Encoding Presentation Emphasis Algorithms for Graphs

Emanuel G. Noik

CSRI, University of Toronto, 6 King's College Rd., Toronto, ON, Canada M4S 1A1

Abstract. While graphs can effectively visualize one or more relations on a set of elements, drawings of large graphs can be difficult to understand. As such, many presentation emphasis techniques for visualizing graphs such as fisheye views have been proposed. A recent survey paper [9] described an abstract space of techniques and identified common shortcomings. Here we outline a high-level language that addresses several of these limitations; the language is used to: 1) select subsets of graph elements; 2) compute a real-valued priority for each element; and, 3) encode presentation strategies that automatically emphasize elements based on subset membership and priority.

1 Introduction

Despite a growing body of literature in automatic graph drawing, one of the most pervasive obstacles for practical graph-based information systems remains the inability of user interfaces to effectively cope with large or highly detailed graphs. Aside from computational complexity issues, the fact remains that even when drawn by the most effective graph layout algorithms (or hand-drawn by an artist, for that matter), drawings of large or complex graphs can be very difficult (if not impossible) to understand. To be fair, this problem is not confined to the graph: most visualization techniques do not scale very well. In particular, there is a limit to the amount of detail that can be effectively shown and discerned on a computer display; techniques such as *pan and zoom* facilitate the exploration of detailed visualizations, but fail to integrate local detail and global context and often confuse users. In response, Furnas suggested an analogy to the wide angle or "fisheye" lens: the *generalized fisheye view* [5]. By balancing local detail and global context, fisheye views (FEVs) display information at several levels of abstraction simultaneously. Unfortunately, even solutions that have incorporated fisheye views have failed to provide flexible means to 1) specify expressions for computing priorities (degree of interest metrics) for the elements in a visualization, and 2) encode alternative presentation strategies that automatically emphasize elements based on their priorities.

In a recent survey of emphasis techniques used in graph visualization, we reviewed a large portion of the literature in this area and identified these and other common shortcomings [9]. In this paper, we outline a high-level language that effectively addresses several of these limitations.

The paper is organized as follows. Section 2 reviews relevant terminology; Section 3 summarizes relevant portions of the survey findings; Section 4 describes the proposed language; and, Section 5 provides brief notes on its implementation.

2 Terminology

2.1 Graphs and Graph Diagrams

A *graph* $G = \langle V(G), A(G) \rangle$ consists of a set of *vertices* $V(G)$, and a set of *arcs* $A(G) \subseteq V(G) \times V(G)$. We employ the hygraph formalism as used in the Hy+ system [1] to express nested graphs. A hygraph $G = \langle V(G), A(G), B(G) \rangle$, is a graph that contains a set of blobs $B(G) \subseteq V(G) \times 2^{V(G)}$; a blob associates a (container) vertex with a set of vertices that it contains; we assume that the containment relation is hierarchical. Vertices, arcs, and blobs are labelled and may have additional attributes (numbers or strings) which make it possible to associate domain-dependent data with graph elements (*e.g.,* arc weights). Thus a graph can be thought of as a set of three relational database tables, with external attributes stored in additional fields.[1]

A graph *diagram* or *layout* is a visualization of the graph in which vertices are depicted as *nodes* and arcs as *links* (in fact, graph diagrams are often referred to as *node-and-link* diagrams). Since the blob containment relation is restricted to be hierarchical, blobs are easily depicted by visual containment (*e.g.,*by nesting closed rectangular or polygonal regions). Graph layouts may be hand-drawn or may be generated automatically by a *graph layout algorithm* [2].

2.2 Emphasis-related Concepts

The following terminology comes in large part from the generalized fisheye view formalism [5].

The generalized FEV metaphor is based on the workings of the fisheye or very wide angle lens used in photography, which magnifies the image at multiple levels – greatest near the focus and least in the periphery. By balancing local detail and global context, FEVs can simultaneously display information at multiple levels of abstraction.

When displaying large structures, the basic strategy uses a *degree of interest* (DOI) function to assign to each point in the structure a number, or *priority*, that quantifies the user's interest in that point given the current task. Priorities may be assigned by the user, or may be computed by an algorithm. In a generalized FEV, the DOI is decomposed into two components: *a priori importance* (API) which computes the global importance of any point in the structure, and *distance* (Dist) which computes the conceptual distance between any two points. If one point is selected as the current focus of interest, or *focal point* (FP), then in its simplest additive form, $DOI(p, f) = API(p) - Dist(p, f)$ is the user's degree of interest in point p given FP f; thus DOI increases with API and decreases with Dist.

