RC22396 (W0204-057) April 15, 2002
Computer Science

IBM Research Report

Managing Graph(ical) Complexity with Raisin and Its Category Explorer

Douglas N. Gordin
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Managing Graph(ical) Complexity with Raisin and Its Category Explorer

Douglas N. Gordin
IBM T.J. Watson Research Center
PO Box 218; Route 134
Yorktown Heights, NY 10598
011-914-945-3266
dgordin@us.ibm.com

ABSTRACT
Graphs offer a powerful way to view the relationships between objects. Yet, as useful as small graphs are for seeing relationships, large graphs are frustrating because their complexity overwhelms our ability to trace through patterns of relationships. The *Raisin* system helps manage this complexity by giving the means to layout a graph well; index a graph using categories based on structure and other criteria; highlight and abstract via selection, hiding, and aggregation; and create new categories that distill the user's conclusions for analysis and presentation.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces– *Graphical user interfaces (GUI), Interaction styles (e.g., commands, menus, forms, direct manipulation), Theory and methods*.

General Terms
Algorithms, Design, Human Factors

Keywords
Graph visualization, Focus+Context, Directed Acyclic Graph, Aggregation, Tree control, Categorization, Simplification

1. INTRODUCTION
Graphs offer a powerful way to view the relationships between objects as well as a strong set of mathematical tools for the analysis of those relationships. In trying to account for the power of visual diagrams and graphs, Larkin & Simon analyzed how diagrammatic representations alter the cognitive ecology of searching for information [1]. Their explanations also shed light on why large graphs are extremely frustrating—the huge number of relationships mean we can not grasp all the connections and tracing through specific pathways is difficult because the background overpowers our attempt to focus on selected parts.

Visualization techniques often focus on two task models: data discovery and communication of results. Data discovery is the search for interesting patterns and explanations. A powerful technique is to partition the data into categories and then devise causal connections between the categories. Afterwards the data is arranged so as to best convey the original data, the explanatory categories, and the casual connections.

The CATEGORY EXPLORER is a graph visualization tool to help users discover categories within graphs and use these categories to analyze and re-present the graphs for communication In brief, the CATEGORY EXPLORER provides an index on the nodes of graph that allows them to be navigated via a series of turn-down controls. Using these controls users can explore a variety of formal graph properties and evolve semantically meaningful categories that can be stored for later use.

The paper is organized as follows: A) An extended example illustrates how *Raisin* is used to explore data; B) the functionality of *Raisin* is explained; C) design issues are discussed.

2. AN EXTENDED EXAMPLE
The need for the CATEGORY EXPLORER is quickly seen when attempting to analyze complex graphs. The graph explored here shows an influence graph beginning with the Renaissance painter Titian, connecting to painters he influenced, and then painters they influenced, and so on, for a total of three levels of indirection [2]. Within this group of painters arcs occur whenever a painter is understood to have influenced another. This Titian influence graph is medium size with 117 nodes and 274 arcs. The arcs are directed, because the influence relation is asymmetric (i.e., saying Michelangelo influenced Rodin, does not imply that Rodin influenced Michelangelo). An initial layout of this graph is shown in Figure 1. This graph should be useful for understanding the spread of style and influence, beginning with Titian. Yet, its complexity makes this goal elusive.

2.1 Focusing In
A primary goal of graph visualization techniques is to flexibly move between focusing in on a subject and putting the subject in a broader context [3]. Here I show how Raisin helps focus in on Titian and the artists whom he directly influenced; next, I show how to view these influences in context.

Once Titian is selected, the artists he directly influenced can be selected, and graph layout redone to produce a graph that shows the influences between the artists that Titian directly influenced (see Figure 2). This raises the issue of how to decide whether a

