01 Apr 1981

A Topological Analysis Of The Complexity Of Computer Programs With Less Than Three Binary Branches

Warren Harrison

Kenneth I. Magel
Missouri University of Science and Technology

Recommended Citation

W. Harrison and K. I. Magel, "A Topological Analysis Of The Complexity Of Computer Programs With Less Than Three Binary Branches," *ACM SIGPLAN Notices*, vol. 16, no. 4, pp. 51 - 63, Association for Computing Machinery (ACM), Apr 1981.
The definitive version is available at https://doi.org/10.1145/988131.988137

A TOPOLOGICAL ANALYSIS OF THE COMPLEXITY OF COMPUTER PROGRAMS WITH LESS THAN THREE BINARY BRANCHES

Warren Harrison[1]
Kenneth Magel[1]
University of Missouri-Rolla

Introduction

Katzan [4] defines the complexity of a system as referring to the number of relations or connections among components of the system. Most would agree that it is an understatement to say that computer programs are complex entities. Indeed, they are so complex that at times one marvels at the fact that they are even developed. Even more important than their original development however, is the fact that they must often be modified - and therefore comprehended over and over again - by people, most likely not the original developer, during the debugging and maintenance phases.

It has long been suspected that the complexity of a large piece of software is a function of the density of transfers of control within the program. Numerous techniques have been developed to either directly or indirectly evaluate this density, and thereby measure the complexity of the software. The underlying concept behind almost all these techniques is an analysis of the topology of the software's flow of control.

The flow of control is usually modeled by dividing the program up into sections with a single exit, a single entry, and no internal transfers of control. These one-in, one-out sections are referred to as basic blocks. The basic blocks are then used in a directed graph, $G=(V,E)$, where V is the set of basic blocks represented as nodes in the graph, and E is a set of edges connecting these nodes and corresponding to the flow of control in the program. The topology of a program is easily inspected when such a graph is constructed to represent it.

The Cyclomatic Complexity Model

The Cyclomatic Complexity Model, introduced by McCabe [5] is one of the most widely accepted topological measures in use today. McCabe's model evaluates the number of basic paths, that when used together, may generate every possible path in the program.

This is accomplished by counting the number of nodes and edges in the directed graph which represents the program. These are then used to calculate the Cyclomatic Complexity Measure,

[1]Work supported in part by NSF Grant MCS 8002667.

V(G) by way of the following formula:

$$V(G)=e-n+2p$$

where e is the number of edges, n is the number of nodes, and p is the number of connected components in the graph. Normally, p has a value of 1.

The Cyclomatic Complexity Measure may also be computed for structured programs by adding 1 to the number of decisions in the program. This allows the measure of complexity to be arrived upon by simply inspecting the program.

Several refinements to McCabe's model have been suggested, including those by Myers [6] and Hansen [2]. The majority of these modifications entail the use of an interval rather than a single number so that some other property can be measured in addition to the cyclomatic complexity. However, the Cyclomatic Complexity Measure by itself requires only a directed graph of the program, and allows the actual content of the nodes to be ignored.

The Scope Complexity Measure

The Scope Complexity Measure [3] is a fairly new method. Like the other topological techniques, it focuses upon the control path as represented by the directed graph. However, rather than simply counting the number of paths that exist, the Scope Measure derives the program complexity from analyzing the scope of control of the control structures.

In the Scope Measure, each node in the control graph has a particular complexity assigned to it depending on the complexity of the statements within the node. This complexity may be computed in several ways, including the use of Halstead's Software Science measure [1]. The complexity of the node is referred to as the node's raw complexity.

In this paper we shall use a value of 1 for each node's raw complexity since McCabe's method fails to take into account the idea that "all nodes are not created equal". This will introduce a certain amount of equivalence between the two methods.

The Scope Measure requires that the flow of control terminate at one common node physically located after all other nodes in the graph. This node may be thought of as being analogous to the FORTRAN "END" statement. This should be the only node in the graph with an outdegree of zero. The raw complexity of this node shall be zero, and therefore will have no direct effect upon the overall complexity of the graph. We shall label this node ε.

