# DRAWING FLOWGRAPHS

***Bob Lockhart***
The Mathematics Department
Universiti Brunei Darussalam
Tunku, Gadong,
Bandar Seri Begawan Negara Brunei Darussalam
email: robert.lockhart@rl.ac.uk

***Robin Whitty***
Centre for Systems and Software Engineering
South Bank University
Borough Road, London SE1 0AA
United Kingdom

***ABSTRACT***

Studies the problem of representing flowgraphs on computer screens and describe an iterative path-searching algorithm which does this.

***Keywords: Flowgraphs, directed graphs, path***

## 1.0 INTRODUCTION

*Flowgraphs* are simply directed graphs with identified *Start* and *Stop* nodes and such that all nodes are on Start - Stop *Walks* and the stop node has out degree zero.

Flowgraphs occur naturally in many situations. We are interested in them because they are convenient descriptions of the logical flow control of computer programs [1].

The basic control structures of imperative programming languages match with corresponding basic *Prime* flowgraphs.

One can parse the flowgraph of a given program into its constituent primes: *REPEAT UNTIL* loops, *WHILE DO* loops, *CASE* statements etc. and, because this decomposition can be shown to be unique, one can use it to obtain software measurements. This idea (which is covered more fully in [1, 2, 3]) is the rationale for the Esprit Two COSMOS project[1] which built a tool to analyse programs into flowgraph for measurement purposes. Formal Specification languages, Imperative Programming Languages, and Entity Relationship Diagrams can all be processed to obtain metric information in this way. So the COSMOS tool provides a common set of measurements for a wide range of programming languages.

The COSMOS development has a user interface. Our field trials convinced us that its potential users wished to have some sort of visual representation of the flowgraphs which underpin it.

For this reason, we began to consider ways of displaying flowgraphs to our customers. This paper reports on that work.

The next section outlines the iterative drawing algorithm which we use and mentions some cosmetic enhancements to that original idea.

Section three covers various display facilities which our approach permits.

The final section gives some technical information about the design and implementation of our drawing program.

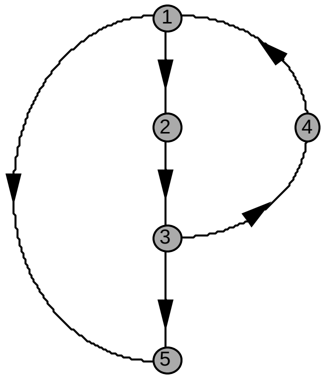
The COSMOS CASE tool is commercially available from Techforce, Leiden, Netherlands. There has been no previous publicly-available account of our drawing algorithm, although brief accounts of it were produced in internal documents within the project. The recent academic enhancements of the tool have not been described before.

## 2.0 THE LONGPATH DISPLAY METHOD

### 2.1 Terminology

A *Walk* through a flowgraph is a finite sequence of nodes in which any two consecutive nodes are connected by a directed edge leading from the first to its neighbour and such that consecutive edge directions do not oppose each other. The *Length* of a walk is the number of edges in it. *Paths* are walks in which no node occurs more than once. *Trails* are walks in which no edge occurs more than once.

Consider this (prime) flowgraph:



$L_2$ (Two-decision loop)

[1] COSMOS is ESPRIT project 2686 and was funded by the European Commission and the Austrian Government's "Innovations & Technologiefond".

Node 1 is designated as the flowgraph start node. Node 5 is its stop node. The sequence of nodes: 1,2,3,4,1,2,3,5 is a (start-stop) walk but not a trail. It has length 7. The sequence 1,2,3,4,1,5 is a (start-stop) trail but not a path. It has length 5. Finally, 1,2,3,5 is a path with length 3. It is the larger of the two start-stop paths in this flowgraph.

Counts of the various sorts of start-stop walks which can occur are useful software *Testability metrics* [2]. Paths are of particular importance to our drawer—its methodology is based on a decomposition of the flowgraph into disjoint paths, ordered by size.