I have extended this simple model to handle multiple variable magnification FPs by introducing several additional concepts. A *focal point response function* $f.resp$ computes the change in relevance of point p due to FP f and is a function of $f.mag$ (see below) and $Dist(p, f)$. Usually $f.resp \propto \frac{1}{Dist}$, however if $f.resp \propto Dist$, then we can define *peripheral points* – selected points that decrease the relevance of nearby

[1] To store a set of blobs in a relational table, the set is represented as a binary relation: $\{(u, v_i) \mid (u, \{v_1, \ldots, v_n\}) \in B(G)\}$.

points.[2] A *focal point magnification* $f.mag$ is a non-negative number that captures how interesting FP f is in a visualization. Usually $f.resp \propto \frac{f.mag}{Dist(p,f)}$, so a FP with a high $f.mag$ will tend to increase the relevance of proximate points more than if it had a lower magnification. Given a set of focal points F and a point p, the *focal point importance* (FPI) of p is an aggregate function $\mathcal{AG}$ of the response of FPs F on point p:

$$FPI(F,p) = \mathcal{AG}_{f \in F}(f.resp(f.mag, Dist(p,f)))$$

where $\mathcal{AG}$ is one of: sum, min, max, avg; by default we assume sum. In this extended model, $DOI(p,F)$ is a function of $API(p)$ and $FPI(p,F)$.

It is easy to show that this model can express the original basic FEV strategy. Since the basic strategy is limited to a single FP without magnification, we get:

$$\begin{aligned} f.resp(mag, dist) &= -dist \\ FPI(p,\{f\}) &= f.resp(f.mag, Dist(p,f)) \\ &= -Dist(p,f) \\ DOI(p,\{f\}) &= API(p) + FPI(p,\{f\}) \\ &= API(p) - Dist(p,f) \end{aligned}$$

Since API values often cannot be directly compared to $Dist$ values, and since it is more convenient to assume that $Dist$ is in $[0,1]$ when formulating expressions for $f.resp$, we assume that all priorities (API, Dist, FPI, and DOI) are normalized before they are used. Given a set of priorities $P = \{p_1, \ldots, p_n\}$, the normalized value of p_i is

$$\begin{cases} \frac{p_i - min(P)}{max(P) - min(P)} & \text{if } max(P) \neq min(P) \\ 1 & \text{otherwise} \end{cases}$$

where $min(P)$ and $max(P)$ are the minimum and maximum values in P. In this formalization, $DOI(p,F)$ will be in $[0,1]$ for all points: the most interesting points will have a DOI of 1 and the least 0.

3 Limitations of Previous Work

A recent survey of presentation emphasis techniques for visualizing graphs [9] identified several common limitations. The language which we present in the next section addresses the following three:

1. **Alternative presentation emphasis strategies:** most existing techniques do not exploit alternative presentation strategies. In particular, most can be grouped into three categories: filtering (suppress the display of low priority elements); distorting (distort the size and position of elements to give more display space to higher priority elements); and, filtering-distorting hybrids. There has been very little study of

[2] Peripheral points are convenient in situations where it is easier to identify uninteresting elements rather than interesting ones.

techniques that generate *adorned views* [9] – views in which elements are emphasized by varying other graphical attributes such as colour, shading, line style and thickness, as well as audio [7], and various types of motion (*e.g.,* "in-betweening" animation [11], vertical oscillations and small random movements [3], and vibration or pulsing [12]). A quick glance at a presentation graphics text (*e.g.,* [13]) suggests that many intriguing possibilities remain unexplored.

2. **Client priority algorithms:** most existing techniques use fixed (domain-dependent) algorithms to compute priorities. Some techniques permit clients (programs or users) to submit a set of static priority values (API), but do not provide the means to express distance or DOI metrics.
3. **Priority-to-Presentation Mapping:** most existing techniques use fixed mappings, which means that users cannot alter the way that differences in priorities are reflected in the visualization. What is required is a flexible means to describe how DOI values should be used to compute appropriate values for relevant graphical attributes.