If, in our graph, we have an edge from a node x to a node y, the node x is said to immediately precede the node y, and the node y is said to immediately succeed node x. If there is a path from node x to node y, then node x is said to precede node y, and node y is said to succeed node x.

Every node in the graph is one of two types. It is either a selection node, which means it has an outdegree of 2 or more, or it is a receiving node with an outdegree of 1.

In order to obtain the Scope Measure, we create a subgraph consisting of all nodes which immediately succeed an individual selection node. In other words, all those nodes which are connected by a single edge from the selection node.

Each such subgraph has at least one node within the graph, G that is its lower bound. A lower bound is a node which is on a path from every node in the subgraph. That is, it succeeds every node which immediately succeeds the selection node. Most such subgraphs will have numerous lower bounds. The lower bound which precedes every other lower bound of that subgraph is called the greatest lower bound.

The raw complexity of every node on the paths leading to (but not including) the greatest lower bound of the subgraph is summed. Note that in most cases, the raw complexity of the nodes within the subgraph are also included in the sum. This sum is then added to the raw complexity of the selection node to give that node's adjusted complexity.

This process is repeated for every selection node within the control graph. The adjusted complexity of the receiving node is simply the node's raw complexity. The adjusted complexities of all the nodes are then summed, and that sum the, Scope Number, is the complexity of the program as a whole.

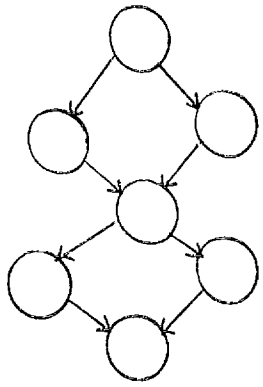
The Scope Number itself, however, can be misleading when the raw complexities do not reflect the actual complexities of the individual nodes. This is due to the fact that certain arrangements of the control structures can reduce the number of basic blocks within the program. When using the usual Scope Number where the raw complexities reflect the complexities of the nodes, this is compensated for. When we are simply assigning every node a raw complexity of 1 as we are doing in this paper, the actual complexity of the program can be distorted. For this reason, we introduce the Scope Ratio. This is the ratio of the number of nodes in the program to the Scope Number. Note that we only count those nodes with a raw complexity of 1 or more.

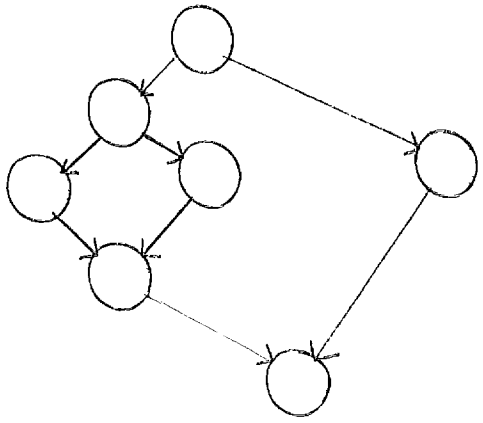
A Scope Ratio of 1 indicates the simplest program possible. This ratio reflects that the control structure configuration has not added any additional complexity to the program. As the

magnitude of the Scope Ratio decreases, the complexity of the program increases. For example, a program with a scope Ratio of less than .4 is fairly complex.

Programs With Small Numbers of Branches

Two different programs can consist of exactly the same statements, yet have their control structures arranged differently (i.e., placed in different places). This can radically effect the complexity of the programs. For example, the following two graphs represent two programs which may have each node containing the same statements, yet by virtue of their placement within the program, have complexities which differ.



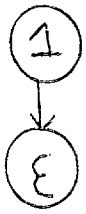


We may find it useful to use the McCabe and Scope Complexity Measures to analyze certain classes of control structure arrangements. This may help us to identify properties of certain arrangements which make programs inherently more complex than alternative arrangements.

We can identify the members of some of the subsets of the class of all control structure arrangements. We may identify these members through the number of transfers of control, or branches in them. Obviously as the number of branches increase, the number of members in the subset increases exponentially. For this reason, we shall limit the subclass which we shall analyze to at most two branches.