## 2.2 The Longpath Method - decomposition

We start by splitting the flowgraph up into paths. This is done in the following three steps and results in each node being assigned a *path number:*

1. Find the longest start-stop path. Assign all of its nodes to *path one*.
2. Examine all paths which start at nodes not yet assigned a path number. Find the longest such path in which all of nodes, other than the final one, are as yet unassigned. Assign all nodes of this path, other than the final, assigned node, to *path number* N + 1, where N is the highest previously assigned path number.
3. If all nodes are now assigned to path numbers: finish. Otherwise: go to (2).

We continue by associating two co-ordinates with each node, based upon the path decomposition we have achieved. These two co-ordinates are the *path number* and the *position of the node on its path*. So, the start node will have the co-ordinates (1,1) because it is the first node on the first path, and the stop node will have the co-ordinates (1,Y) (assuming Y nodes appeared on the original start-stop path).

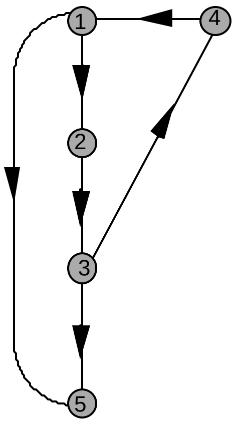
The co-ordinates associated with each node define a conceptual *grid* which is the basis of our display method. Suppose that the maximum first co-ordinate is X and the maximum second co-ordinate it Y. In that case, the algorithm detected X paths, the longest of which had length Y

## 2.3 The Longpath Method - Display

We display the flowgraph as follows:

1. Consider the top left vertex of the screen as the origin of two Cartesian axes: an X-axis stretching horizontally from the origin, and a Y-axis extending vertically down from it. These axes are coordinatised uniformly so as to give the bottom right screen vertex the co-ordinates (X+1, Y+1). Nodes of the flowgraph are now allocated screen positions defined by their co-ordinates on the grid formed by our two co-ordinate axes. It will be appreciated that disjoint paths appear on equidistant, parallel, vertical lines.
2. Where possible, edges connecting flowgraph nodes are drawn as straight lines. The policy is to first attempt to draw straight lines connecting nodes with nodes either vertically below them on the grid or to their right, and then to draw in the other edge connections. Curves are used only to avoid line clashing or edges appearing to intersect with intermediate nodes to which they are not, in fact, connected. The virtual grid structure facilitates this kind of simple geometric decision-making.

The flowgraph $L_2$ would be drawn like this:



$L_2$ (unenhanced

There are two paths, the longest being the start-stop path (which has length 3).

The co-ordinates of the nodes (in order) are: (1,1), (1,2), (1,3), (2,1) and (1,4).

## 2.4 Cosmetic Enhancements

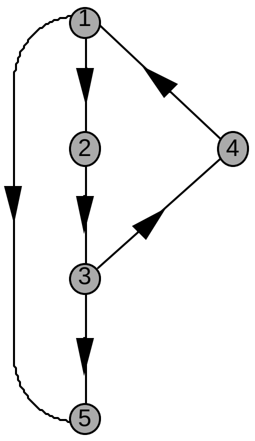
In response to the reaction of our customers we modified our display algorithm in these respects:

1. We cater for pathological “flowgraphs” which are disconnected or which have several nodes with out-degree zero. These came up in the application of our tool to formal specifications. Our policy is to display multiple stop nodes at the lowest vertical level (that is - to give each of them the highest possible Y co-ordinate).
2. Paths subsequent to the first are not always displayed with their first nodes at the highest vertical level (= lowest possible Y co-ordinate). This enhancement was made for purely cosmetic reasons after alpha-testing. We now fix the vertical orientation of these paths by considering their connections with paths to the left, “averaging” the effects in order to optimise the number of horizontal connections.
3. If it should so happen that the algorithm delivers a whole path with nodes {(a,m) : m=k ...N}, such that there are no nodes in any of the screen positions { (a-1,m) : m=k ... N} then the nodes in question are re-assigned to screen co-ordinates {(a-1,m) : m=k ...N}. That is, the path is displaced vertically to the left.