4 Dye: A Presentation Emphasis Language

Dye[3] is a special purpose specification language that is based on relational algebra with arithmetic and aggregation [14]. The current version of the language supports the standard relational algebra union and difference operators, and the *closed join* operator, defined as follows:

$$\dot{\bowtie}(\{R_1, \ldots, R_n\}, F) = \pi_{R_1}\sigma_F(R_1 \times \ldots \times R_n)$$

Notice that for $n = 1$, the closed join reduces to the relational algebra selection operator:

$$\begin{aligned}\dot{\bowtie}(\{R_1\}, F) &= \pi_{R_1}\sigma_F(R_1) \\ &= \sigma_F(R_1)\end{aligned}$$

It is not difficult to show that the above operators define a subset of relational algebra that is schema- and tuple-closed under the three basic operations and the operators derived from these. That is, given a set of relations $R_1, \ldots, R_n$, a relation R produced by applying a sequence of the above operators will have the scheme of one or more R_i, and will not contain any tuples that were not present in at least one R_i.

A Dye program captures four types of information: subset selections; focal point response functions; priority functions; and, actions that use set membership and DOI values to assign values to various graphical attributes.

In addition, Dye is extensible – algorithms can be added to evaluate queries that cannot be expressed in relational algebra (*e.g.,*transitive closure). The current version of Dye, for example, includes support for computing (the lengths of and elements along) shortest paths [4]. The general syntax of a Dye program is:

[3] Graphite [1, 8] can be thought of as the raw material (carbon) for producing drawings of graphs; Dye adds *colour* (visual emphasis through the systematic manipulation of various graphical attributes including colour) to Graphite.

```
<program> ::=
  ["sets" [<set_assignment>]+ ]
  ["fpresponse" [<response_function>]+ ]
  ["doi" [<doi_assignment>]+ ]
  ["actions" [<action_block>]+ ]
```

Unfortunately, due to strict page limits, a complete description of the language could not be included – it can be found in [10]. In the absence of a complete language description, we hope that the following example as will whet the reader's appetite. More extensive and numerous examples consisting of graph drawings and Dye source code were presented at the DIMACS 1994 Workshop on Graph Drawing Poster Gallery. An electronic version of this poster is available by anonymous ftp from ftp.db.toronto.edu in pub/noik/gd94poster.ps.Z.

Figure 1 shows a graph drawing of a passenger flights database generated by an anchored force-based layout algorithm. The vertices of the graph represent cities; an (undirected) arc between a pair of vertices implies that some airline operates flights between the corresponding cities. Thus in addition to the purely topological information, arcs are labelled (by the name of the airline – arc labels are suppressed by default to reduce visual clutter) and have two additional attributes, namely the price of a one-way ticket (in dollars), and the duration of a one-way trip (in minutes). Similarly, each vertex is labelled by the name of the corresponding city.

Figure 2 shows a diagram and emphasis program that adds a number of presentation elements. Link colour and line thickness are assigned by iterating over each arc `a` in the set of arcs `A`. In this example, colour is chosen depending on the arc label, and thickness is set relative to arc DOI which is the ratio of ticket price (numeric arc attribute 0) to flying time (attribute 1). In the flights database as in most domains, paths in graphs usually have a meaningful and significant interpretation. In particular, paths can provide a fundamental measure of distance between a pair vertices in a graph. The *minpath* primitive in this example takes four arguments: a set of arcs, a traversal direction (forward, reverse, or undirected – may traverse arc in either direction), an expression to evaluate for each vertex along the path, and an expression to evaluate for each arc along the path. Here, the primitive computes the length of the shortest undirected path using the *time* arc attribute and incorporating an additional 30 minute delay for each vertex along the path. The focal point response curve in this example varies as the cube of proximity – this means that the FPI of nodes that are conceptually close to one or more focal points will be higher than the FPI of those that are conceptually distant. Vertex API is computed with the `avg` aggregate function; here it is the average *price* to *time* ratio of all incident arcs – a good indication of how expensive the flights through the corresponding airport are. In this example vertex DOI is a weighted sum of API and FPI and is used to vary node size through the `v.s.x` and `v.s.y` scaling primitives.