Programs with Zero Branches

This subset includes as members, all programs which have no conditional transfers of control. The analysis of the programs in this class is obviously trivial, and the outcome borders upon the intuitive. All programs within this subset can be represented by the following directed graph:



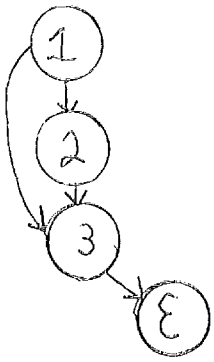
(a)

Both the Scope Ratio and the Cyclomatic Complexity Number for every element within this subset is one. It may be said without any fear of contradiction that this subset is the most inherently simple of all possible configurations.

Programs With One Branch

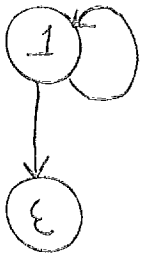
This subset includes as members all programs which have one conditional transfer of control. We may partition the members of this subset into two subclasses. One subclass has as members all those programs with one forward branch. The other subset has as members all programs with a single backward branch.

The members of the first subset all have control graphs similar to the following:

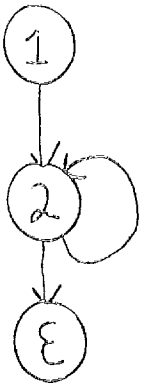


(b)

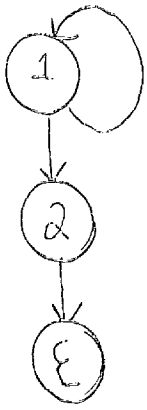
Every member of the second subset can be represented by one of the following control graphs, depending upon the arrangement of the control structures:



(c)



(d)



(e)

We shall analyze the complexity of these various arrangements using the McCabe and Scope complexity measures. The McCabe Cyclomatic Complexity Number, since it only counts paths, will give the same measurements for all of these control graphs. For programs with only one branch, V(G)=2.

The Scope Measure however, differs greatly among the various configurations. The Scope Ratio for programs with one forward branch is 3/4, or .75.

The Scope Ratio for the control graphs representing programs with one backward branch is 1/2, or .50; 2/3 or .66; and 2/3 or .66; for c, d and e respectively.

In terms of complexity, the Scope Ratio ranks the programs in the following order:

CONTROL GRAPH	SCOPE NUMBER	SCOPE RATIO	V(G)
b	4	.75	2
d	3	.66	2
e	3	.66	2
c	2	.50	2

The complexity of each graph can be better understood if we examine what the graphs actually represent. Graph b, for instance, represents a program with a single if-then type of control structure. This is normally a fairly simple control structure, as evidenced by a Scope Ratio of .75.

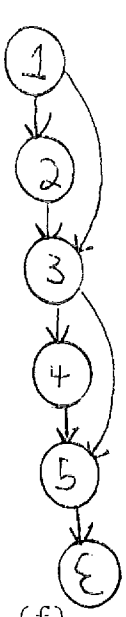
Control graphs c, d and e are quite a different story. Any time we have a backward branch in a program, we can conceivably have an infinite number of paths which may be executed. The Scope Ratio for these configurations demonstrate the increased complexity over a single if-then structure such as b.

Control graph c illustrates a program whose execution may be repeated in its entirety. The complexity of this situation is fairly clear. The Scope Ratio for d and e reflect the decreased complexity generated when only a fraction of the program may be repeatedly executed. Note that we are assuming roughly equivalent program length. This would indicate that less of the program would be located within the range of the loop, and therefore, less information is to be absorbed concerning the comprehension of the loop in d and e than in c.

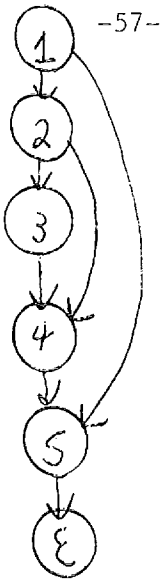
Programs With Two Branches

Up till now we could illustrate a small number of control graphs to represent every program possible within the subclass. When we increase the number of branches, we increase the number of possible control graphs greatly. In this case, we may partiton the subset into three distinct classes. We may have a program with either two forward branches, two backward branches, or one forward branch and one backward branch.

