GRAPH-LEVEL OPERATIONS: A HIGH-LEVEL INTERFACE FOR GRAPH VISUALIZATION TECHNIQUE SPECIFICATION

A Thesis
Presented to
The Academic Faculty

by

Charles D. Stolper

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Interactive Computing
Georgia Institute of Technology
December 2016

GRAPH-LEVEL OPERATIONS: A HIGH-LEVEL INTERFACE FOR GRAPH VISUALIZATION TECHNIQUE SPECIFICATION

Approved by:

John Stasko, Advisor
School of Interactive Computing
Georgia Institute of Technology

Duen Horng (Polo) Chau, Co-Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

James Foley
School of Interactive Computing
Georgia Institute of Technology

Rahul Basole
School of Interactive Computing
Georgia Institute of Technology

Alex Endert
School of Interactive Computing
Georgia Institute of Technology

Jarke J. van Wijk
Dept. of Math and Computer Science
Eindhoven University of Technology

Date Approved: 9 August 2016

For Tory,

my sister,

my favorite one.

ACKNOWLEDGEMENTS

I, and therefore this thesis, have benefited from the advice, encouragement, time, and skill of so many over the course of my five years at Georgia Tech, let alone my twenty-three before that. Everyone I am about to mention (and plenty that I had to leave out) deserve far more than my brief thanks here.

Thank you to my parents (Charlie and Christy), my sister (Tory), my grandfathers (Dave and Max), and my family (Ted, Paige, Pierce, Tim, Lori, Tona, and Bob) who have loved, encouraged, and supported me my whole life.

Thank you to the B-105 women (Amy, Mary Katherine, Mary Leah, May Lauren, and Cissy) for adopting me seven and a half years ago and the resulting hugs, smiles, encouragement, and food. Thank you to Sara, Caitlin, Lauren, Justin, Katie, Rachel, Courtney, and Bobby for all the fun. Thank you to Kate, Sam, and the MDP crew for the same. Thank you to Jon and the rest of the CoC/EES frisbee teams over the years and Nolan, Qiyu, Jeremy, and the rest of the pickup gang for the exercise, fun, and an all-school championship. Thank you to Lacy for always listening and for occassionally telling me that I am an idiot. Thank you to Daniel for the conversations about sports, grad school, and engineering. Thank you to Aayush, Guarav, Nasser, Brad, Brad, Cody, and Brendan for being such great roommates over the years. Thank you to Paul, Angelica, J-D, Kim, and Verun for the board game nights. Thank you to Jon for all the Red Sox games. Sam and Mark, you two are my oldest friends, and my life has only been better because of you. Thank you to Paul and Koh for the tech talk, the jokes, and the far-too-infrequent reunions.

Sarah, Ben, Miranda, Emily, Heather, Jess, Casey, David, Mariam, Kim, Catherine, Hank, Chris, Emma, Yacin, Andrew, and the rest of the Happy Hour, Tailgating,

and TSRB crews: Someday I will write a thousand words about how much you all mean to me. You all were my lifeline throughout my entire time in Atlanta. I could never have done this without you.

Thank you to Kayla for being my partner in first-time ethics educating. Thank you to Gabriel for surviving quals with me, snowstorm slumber parties, beautiful lakeside weddings, and giant bear hugs. Thank you to Bri for the lunches, the rants, the high fives, and for joining me in panicking over and then hitting every milestone in the PhD and job search processes. Thank you to Maia for being my secretkeeper and, of course, for *Friends*. Thank you to Emily for never letting me forget that she will always be one of my closest friends. Thank you to Prateek for the puzzles, karaoke, board games, lifting, countless stories, and sprinting to the finish with me. Thank you to Lydia for helping me check off nearly my entire Atlanta bucket list while also getting me to write this thesis. Thank you to my roommate Matt for being an amazing friend and my externalized common sense. Thank you to Arya for the random meals, the random video games, the random conversations, and for the random round of Rock Band that helped my decision to come to Georgia Tech in the first place. Brian, thank you for the football games, baseball talk, caffeine, and always telling me that I could do it. Max, thank you for epic road trips, epic games of telephone tag, and for listening to everything.