5 Implementation

Dye was implemented as a combination lex tokenizer and yacc parser that generates C++ source code that is automatically compiled and dynamically linked with the Dye run-time library to the Graphite layout engine [8]. The Graphite framework has been

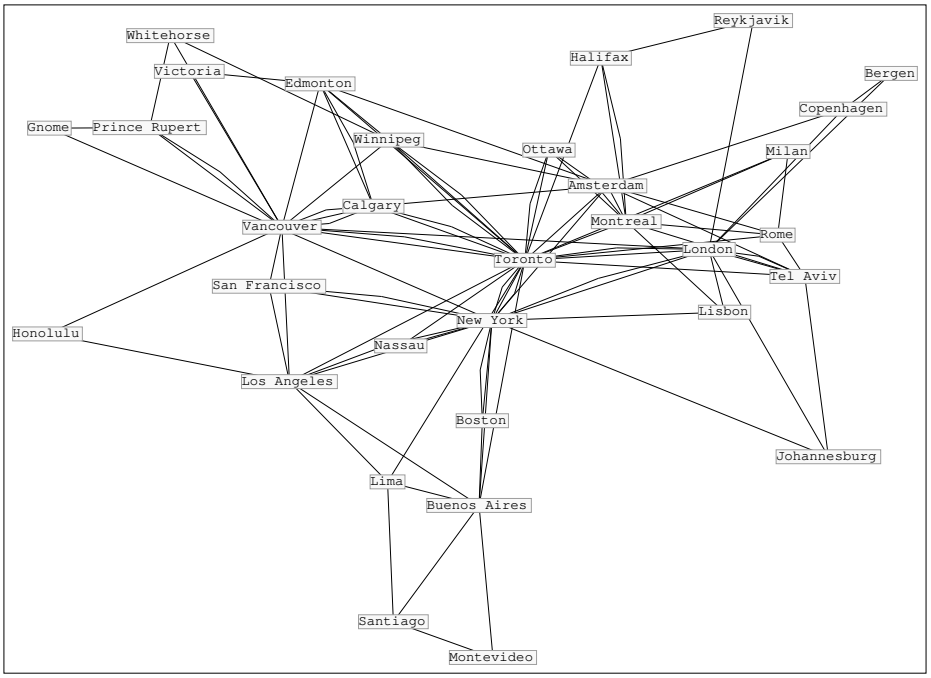
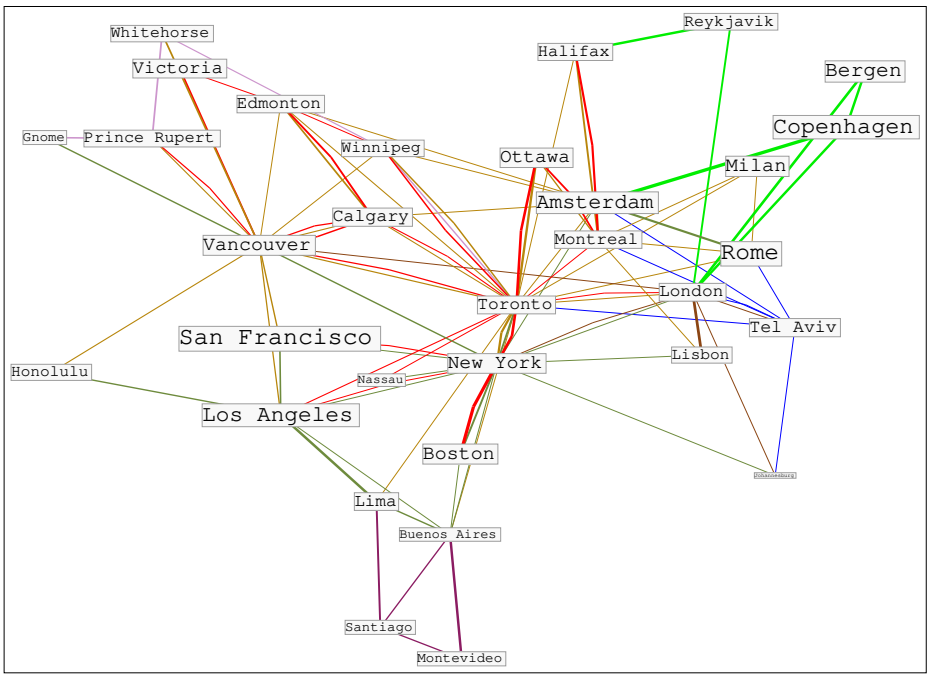


Fig. 1. The original drawing of the flights database produced by an anchored force-based layout.

modified to take into account the graphical attribute information generated by Dye. Thus for example, nodes are pre-scaled based on the `s.x, s.y, s.z` attributes and post-translated based on the `d.x, d.y, d.z` attributes; space is allocated only for objects that have not been hidden. The `layout` attribute directs the Graphite layout engine to use specific layout algorithms for selected elements of the visualization, which realizes a composite layout strategy [6]. Other attributes such as line style and width, and colour do not typically impact the layout phase, and are therefore only used in the final rendering phase.

