering chain of function calls gives true complexity measure of the program. Comparison between sequential cyclomatic complexity and cyclomatic complexity for chain of function calls is shown in the above figures.

Conclusion and Future Scope:

Our experimental results reflects the true cyclomatic complexity of the program by considering the chain of function calls. In future we want to extend this work for object-oriented programming languages.

Bibliography

[1]Kiran Kumar B,Jayadev Gyani and Narsimha G,"Sequential Cyclomatic Complexity Over a Chain of Function Calls",2012 International Conference on Networks and Information (ICNI 2012).

[2] http://pdepend.org/documentation/software-metrics/cyclomatic-complexity.html

[3] http://docs.klocwork.com/Insight-10.0/McCabe_Cyclomatic_Complexity

[4] http://www.guru99.com/cyclomatic-complexity.html