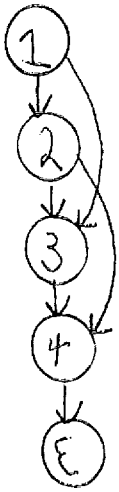
The following graphs represent some possible configurations available for programs with two forward branches:



(f)

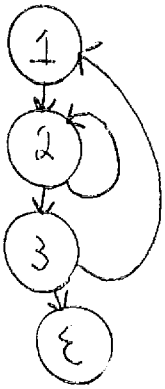


(g)

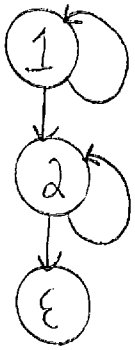


(h)

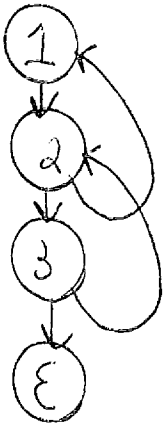
Programs with backward branches allow a few more arrangements than are available with forward branches. These are illustrated below:



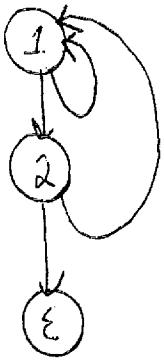
(i)



(j)

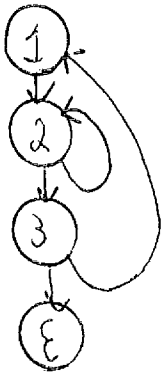


(k)

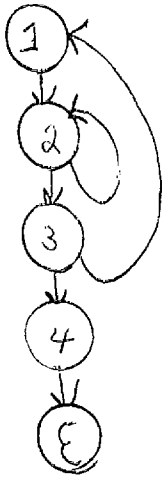


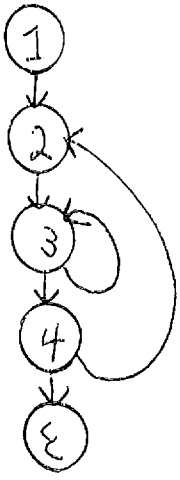
(l)

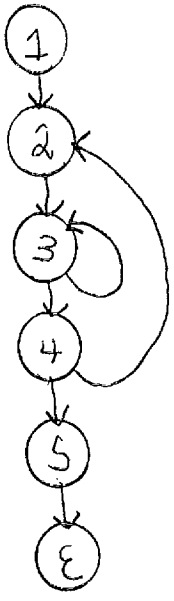
Obviously these are not all possible control graphs with two backward branches. Numerous others may be formed by extending one of the illustrated graphs in the following manner:



(;)

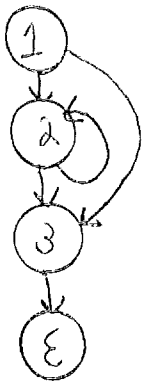




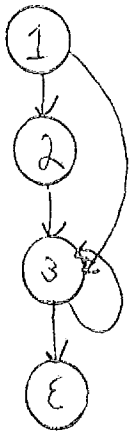


These extensions to the original graphs would result in slightly less complex programs for the reasons mentioned in our discussion of one-branch programs. While we do not challenge the legitimacy of these as unique control structure configurations, we feel that the complexity of these control graphs may be easily inferred from our analysis of their basis graphs.

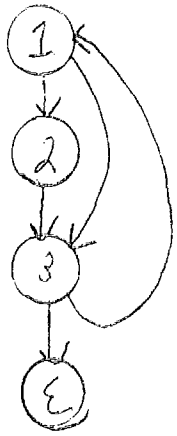
As we mentioned earlier, we also may have a subset of two-branch programs which have transfers of control going in both directions. Obviously there are a great many such control graphs representing these programs. Some of them are presented below:



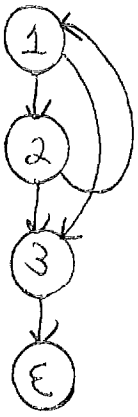
(m)



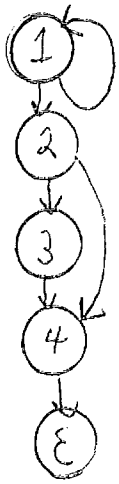
(n)