6 Summary

We have argued that current approaches for incorporating visual emphasis in graph visualization are too limited. In particular, most techniques do not provide adequate means to encode degree of interest metrics, alternative presentation emphasis strategies, and mappings from the former to the latter. We have proposed a high-level language that can be used to encode precisely this type of information in a simple yet powerful manner. In the future we would like to see a tighter coupling of the graph layout and presentation emphasis components (*e.g.,*make degree of interest an explicit input to




```
#include <dye/rgb.h>
#define price %0
#define time  %1
fpresponse fpr := (mag * ((1-dist) ** 3 ));
doi
  v.api  := avg(union(v.inA,v.outA),true,(a.price/a.time));
  v.dist := minpath( A, undirected, 30, a.time );
  v.doi  := ( ( v.api * 0.3 ) + ( v.fpi * 0.7 ) );
  a.api  := ( a.price / a.time );
  a.doi  := a.api;
actions
  for v in V
    v.s.x = v.s.y := ((0.25 + (v.doi * 4.75)) ** 0.5);
  endfor
  for a in A
    if a.label == "aa" then a.fg_clr := DarkOliveGreen4; endif
    ...
    a.line.width := (1.0 + (a.doi * 5.0));
  endfor
```

Fig. 2. Colour links based on arc label (airline); set link width based on the flight's *price* to *time* ratio; generate a fisheye view with API = average price-time ratio of incident arcs, and distance = total flying time along fastest route assuming each stop contributes an additional 30 minutes; set node size relative to DOI; here *San Francisco* and *Rome* serve as focal points.

graph layout algorithms); we also plan to extend Dye to include dynamic presentation elements such as animation.

Acknowledgements

This work was done as part of on-going research in relational data visualization within the database group at the University of Toronto. I wish to thank Alberto Mendelzon, Dimitra Vista, and Yoram Kornatzky for their insightful comments and discussion.

References

1. M.P. Consens, F. Ch. Eigler, M.Z. Hasan, A.O. Mendelzon, E.G. Noik, A.G. Ryman, and D. Vista. Architecture and applications of the Hy+ visualization system. *IBM Systems Journal*, 33(3):458–476, Aug. 1994.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, to appear. Preprint avail. by anon. `ftp` from `ftp.cs.brown.edu:pub/papers/compgeo/`.
3. K.M. Fairchild, S.E. Poltrock, and G.W. Furnas. Semnet: Three-dimensional graphic representations of large knowledge bases. In R. Guindon, editor, *Cognitive Science and its Applications for Human-Computer Interaction*, pages 201–233. Lawrence Erlbaum Associates, 1988.
4. R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
5. G.W. Furnas. Generalized fisheye views. In *ACM CHI '86*, pages 16–23, Boston, MA, Apr. 1986. ACM.
6. T.R. Henry and S.E. Hudson. Interactive graph layout. In *ACM UIST '91*, pages 55–64. ACM, 1991.
7. B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *IEEE Visualization '91*, pages 284–291, San Diego, CA, Oct. 1991.
8. E.G. Noik. Graphite: A suite of hygraph visualization utilities. In A. O. Mendelzon, editor, *Declarative database visualization: recent papers from the Hy+/GraphLog project*, pages 108–126. Tech. rep. CSRI-285, U. of Toronto, Jul. 1993.
9. E.G. Noik. A space of presentation emphasis techniques for visualizing graphs. In *GI '94: Graphics Interface 1994*, pages 225–234, Banff, AL, Canada, May. 1994.
10. E.G. Noik. *Encoding Presentation Emphasis Algorithms for Graphs*. PhD thesis, Dept. of Comp. Sci., U. of Toronto, in preparation.
11. S.P. Reiss. A framework for abstract 3d visualization. In *VL '93: IEEE Symposium on Visual Languages*, pages 108–115, Bergen, Norway, Aug. 1993.
12. D. Schaffer, Z. Zuo, L. Bartram, J. Dill, S. Dubs, S. Greenberg, and M. Roseman. Comparing fisheye and full-zoom techniques for navigation of hierarchically clustered networks. In *Graphics Interface '93*, pages 87–96, May. 1993.
13. E.R. Tufte. *Envisioning Information*. Graphics Press, P.O. Box 430, Cheshire, Connecticut, 06410, 1990.
14. J.D. Ullman. *Principles of Database and Knowledge-base Systems*, volume 1. Computer Science Press, 1803 Research Boulevard, Rockville, MD, 20850, 1988.