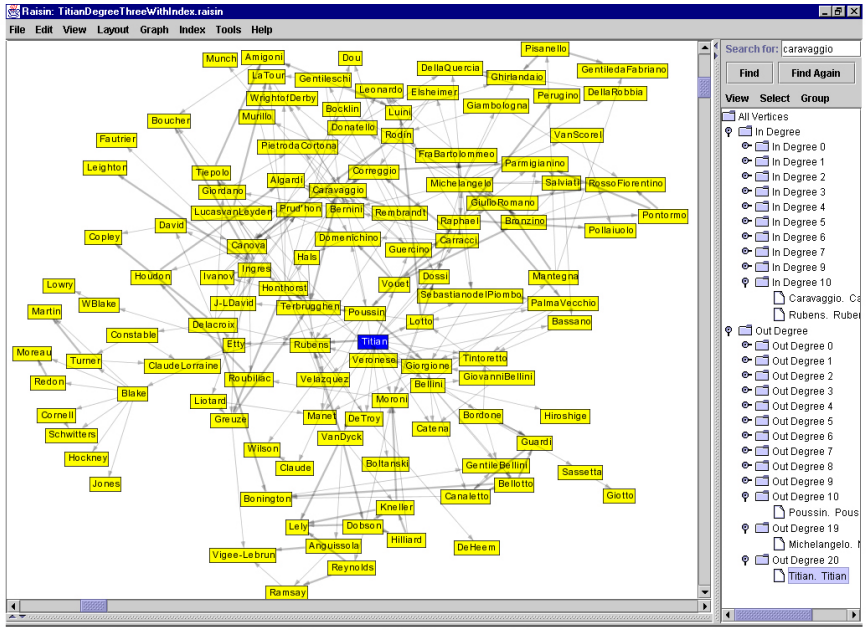

Figure 1. Influence graph of Titian with Category Explorer showing degree of nodes.

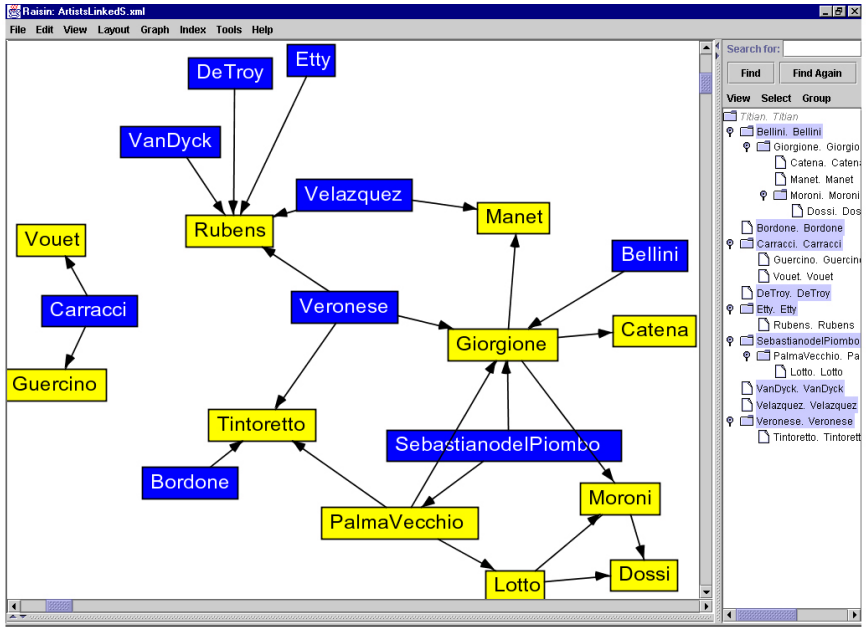

Figure 2. Direct influences with depth first tree. Roots are highlighted.

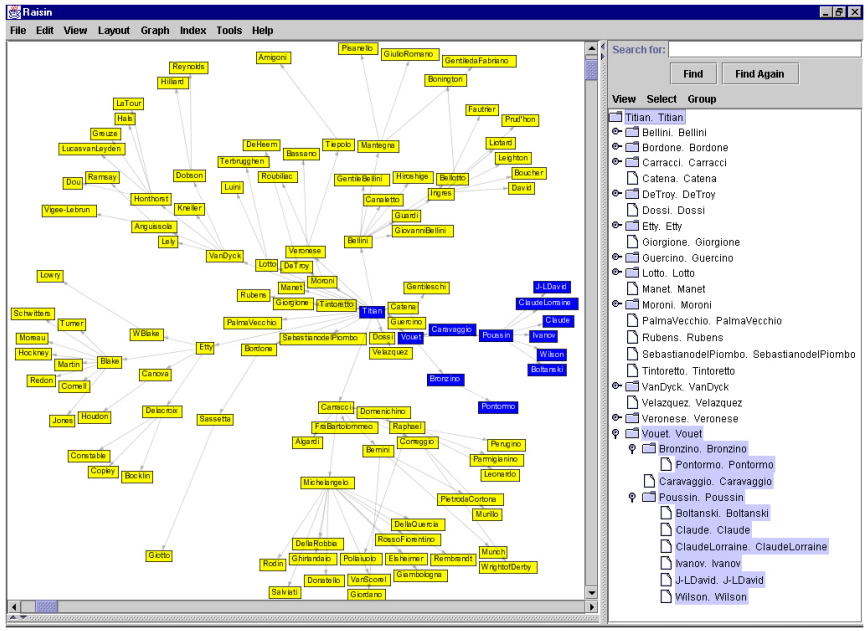

Figure 3. Breadth first tree of Titian Influences. Vouet subtree highlighted.