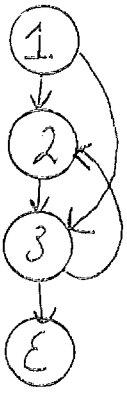
(o)



(p)



(q)



(r)

Because each program contains two branches, the McCabe metric will yield a V(G) of 3 for each control graph. The following table arranged in order of Scope Ratio complexity summarizes the complexity of the various arrangements:

CONTROL GRAPH	SCOPE NUMBER	SCOPE RATIO	V(G)
f	7	.71	3
q	6	.66	3
m	5	.60	3
n	5	.60	3
h	7	.57	3
g	9	.56	3
j	4	.50	3
p	6	.50	3
r	6	.50	3
i	7	.43	3
o	7	.43	3
1	6	.40	3
k	8	.38	3

We illustrate the various configurations in order of complexity in Appendix A. Certain properties may be observed as the complexity of the arrangements increases.

The least complex configuration (f) is simply a sequence of if then structures. Configuration q is a do loop followed by an if then structure. With a Scope Ratio of .66, it is only slightly more complex than f.

The next level of complexity, a Scope Ratio of .60 is shared by both m and n. These control graphs illustrate a do-loop with an if then structure and a do loop following an if then structure.

The next two configurations, h and g are so close as to almost be the same complexity with Scope Ratios of .57 and .56. They illustrate, respectively, an if then structure, with another if structure nested within it which may exit the entire structure; and a normal nested pair of if then structures.

The next three configurations all have Scope Ratios of .50 and represent: a pair of do loops in sequence; a do loop with a branch out of the loop; and a do loop with a branch into it. These are of course j, p and r.

Configurations i and o both have Scope Ratios of .43. They represent a pair of nested do loops; and a do loop with an if then structure nested within it.

Configuration 1 has a Scope Ratio of .40 and illustrates a pair of nested loops. The abnormal complexity reflects the fact that they both have common entry points.

The most complex configuration, k has a Scope Ratio of .38 and illustrates two overlapping loops.

Appendix B illustrates programs which correspond with the control graphs listed in order of complexity. It is hoped that an opportunity to analyze programs representing the control graphs will help illustrate the properties which increase complexity.

Conclusions

Two programs may be of the same length and possess equivalent properties in all respects except for the control structure configuration. We have illustrated the variations in complexity which may arise from such situations by using two topological measures, viz., McCabe's Cyclomatic Complexity Number and the Scope Complexity Ratio. The Scope Measure is able to distinguish among programs which the cyclomatic number measure considers to be equally complex.

References

1. Halstead, M., Elements of Software Science, Elsevier North-Holland, Inc., New York, 1977.

2. Hansen, W., "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", ACM SIGPLAN Notices Vol. 13, No. 3 (Mar 1978), pp 29-33.

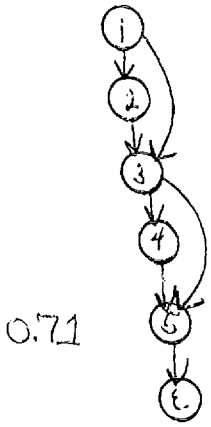
3. Harrison, W. and Magel, K., "A Complexity Measure Based On Nesting Level", ACM SIGPLAN Notices (to appear).

4. Katzan, H., Systems Design and Documentation, Van Nostrand Reinhold Co., New York, 1976.

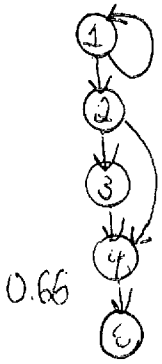
5. McCabe, T., "A Complexity Measure", IEEE Transactions On Software Engineering, Vol SE-2, No. 4, (Dec 1976), pp 308-320.

6. Myers, G., "An Extension to the Cyclomatic Measure of Program Complexity", ACM SIGPLAN Notices, Vol. 12, No. 10 (Oct 1977), pp 61-64.

