Visual Programming: Limits of Graphic Representation

Jeffrey V. Nickerson

New York University
jnickerson@acm.org

Abstract

The effectiveness of graphic representations of computer programs is analyzed. Existing software metrics are modified for use in analyzing diagrams, and two new metrics are proposed: graphic token count and diagram class complexity. A graphic design measure, data density, is transformed into a computer science measure, token density. Using these metrics, graphic representations can be compared to each other and to textual representations. Conclusions are drawn about the relative strengths of graphic and textual representation.

1. Metrics for comparing representations

Visual programming languages are often compared with textual languages, but there is little discussion of quantitative ways of comparing graphic to textual representation. We approach the problem by finding comparable metrics for both text and graphic representation.

In [1], the author explores the most used textual metrics and their graphic equivalents, choosing Levitin's Token Count [2] (referred to here as *textual token count*) as an indicator of textual complexity. This metric is a count of the tokens in a program as parsed by a compiler.

We propose that the equivalent graphic complexity metric is:

Graphic token count = *number of nodes* + *number of edges* + *textual token count* + *number of enclosures* + *number of adjoinments*

The terms in this sum come from the observation that the following constructions when combined make up the great majority of computer science diagrams:

i. adjoinment:

where A and B are cells that physically adjoin - that share a side.

ii. linkage:

where A and B are nodes that are explicitly linked by an edge. All graphs and trees are forms of linkage diagrams.

iii. containment:

where one node is enclosed within another. Containment is most often used to indicate set relationships, as with Venn Diagrams.

The graphic token count of the above three diagrams are all 5. The corresponding textual representations *adjoin(a, b), link(a, b), contain(a, b)* are all of textual token count 5.

The graphic token count can be directly compared with the textual token count. However, Halstead [3] has observed that the characteristics of a programming language is related to the length of a program representation. This observation also applies to visual languages, leading to a metric for a graphic notation:

Diagram class complexity = *number of node types* + *number of edge types* + *number of label types.*

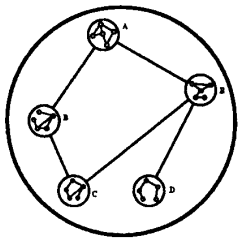
Node types are often distinguished by shape, edge types by line weight or arrowhead variations, and label types by font. In [1], the author shows how an object-oriented representation has a lower graphic token count but a greater diagram class complexity than an entity-relationship representation of the same domain.

Finally, modifying a Glinert metric [4] to be additive rather than multiplicative, we define a confusion metric:

Confusion count = *number of crossings* + *number of elbows*

It is clear that as the graphic token count of a linkage-based representation goes up, the confusion count tends to go up, due to the nature of planarity. In order to avoid a confusion count in graph representations, a hierarchical convention such as Pratt's H-Graphs [5] must be used. This can be visualized as graphs within nodes:

0-8186-6660-9/94 $04.00



We can add another dimension to our metrics by considering the area of page or screen over which the representation is made. Tufte [6] creates a data density metric, oriented toward the display of statistical data:

$$data\ density = \frac{number\ of\ entries\ in\ data\ matrix}{area\ of\ data\ graphic}$$

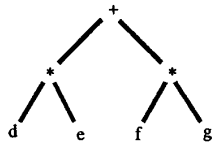
We propose that a similar metric be created for topological diagrams:

$$token\ density = \frac{number\ of\ graphic\ tokens}{area\ of\ data\ graphic}$$

In general, diagrams that make use of metric space, such as statistical graphs or maps, have much greater information densities than topological diagrams. Usually, textual representations of programs are more dense than graphic representations, sometimes by an order of magnitude. Hybrid languages, in which diagrams provide structure for text, as in Buhr [7], can have densities similar to that of pure textual languages.

2. Trees and Graphs

Consider the minimal diagrammatic representation of a binary tree:

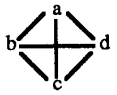


The graphic token count is 13, or 2n - 1, where n is the number of nodes in the tree. The alternate textual representations of this tree,

infix with parentheses: (d * e) + (f * g)
postfix: d,e*f,g*+
prefix: +*d,e*f,g
infix: d*e+f*g

have textual token between 7 (the number of nodes) and 9. The minimum textual token count is practically half that of the graphic representation.

Consider a graph:



The graphic complexity a graph is $e + n$: in this example, 10. One of the most common ways to represent this textually is with an edge list:

<a, b>, <b, c>, <a, c>, <a, d>, <b, d>, <c, d>

Each edge contains two nodes, a separator, and a bracket expression. Therefore the token count of an edge list representation is 4e, or 24 for this example. This is substantially greater than the graphic representation. Consideration of other textual representations of graphs in [1] shows that the textual count is 1 - 4 times greater than the graphic count.

3. Conclusions

Application of these metrics to current visual programming languages does not paint an optimistic future for the use of fully general, fully diagrammatic visual programming languages due to their low density. Instead, it suggests that those areas where metric space are part of the problem domain will lend themselves to the visual. Also, those problems lending themselves to graph rather than tree representations will be best handed visually. Due to planarity problems, it suggests that use of embedding constructions such as H-Graphs will be important. Finally, hybrid languages in which graphic representations of program structure are combined with textual expressions can match textual densities.

References

[1] Nickerson, Jeffrey V., *Visual Programming* , Ph.D. Dissertation, Dept. of Computer Science, New York University, 1994.

[2] Levitin, A. V., How to Measure Software Size, and How Not To, *10th International Computer Software and Applications Conference*, Chicago, 1986, 214-239.

[3] Halstead, Maurice H., *Elements of Software Science*, New York: Elsevier, 1977.

[4] Glinert Ephraim P., Nontextual Programming Environments, in Chang, Shi-Kuo, ed. *Principles of Visual Programming Systems*, Englewood Cliffs, NJ: Prentice Hall, 1990, 144-230.

[5] Pratt, Terrence W., Formal Specification of Software Using H-Graph Semantics, in Ehrig, H., M. Nagl, and G. Rozenberg (eds.),*Lecture Notes in Computer Science #153:* Berlin: Springer-Verlag, 1973, 314-332.

[6] Tufte, Edward R., *The Visual Display of Quantitative Information*, Chesire, Conn: Graphics Press, 1983.

[7] Buhr, R. J. A., *Practical Visual Techniques in System Design*, Englewood Cliffs, NJ: Prentice-Hall, 1990.