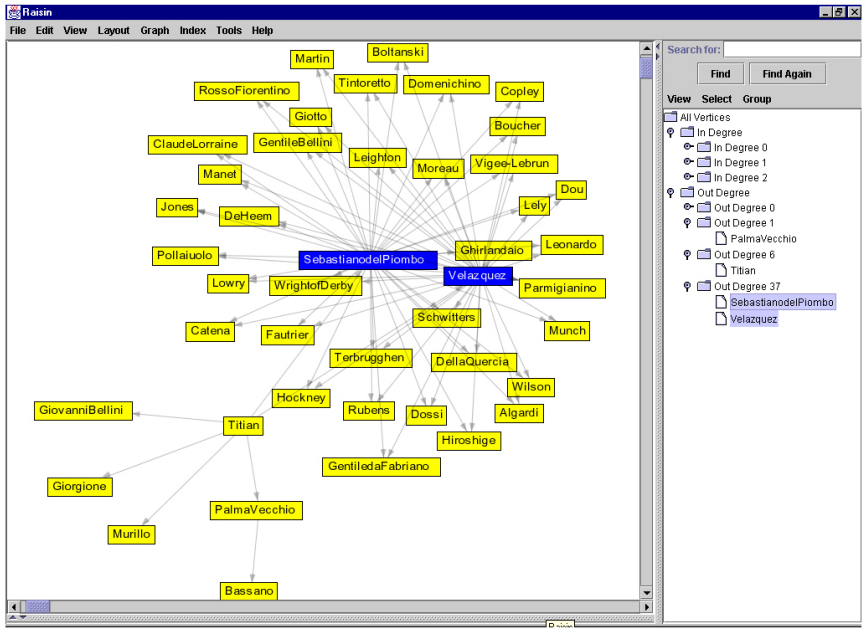

Figure 4. Partial order with most influential highlighted.

given influence was received directly from Titian or through an intermediary. This layering of influence can be analyzed by producing a depth first spanning tree in the CATEGORY EXPLORER and use that to highlight root nodes (i.e., nodes with no incoming influence arcs). Hence, Titian's influence is most unambiguous for these highlighted artists, since there is no question of the influence coming by way of an intermediary. Further, one might speculate that these artists demonstrate the most differentiated aspects of Titian's influence.

2.2 Seeing Titian's Influence in Context

A broader view of Titian's influence looks at direct and indirect influences without the distraction of influences amongst these artists (see Figure 3). The layout is radial with Titian in the center and surrounding him those that he directly influenced, followed by indirect influences, and so on. This visualization was generated by having the CATEGORY EXPLORER generate a breadth-first spanning tree and then visualizing that tree. This tree is shown in the CATEGORY EXPLORER (see Figure 3; right pane). The highlighted nodes show the subtree beginning with Vouet who was directly influenced by Titian.

A key question is which of the artists Titian influenced were the primary ones influencing others. This question can be addressed by generating a partial order (or lattice) based on the influences relation (as shown in Figure 1). For example, if artist A influenced X, Y, and Z, and artist B influenced Y and Z, then A is greater than B in a partial order since A influenced all of the artists that B did and more (see Figure 4). Note how the two highlighted artists, Sebastiano del Piombo and Valazquez are the most central. This centrality is also borne out by their node degree as shown in the CATEGORY EXPLORER (see Figure 4). There it shows that in the partial order they are each above 37 other artists, thus they appear to be the primary carriers of Titian's influence. In contrast, Palma Vecchio appears to carry Titian's influence on to very few other painters. Thus, we can conclude that Titian's influence was best diffused through the highlighted artists.

3. RAISIN & ITS CATEGORY EXPLORER

This section discusses the functionality of Raisin as divided between its GRAPH VIEWER (i.e., in Figure 1 the left pane) and CATEGORY EXPLORER (e.g., in Figure 2, the right pane).

3.1 The Graph Viewer

The Graph Viewer supports visualizing and manipulating the nodes and arcs that make up the graph. The graph representation is general in that any graph consisting of simple nodes and arcs may be represented. The arcs may be directed or undirected and an optional weight can be supplied. Key features are the ability to layout the graph using multi-dimensional scaling (MDS) and selecting, dragging, hiding, and aggregating nodes. In addition, there are numerous display options (e.g., zooming, centering).