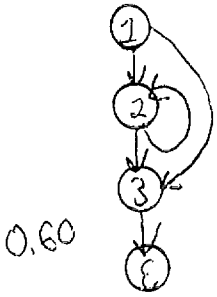
Appendix A



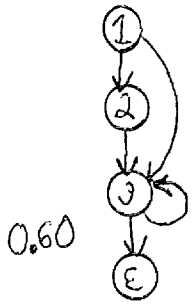
(f)



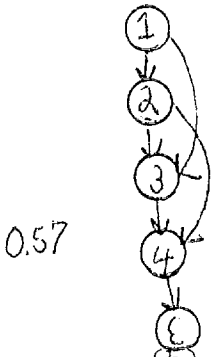
(q)



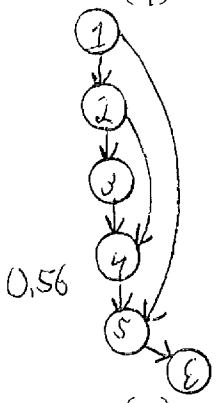
(m)



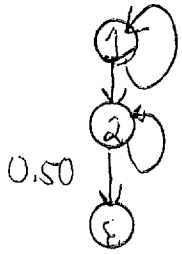
(n)



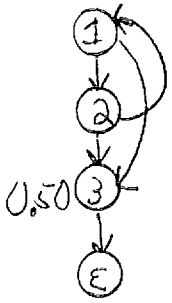
(h)



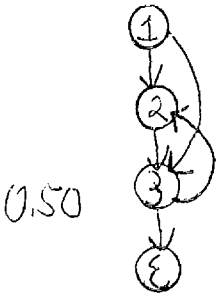
(g)



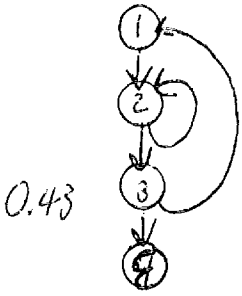
(j)



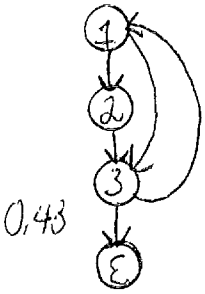
(p)



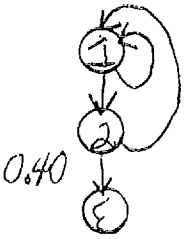
(r)



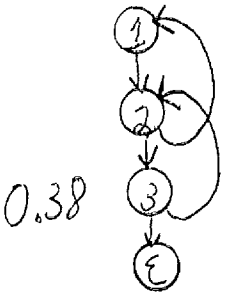
(i)



(o)



(l)



(k)

Appendix B

```
S1;
if P1 then S2 fi
S3;
if P3 then S4 fi
S5;
end;
```

(f)

```
do until (P1)
S1;
od
S2;
if P2 then S3 fi
S4;
end;
```

(q)

```
S1;
if P1 then do
  until (P2)
  S2;
  od
fi
S3;
end;
```

(m)

```
S1;
if P1 then S2 fi
do until (P3)
   S3;
od
end;
```

(n)

```
   S1;
   if P1 then
         S2;
         if P2 then go to L1 fi
   fi
   S3;
L1:S4;
   end;
```

(h)

```
S1;
if P1 then
     S2;
     if P2 then
          S3;
     fi
     S4;
fi
S5;
end;
```

(g)

```
do until (P1)
   S1;
od
do until (P2)
   S2;
od
end;
```

(j)

```
do until (P1)
   S1;
   if (P1') then go t
        L1;
   S2;
od
L1:S3;
end;
```

(p)

```
   S1;
   if P1 then go to L1;
   do until (P2)
        S2;
L1:     S3;
   od;
   end;
```

(r)

```
do until (P1)
   S1;
   do until (P2);
      S2;
   od
   S3;
od
end;
```

(i)

```
do until (P1)
   S1;
   if (P1') then
           S2;
   fi
   S3;
od
```

(o)

```
L1:  S1;
     if P1 go to L1;
     S2;
     if P2 go to L1;
     end;
```

(l)

```
L1:  S1;
L2:  S2;
     if P2 then go to L3;
     S3;
     if P3 then go to L2;
     end;
```

(k)

Note that as the complexity increases, it begins to get harder and harder to represent the flow of control with predefined structure such as do until, etc.