Software metrics for aiding program development and debugging

by N. F. SCHNEIDEWIND
Naval Postgraduate School
Monterey, California

INTRODUCTION

Computer program graphs have proven very useful because they illuminate the structural characteristics of a program. Structural characteristics, as a representation of program complexity, have been shown to be strongly related to program development time, program quality and difficulty of debugging. The use of graphs for these purposes is not widely known or understood in the data processing community. It is the aim of this paper to provide an introduction to graphs as they apply to program representation and to show examples of their use in program design and debugging.

GRAPH DEFINITIONS AND PROPERTIES

DEFINITIONS

The following definitions are due to Chan:

Graph

A graph is a set of line segments called edges (e_j) and points called vertices (v_i) which are the end-points of the edges, interconnecting in such a way that the edges are connected only to the vertices. A non-directed graph has no orientation of the edges; a directed graph does have edge orientation in the form of arrows. Figure 1a shows a directed graph G.

Degree of a vertex

The degree of a vertex is the number of edges incident with that vertex. The degree of v_6 of G is 3.

Sub-graph

A portion of a graph, containing a subset of the edges and vertices of the graph is called a sub-graph. Two sub-graphs of G, G_1 and G_2, are shown in Figure 1b.

Path

If a set of edges e_1, e_2, . . . , e_i can be ordered in the form e_1(v_1, v_2), e_2(v_2, v_3), . . . , e_i(v_i, v_i+1), where v_1 and v_i+1 are the terminal vertices and all vertices are distinct, then the set of edges forms a path. In G, the set of edges a, d, e forms a path. It is important to note that by this definition vertices may not be revisited. Thus, the sequence of edges a, b, c, d in G is not a path because v_2 appears twice in the edge sequence. In graph theory this sequence is called a walk. However, since iteration is an important characteristic of computer programs, we will modify the above definition of path to include walks in order to avoid using two terms when describing a program graph. When the edges of a path have consistent orientations, the path is directed. The above paths are directed.

Circuits

If the two terminal vertices of a path coincide and the remaining vertices are distinct, this path is a circuit. A directed circuit has all edges with the same (clockwise or counter-clockwise) orientation. Thus G_1 and G_2 are circuits but only G_1 is directed.

Connected graph

A graph is connected if there is at least one path between every pair of vertices. G, G_1 and G_2 are connected.

Tree

A tree T of a connected graph is a connected sub-graph that contains all vertices of the graph but no circuits. The edges contained in a tree are called branches. The complement set of edges T'; that is the remaining edges of the graph, are called chords. One of the trees of G is shown in Figure 1c, where T=(a, b, d, e, f, g, h, i, k) and T'=(c,
Figure 1a—Graph G.

Figure 1b—Sub-graphs $G_1$ and $G_2$.

MATRICES PROPERTIES

**Adjacency matrix**

The adjacency matrix $X=[x_{ij}]$ of a directed graph with $V$ vertices is a $V \times V$ matrix consisting of 0, 1 elements, where $x_{ij}=1$ if there is an edge directed from $v_i$ to $v_j$, and 0 otherwise. The $r$th power of the adjacency matrix $X^r$ is equal to the number of directed paths of length $r$ edges between each pair of vertices $v_i$ and $v_j$. Shown in Figures 2a, 2b and 2c are $X$, $X^2$ and $X^3$ of $G$, respectively. Thus Figure 2b shows that there are two paths of length 2 from $v_6$ to $v_9$; these consist of edges $g$ and $i$ and $h$ and $j$ (see Figure 1a). A directed path of length 3 from $v_1$ to $v_9$ is indicated in Figure 2c; this is a path from start to terminal vertices consisting of edges $a$, $d$ and $l$. If $v_i$ designates the terminal node of a graph and recognizing that the maximum possible path length is $r=E$ edges, the matrices $X$, ..., $X^E$ with non-zero entries in the $1\times t$ cells will enumerate all of the paths which start at $v_1$ and terminate at $v_t$.