The graph's layout of is critical to helping users see relationships between nodes. Common problems are arcs that cross one another, arcs that are too long, and overlapping nodes. MDS produces a layout by bringing into alignment the structure of the graph with its appearance on the screen. Its structure is described through graph distance or the shortest paths between pairs of nodes. Its appearance is described through Euclidean distance how far apart pairs of nodes are on the screen. For a given pair of nodes the difference between the measures is its stress. MDS seeks to minimize the overall stress by iteratively moving the nodes (see [4] for full details on this algorithm). Following the MDS a cleanup pass is needed to move nodes that overlap. This is done by a sweep line algorithm that goes from left to right and top to bottom, moving overlapping nodes rightwards or downward, respectively.

3.2 The Category Explorer

The CATEGORY EXPLORER can simplify and focus the GRAPH VIEWER by selecting, hiding, and aggregating the graphs' nodes. The CATEGORY EXPLORER interface is similar to the familiar tree control used by the Windows Explorer and here implemented via a Java 1.3 JTree. As a tree control it provides the standard functionality such as choosing levels of detail through turn-downs. Further, menu items are provided for more systematic exploration (e.g., Expand All, Expand Next, Select Descendents). As a control on the GRAPH VIEWER it can select, hide, and aggregate nodes. When a node is selected in the CATEGORY EXPLORER the corresponding node (if there is one) in the main Graph is selected. When nodes are hidden they are grayed out in the CATEGORY EXPLORER. When aggregation occurs all of the nodes being aggregated are hidden and therefore grayed out (but see section 4 for more on this) and the new aggregate node inherits the links of these nodes.

There are two types of indices that the CATEGORY EXPLORER shows: Structural (intrinsic) and Extrinsic. Structural indices are based on analysis of the graph's nodes and arcs such as measuring the degree of nodes and creating spanning trees. All the examples discussed above, in Section 2, are structural indices. In contrast, an extrinsic index is an independent categorization of the nodes that need not be consistent or inferable from the graph (e.g., example, grouping artists by country of birth). The CATEGORY EXPLORER can infer seven structural indices: (A) A simple list of all nodes; (B) Node degree (e.g., all the nodes with two outgoing arcs are grouped together); (C) Breadth first spanning tree; (D) Depth first spanning tree; (E) Strong components (i.e., a strong component is a group of nodes where from any of the nodes you can get to any other of the nodes); (F) Partial order (i.e., an ordering of the nodes based on their connections as in Section 2.2 above); (G) Directed acyclic graph (DAG): This retains as much of the graph as is possible for use as an index.

As this list makes clear, the CATEGORY EXPLORER supports the use of directed acyclic graphs for indices. This is considerably more flexible than supporting tree-based indices. One important difference is that each node in a tree can have at most one parent, while nodes in a DAG can have many. This is important for categorization since DAGs allow things to be multiply categorized, while this is possible with a tree only with great difficulty.

Thus, In terms of the Model-View-Controller paradigm, the CATEGORY EXPLORER model is a DAG, but its View is as a tree. The tree view is defined by "unrolling" the DAG so as to appear to assign to every node a unique parent. This means that a given node may appear multiple times in the CATEGORY EXPLORER View, but represent only a single node in the CATEGORY EXPLORER Model. Accordingly whenever the node is selected all versions of it become selected and similar for de-selecting. Further, the node it corresponds with in the Graph is selected when it is selected. However, note that selecting a node in the

Graph pane does not select it in the CATEGORY EXPLORER. This is done to simplify the relationship between the CATEGORY EXPLORER and the Graph, that is, the former acts as a control for the later, but the GRAPH VIEWER does not control the CATEGORY EXPLORER. It is often helpful to view directly the DAG that the CATEGORY EXPLORER is modeling via a Tree control. This graph can be viewed and manipulated in a GRAPH VIEWER window as was done in Figure 3. Users can also interactively create CATEGORY EXPLORER indices, thus allowing data explorations to be reified for later use and presentation.

4. CATEGORY EXPLORER DILEMMAS

Allowing the CATEGORY EXPLORER index to be a DAG, rather than a tree, creates some user interface design dilemmas. The choices are interesting in that each of them has good and bad aspects and neither seems totally satisfactory. The issue is what happens in the case where a node appears in two categories and then one of the categories is aggregated: Should the node disappear because it is part of the aggregated category or should it be retained since it is also part of a category that has not been aggregated. Note that this question is independent of whether or not we use a Tree View to implement the CATEGORY EXPLORER DAG model. The issues stems directly from the decision to use a DAG as an index and occurs no matter what view is provided.