However, this is not done if path in question ends on a multiple stop node. This particular modification reduces the number of vertical lines appearing in the grid and means that some paths can appear vertically above some of their fellows.

The input to our drawer is a textual description of the flowgraph to be drawn (see (4.3.1) for details). The drawer counts nodes in the order in which they are presented to it and displays them as small circles containing their number. Parameterised size restrictions cause the program to display nodes as small black circles, or even not to display them at all, in the case of large flowgraphs. Start and stop nodes are distinguished by slightly larger screen node representations. All edges are depicted with small arrows at their centre, giving orientation.

These cosmetic enhancements would produce this display for $L_2$.



$L_2$ (enhanced)

Notice that the co-ordinates of node 4 have changed from (2,1) to (2,2) in order that its screen position is symmetric with respect to nodes 1 and 3 (with which it connects). All node numbers are shown.

Obviously, our drawer is designed to handle considerably more complicated flowgraphs than the one shown.

## 3.0 DISPLAY FACILITIES

The grid facilitates various geometric manoeuvres on the screen image of the flowgraph.

At present, we have implemented three of these (Zoom Lens, Store and Retrieve, flowgraph decomposition) but we have other developments in the pipeline.

Facilities are invoked by standard response buttons built into the display.

### 3.1 The Zoom Lens

The whole flowgraph is displayed together with a small rectangular box at the top left screen position (covering the start node). This box is to be thought of as a “Zoom Lens”. Its edges are always aligned with half point grid co-ordinates (so, its top left corner would initially have the grid co-ordinates (1/2,1/2).

One can perform these operations using the lens:

1. Alter the size of the lens - independently changing its horizontal and vertical dimensions x and y to any integral values in the range $1 \leq x \leq X$, $1 \leq y \leq Y$.
2. Move the lens through the screen representation of the flowgraph in any horizontal or vertical direction at the rate of one grid dimension at a time, in order to focus on a particular area, or to return instantly to the initial position.
3. Zoom in on the part of the display which the lens currently covers - that is, replace the existing display by a whole-screen representation of the part of it which was covered by the lens (this will usually cause some magnification). If the flowgraph is so large that node numbers are not displayed (see (2.4) the zoom option can be used to reinstate them in an enlarged detail of the whole representation.
4. Move the “Zoomed Detail” vertically or horizontally, or back to the initial position. This is analogous to moving a microscope lens over a specimen.
5. Un-Zoom from a given detail to the whole flowgraph, leaving the lens box shown in the same part of the flowgraph and with the same relative size.

### 3.2 Store and Retrieve

The grid co-ordinates of the nodes and the original description are all that is required to reproduce a display. Once obtained, it is possible to store this information into a text “retrieve file” which is headed by a parametrised keyword. If the program subsequently encounters a text file which starts with the keyword, it simply displays the flowgraph without repeating the path decomposition. The store and retrieve option makes it possible to do the decomposition processing in batch mode, retrieving the screen image at will, in real time[2]. One can even design a particular screen image by manipulating the information given in a retrieve file. This can be useful in expositions.

### 3.3 Flowgraph Decomposition

The decomposition of flowgraphs into constituent primes is important to the calculation of a wide range of software metrics. Our path searching methods can be used to give a

[2] Our workstation could take an hour to decompose flowgraphs with more than a thousand nodes but the subsequent display takes just seconds.

visual, successive, account of this decomposition.[3] In particular, the constructive processes of *Sequencing* and *Nesting* may be represented.

## 4.0 FURTHER TECHNICAL INFORMATION

### 4.1 Path Searching

The path searching algorithm works by running Monte Carlo simulations on the proffered flowgraph. The simulations use random numbers to make choices at nodes with out degree more than one.

The simulations are not designed to be exhaustive: exhaustive searching would result in unacceptably lengthy processing. A good approximation is often all that is really required to produce a reasonable screen image.