**Fundamental circuit matrix**

A fundamental circuit matrix, with respect to a tree $T$ of a graph $G$ of $V$ vertices and $E$ edges, is the matrix $B_r$ of
order $(E-V+1)\cdot E$ with each row identified by a fundamental circuit $c_i$ (with respect to $T$) and each column by an edge $e_j$, where $b_{ij}=1$ if $e_j$ is in $c_i$ and has the same orientation as chord in $c_i$, $b_{ij}=-1$ if $e_j$ is in $c_i$ and has opposite orientation of chord in $c_i$, $b_{ij}=0$ if $c_i$ is not in $c_i$. The $B_f$ for $G$ is shown in Figure 3. The chord set $T''=(c, j, l, m)$ forms a unit matrix on the left. The branches of $T'$ are on the right. Circuits are formed by adding one chord at a time to $T$. Thus the circuits are: $bc$, $ghij$, $efgikl$ and $adefgikm$, corresponding to $c_1$, $c_2$, $c_3$ and $c_4$, respectively, where $c_1$ and $c_4$ are directed circuits. Fundamental circuits have the property that no circuit in the set can be obtained by a linear combination of other circuits in the set. The number of fundamental circuits in a graph is given by $E-V+1$, the number of chords. Once $B_f$ has been determined, all circuits in a graph, comprising the circuit matrix $B_a$, can be generated by performing all possible ring sum (Exclusive OR) operations indicated by $(\oplus)$ on the rows.
Reachability matrix

The reachability matrix \( R = [r_{ij}] \) has a value \( r_{ij} = 1 \) if a directed path exists between \( v_i \) and \( v_j \) and 0 otherwise [6]. The \( R \) matrix for \( G \) is shown in Figure 4. This matrix does not include the edge \( m \), because this would result in \( R \) having all ones, a special case where each vertex can be reached from every other vertex (strongly connected graph).

**APPLICATION OF GRAPHS TO PROGRAM DEVELOPMENT AND TESTING**

Directed graph representation of computer programs

The use of a directed graph to represent a program will now be demonstrated. In fact, the connected graph \( G \) which has been discussed in the examples is the graph of the ALGOL procedure in Figure 5. The circled numbers in this figure correspond to the vertex numbers in Figure 1a; edges correspond to ALGOL statements between vertices. The program constructs (e.g. *If Then Else*) of this procedure are shown in Figure 6. The four constructs (While Do, If Then Else, If Then and Main Line) are connected sub-graphs. The part of the procedure corresponding to no iterations and the satisfaction of all true conditions is called the Main Line. Each of the constructs can be obtained from the tree in Figure 6 by adding a chord to the tree. These chords are \( c \) for While Do, \( j \) for If Then Else, \( I \) for If Then and \( m \) for Main Line. Each of the constructs is an independent circuit as previously defined. Edge \( m \) is an artificial edge which has been added to the graph for the purpose of obtaining the Main Line construct as an independent circuit; it is not part of the ALGOL procedure. Using Main Line allows edges \( a \) and \( d \) which do not appear in the other three constructs, to be represented in the set of independent circuits. The independent circuits in matrix form \((B_f)\) are shown in Figure 3. The extent of branching at a vertex is given by the degree of the vertex. For example the beginning of the *If Then Else* construct is at \( v_s \). This vertex has degree 3, corresponding

```
PROCEDURE TEST_CONDITIONS;
  COMMENT TEST ALL CONDITIONS FOR MEMBER IDENTIFIED BY CURRENT_NODE;
  COMMENT IF ALL CONDITIONS HOLD ADD MEMBER TO LINKED LIST;
  BEGIN
    INTEGER A, I;
    LOGICAL FAIR;
    FAIR:=TRUE;
    I:=1;
    WHILE ((REQUEST(I) = "Q") AND ( FAIR = TRUE )) DO
      BEGIN
        FAIR:=MATCHING(I);
        I:=I+1;
      END;
    IF FAIR = TRUE THEN
      BEGIN
        A:=ALLOCATE1;
        IF LIST_POINTER = NIL THEN LIST_POINTER:=A
        ELSE SETCDR1(LAST,A);
        LAST:=A;
        SETCDR1(LAST,NIL);
        SETCAR1(LAST,CDR2(CURRENT_NODE+1));
      END;
  END TEST_CONDITIONS;
```