For example, consider the somewhat artificial example in Figure 5. There *Orange* has been classified as both a *Fruit* and a *Color*. The question is what happens if the *Fruit* category is aggregated? One option is to include *Orange* in the aggregation and to subsequently hide *Orange* in the GRAPH VIEWER. However, this means that *Orange* has vanished from the *Color* category. So, if someone highlighted the members of *Fruit* in the CATEGORY EXPLORER no corresponding *Orange* would appear in the GRAPH VIEWER. Another option is to include *Orange* in the *Fruit* aggregate, but not hide it in the GRAPH VIEWER. This means that when *Fruit* is aggregated it inherits the arcs of *Orange*, but *Orange* still appears as a node. Only after all categories that contain *Orange* have been aggregated would *Orange* be hidden. So, if *Color* were aggregated after *Paint*, then *Color* would inherit the arcs of *Orange* too, but since it was the last non-aggregated category in which *Orange* occurs, *Orange* becomes hidden when it is aggregated. These two designs I call *HideOnAny* and *HideIfAll*, respectively.

The difficulty is that both design choices have strengths and weaknesses. In general, there are conflicting design goals: (A) Keep the information you need close at hand; (B) operations should be reversible and not order dependent; and (C) aggregations should simplify the graph not complicate it.

HideOnAny does a better job of simplifying the graph and it is the information on what will be aggregated is clear and local, but the effect differs based on the order of aggregations and it can be difficult to find the aggregation that has caused a node to be hidden. *HideIfAll* does the same thing independent of the order of operations and it is clear what will happen when you disaggregate, but aggregations can make the graph more complex rather than simpler (e.g., if all the nodes being aggregated are contained in other categories) and it can be difficult to find all the categories that need to be aggregated to hide a node.

5. CONCLUSION AND FUTURE WORK

When the raw material to explore are objects and their relationships a graph representation can provide powerful assistance by reformulating the way search and inference is preformed. Realizing the potential of graph representations for aiding discovery and presentation requires surmounting the complexity that is often present in even small to medium problems. The *Raisin* system helps manage this complexity by giving the means to layout a graph well; to index a graph using categories based on structure and other criteria; to highlight and abstract via selection, hiding, and aggregation; and to create new categories that distill the user's conclusions for analysis and presentation.

Future work will investigate how well users are able to use these facilities for given classification and meaning making tasks. Of particular interest in providing users assistance in designing taxonomies for use in classifying documents in knowledge management systems and for use in semi-automatically building Yahoo-like taxonomies for Web sites. Similarly, future work will investigate how to heuristically aggregate graphs to reduce the graph to a manageable size, yet per the user's cues, retain sufficient detail that the user's task is enabled.

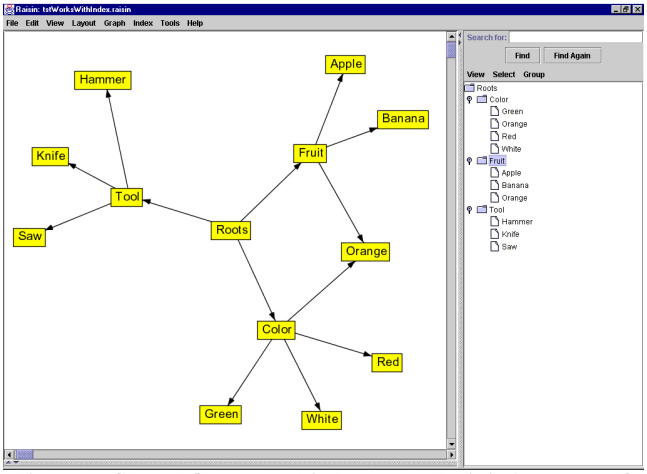

Figure 5. Should Orange be hidden when Fruit is Aggregated?

6. ACKNOWLEDGEMENTS

This work has benefited greatly from many conversations with Robert Farrell. Thanks to Donna Gresh and Bernice Rogowitz for providing the data set on artists' influences.

7. REFERENCES

1. Larkin, J.H. & Simon, H.A. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11*, 65-99, 1987.
2. Gresh, D. & Rogowitz, B. E. Personal Communication.
3. Herman, I., Melançon, G., Marshall, M.S., Graph Visualization and Navigation in Information Visualisation: A Survey. *IEEE Transactions on Visualization and Computer Graphics, 6*(1), pp. 24-43, 2000.
4. Cohen, J.D. "Drawing graphs to convey proximity: an incremental arrangement method." *ACM Trans. Comput.-Hum. Interact. 4*, 3, pp. 197-229. Sep. 1997.