The point is that we are concerned only with displaying flowgraphs intelligibly. Whether or not we achieve the "best decomposition", as that might be defined by the pure longpath algorithm, is of secondary importance.

The speed at which the path searching operates depends on the size of the flowgraph and the power of the processor. Our Spark 1[4] processor handled 50 node flowgraphs in seconds, hundred node flowgraphs in minutes and thousand node flowgraphs in hours.

We have implemented similar simulation techniques to obtain stochastic approximations to *Testability Metrics* [2] as part of the COSMOS tool. This important class of metrics has hitherto been unavailable to static analysers because they are difficult to calculate. We believe the COSMOS tool is the first CASE tool to offer them.

Some recent work makes it possible to calculate some of the testability metrics exactly [5]; but these methods are heavily algebraic, and not so far appropriate to an industrial tool.

### 4.2 Program Structure

The program is written in C. All displays were handled using the Sunview window environment. Although this is now obsolete, the program was designed to be as independent of particular window systems as possible and should port to other systems with the minimum of work because the distinct functions of the program are represented by distinct program modules and all appeals to Sunview are quite basic, and made from one small module.

The program allocates and releases workspace dynamically based on the size of the flowgraph that it is processing. There is no theoretical limit to flowgraph size but, obviously, there will be an upper limit to the amount of available space in any real environment.

### 4.3 Textual Input and Output

Textual processing is handled by one program module.

#### 4.3.1 Input

Textual descriptions of flowgraphs are structured by keywords indicating where node lists, edge lists, and start and stop information is to be expected. These keywords are defined in a C "include file" and may be tailored to particular situations.
The program recognises all strings of displayable characters which occur between the node keyword and the edges keyword as individual program nodes (blanks terminate the strings).

Repetitions are not treated as separate nodes.
Flowgraph files may contain annotation anywhere after the start of the edge descriptions.

This would be an acceptable description of the flowgraph $L_2$ (the keywords used in this particular implementation are shown in bold type and annotation is in parenthesised italics).

| | |
|---|---|
| **Node** | first second 3rd 4th fifth |
| **Edges** | *(annotation can occur at any subsequent point)* first second *(this says there is an edge linking the node 'first' to the node 'second')* second 3rd 3rd fifth 3rd 4th 4th first first fifth |
| **start** | *(this keyword signifies the end of the edge descriptions)* first |
| **stop** | fifth |

Our program first recognises the five strings which represent nodes and, thereafter, matches those strings with the strings it finds in the remainder of the description.

Unmatched strings, such as those occurring in annotation, are ignored. Node numbering is based on the order in which strings recognised as nodes occur in the input file and is unrelated to any semantic meaning of the strings.[5]

This textual file would be displayed as was shown in (2.4).

#### 4.3.2 Output

Although the program needs Sunview to set up a drawing environment and produce a mechanism to get input from the user (using buttons) the drawing mechanisms used

[3] So far, this has only been implemented in the academic version of the tool and used for teaching purposes.
[4] © Sun Microsystems Ltd.

[5] This policy may be altered a little by the program in that it always numbers the start node as one and the stop node as the largest node number appearing.

include only the ability to set/unset an individual pixel and the ability to draw/undraw a straight line. All features of the display, including curves, arrows and nodes, are produced by appealing to these two facilities. We think that any conceivable window system would allow us them and that is the basis for our claim of extreme portability for the program.

The current implementation of the Zoom lens presupposes that the window system will “clip” displays that are logically off screen.

This probably makes the lens a good deal less efficient than it might otherwise be and is not an essential feature of its operation. This sort of clipping is available in all window systems.

### 4.3.3 Curve Drawing

We only link nodes by curves when straight lines would produce a false or obscure screen image. Curves are constructed by our own curve-drawing algorithm which is used to produce all the curves and arrows in our drawer.

A previous account of this was given in [4]. So far as we are aware, our method is unique to us.

The algorithm requires the user to specify the two *endpoints* which are to be linked by the curve, and to specify whether the curve is to arch to the left or right of the first point (called the *start point*) in connecting to the second point (called the *end point*).