Figure 5—ALGOL procedure corresponding to graph of Figure 1a.
Implications of graph properties for program development and testing

Program constructs

A program graph can be partitioned into its constructs by first identifying a tree and then adding a chord at a time as shown in Figure 6. Each construct is a basic unit of a program which must be tested. The number of constructs or independent circuits is called the cyclomatic number. This was previously given as $E - V + 1$. This quantity has been shown to be highly related to difficulty of debugging.\(^1\)\(^2\)

Program paths

The Adjacency Matrix and its derivatives provide program path information. This information can be used to identify the various paths whose correct execution should be verified. Two elementary program paths are given by Figure 2b, where it is shown that there are two paths of length 2 from $v_8$ to $v_9$; these correspond to the If Then and If Then Else branches. It should be noted that path length as used in Figure 2 refers to number of edges and not to number of source statements.

Complete paths from $v_1$ to $v_{10}$ can be obtained by performing ring sum operations on the independent circuits of matrix $B_T$, as explained previously. The six possible paths so obtained are

- $a \ b \ c \ d \ e \ f \ g \ i \ k$
- $a \ b \ c \ d \ e \ f \ h \ j \ k$
- $a \ b \ c \ d \ e \ f \ g \ i \ k$
- $a \ b \ c \ d \ e \ f \ h \ j \ k$
- $a \ b \ c \ d \ e \ f \ g \ i \ k$
- $a \ b \ c \ d \ e \ f \ g \ i \ k$

These are paths from start vertex to terminal vertex which should be tested.
**Code reachability**

The reachability matrix can be used to ascertain whether any program code is not used. This would be indicated by one or more zero rows in $R$. The relative importance of vertices can also be ascertained by examining $R$ and noting the number of ways a given vertex can reach other vertices. A high number indicates that the vertex and the edges comprising the paths to the other vertices are relatively important for correct execution of the program and should be accorded corresponding emphasis in testing; $v_2$ is such a vertex in Figure 4.

Reachability may also be defined as the summation, over all vertices, of the number of ways in which a vertex can be reached. Average reachability can be obtained by dividing this figure by number of vertices. This is the way reachability was calculated in Table I, which will be described subsequently.

To make the use of directed graphs practical for program representation and complexity measurement, it is necessary to significantly automate the production of the various matrices and complexity measures from a definition of the program graph. Even the latter can be generated, if the problem has been put in the form of a decision table. Several automated tools exist for directed graph manipulation.

**PROGRAM COMPLEXITY MEASURES OBTAINED FROM DIRECTED GRAPHS**

The data in Table I show the results of an experiment conducted at the Naval Postgraduate School involving four ALGOL programming projects, where average values of four complexity measures were computed for programs with and without detected errors [3]. Three of the measures (cyclomatic number, number of paths and reachability) were obtained from directed graphs and were discussed previously in this paper. Complexity measure values were significantly lower for the no-error case, suggesting a set of quantitative measures for program quality control and error avoidance.

<table>
<thead>
<tr>
<th>Complexity Measure</th>
<th>No Errors</th>
<th>Mean Value</th>
<th>Number of Procedures</th>
<th>Errors</th>
<th>Mean Value</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclomatic Number</td>
<td>1.699</td>
<td>4.74</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Source Statements</td>
<td>9.361</td>
<td>27.23</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Paths</td>
<td>2.671</td>
<td>27.1</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reachability</td>
<td>10.1</td>
<td>120.3</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SUMMARY**

Several directed graph properties which are useful for program representation and complexity measurement were described. These were then applied to a small ALGOL program. Evidence was then presented suggesting that directed graph properties can provide quantitative measures of program quality.

**REFERENCES**