One may also specify two curvature parameters -- real numbers Z and $\gamma$ in the range: $0 < Z, \gamma < 1$ which influence the shape of the curve.

In practice, our program adopts fixed default values for these parameters which we apply to all our curves. We usually set $\gamma$ to the golden number: $(\sqrt{5} - 1)/2$ and Z to 0.4.

The algorithm exploits the iteration possible in C functions to draw the curve as follows:
Starting with the two nominated screen positions A & B, the mid point C of the straight line connecting them is found. A point P(0) is then constructed in such a way that the ratio of lengths C-P(0) / A-C is equal to $\gamma$ and A-C-P(0) is a right-angle triangle with hypotenuse A-P(0).

Having found the point P(0), the algorithm calls itself twice more, nominating as endpoints the points A and P(0), on one call, and P(0) and B on the other. These two calls produce further points P(1) and P(2) forming two new right-angle triangles, but now the defining ratio is set to $(Z. \gamma)$.

The iteration then proceeds with four new calls on itself, nominating the end points:
A-P(1), P(1)-P(0), P(0)-P(2), and P(2)-B.

Now, the defining ratio is set to $(Z^2. \gamma)$ and four new points are produced: P(3), P(4), P(5) & P(6).

Each time a point is found it becomes the start point and the end point for two further calls, on each subsequent iteration, and each such call produces one further point.

At the nth iteration a further $2^{(n-1)}$ points are produced:
$P(2^{(n-1)} - 1), \ldots. P(2^n - 2)$
and the ratio used to do this is $\gamma . Z^{(n-1)}$.

The points created by this method define a curve from A to B.

Clearly, we do not perform an infinite number of iterations to construct our curve. In practice, we continue until contiguous points are within some pre-set Euclidean distance of each other and then connect them with straight lines.

There is no point in continuing with the iterations when the points are less than one pixel distance apart. That would be the ultimate resolution of any curve which could be drawn by any method whatsoever.

We have found this method gives us nice symmetric curves.

Since the input information is simply the points to be connected and the curve orientation, we need make no special allowances for the expanded areas of the screen that the Zoom facility provides. All Zooming does is alter the screen co-ordinates of points that are connected by our curves.

It would be possible to develop this idea, to draw asymmetric curves, or symmetric curves through several nominated points. It might also be interesting to investigate how this technique relates to others and how efficient it is.

## REFERENCES

[1] N. E. Fenton, *Software Metrics - a Rigorous approach*. Chapman and Hall, London, 1991.

[2] R. Bache and M. Müllerberg, “Measures of testability as a basis for quality assurance”, *Software Engineering Journal*, March 1990, pp. 86-92.

[3] N. Fuchs and A. Pengelly, “Software Structure and Cost Management - the Esprit 2 COSMOS project”, *British Telelcom Technology Journal*, Vol. 9 No 2, April 1991.

[4] R. Lockhart, “The COSMOS flowgraph editor”, in (J. Encarnaçáo, ed.) Eurographics Technical Report Series-Graphics Research & Development in European Community Programs (Eurographics ‘91) pp. 27-38.

[5] R. Lockhart. “Direction calculation techniques for testability metrics”, submitted.

[6] R. Bache & L. Leelasena. *QUALMS - A Tool for Control Flow Analysis and Measurement.* Obtainable from the Centre for Systems and Software Engineering South Bank Polytechnic, Borough Road, London SE1 0AA.

## BIOGRAPHY

**Bob Lockhart** and **Robin Whitty** were the Goldsmiths’ College representatives of the COSMOS project, which involved six industrial and academic institutions in four countries.

**Bob Lockhart** now works in computing at the Central Laboratory of the Research Councils, Chilton, Didcot, England. His research interests are in Homological Algebra, Software Engineering, and Biology.

**Robin Whitty** is Professor of Software Engineering and Director of the Centre for Software and Systems Engineering, South Bank University, London. His research interests include Software Metrics, Graph Theory, and the industrial application of Software Engineering.