Thank you to everyone who has ever shared the Interactive Interfaces Lab (and the GT Vis Lab, when it formed) with me over the years, including Youn-ah, Mengdie, Ramik, Yi, Julian, Jaegul, Tanyoung, Rosa, HP, Hannah, Anand, Sanjay, Bahador, John, Melanie, Alexander, Sakshi, Arjun, Emily, Valentino, Swarnika, Andrew, and Ari. Thank you to those that came before me who were happy to share their advice, wisdom, and friendship: Carsten, Chris, James, Duke, Ji Soo, Scott, and Dorn. Thank you to Leo for being my friend, mentor, and inspiration since Day 1. Thank you to everyone in the Polo Club for Data Science. Special thanks especially to Jerry,

Florian, and Aakash for being wonderful as well as for all of your work with GLOv1 and GLO-STIX. Even more thanks to Brian for the same, and also for helping revise the paper while on your honeymoon. Thank you to the XDATA team (Jeff, Roni, Curt, Arvind, Alex, Alex, Hendrik, and Kris) for your feedback on early designs of GLOs. Thank you to all of my VIS, SIGCSE, and internship friends. Thank you to Joyce for the Summer of NY Baseball. Thank you to Joseph for the discussions on visualization languages, dancing, and getting me a glass of water. Thank you to Eli for the food, the advice, and the hugs. Thank you to Robert for being a friend, a labmate, a sounding board, and a stand-up comic and for the pleasure of sharing five years at Tech with you. Thank you to Alex for being my labmate, my sounding board, my sage, and one of my dearest friends and confidants. Thank you to Hannah for the lifelong friendship that can only be forged in the furnace that is the first years of a PhD.

Thank you to all of my mentors over the years at Thoreau, Sandborn, Fessenden, Concord Academy, and Furman. Thank you to Ben Stumpf for the advice, especially to write in a text editor with spell-check turned off. Thank you to Bill Adams for teaching me Java and the basics of computer science and starting me on this crazy path. Thank you to Craig Caldwell, John Barrington, Tim Fehler, David Spear, and Lane Harris for the classes, academic life conversations, and grudging acceptance that I was not going to get a PhD in History. Thank you to Kala Kennemore for all of the help and smiles. Thank you to Chris Healy, Bryan Catron, John Harris, and Mark Woodard without whose excellent computer science and math education I could never have attended Georgia Tech, let alone done the work I have done. Thank you to Kevin Treu for his mentorship and friendship, as much during my time at Furman as since I graduated. A huge thanks to Hayden Porter for pulling me aside in October of my freshman year and recruiting me to do a research project with him. It was related to graph visualization. I clearly still haven't looked back.

Thank you to Stuart Rose, Court Corley, and the Visual Analytics group at PNNL, Adam Perer, David Gotz, and the Healthcare Analytics group at IBM, Bongshin Lee, Nathalie Riche, and the CUE and neXus groups at MSR. Thank you for the mentorship, advice, brainstorming, and patience. I learned so much from each of you.

Thank you to Chris Collins and Shixia Liu and the rest of the InfoVis DC participants for all of their advice on what would become this work.

Thank you to all of faculty and staff at Georgia Tech for five incredible years. Thank you to Monica Ross and Wanda Purinton, who are the nicest women in the world and without whose talent and skill I am convinced IC would not function. Thank you to Jacob Eisenstein, who taught me everything I know about machine learning. Thank you to Keith Edwards for the smiles, the jokes, the font nerd talk, and for declaring me Qualified. Thank you to Mark Guzdial for his mentorship in how to teach computer science and for breaking the news early that I got into Tech. Thank you to Amy Bruckman for being a passionate faculty voice for students. Thank you to Annie Anton for her leadership, zeal, energy, and for inviting me into her circle of councilors.

Finally, my most important thanks go to my patient, supportive, and brilliant thesis committee. Jack, thank you for your dry sense of humor, spot-on probing questions, and for finding me at the VIS reception in Paris to talk about GLOs. Rahul, thank you as much for our conversations about soccer as our conversations about network visualization. Alex, you could have been in any number of prior sections here: from VIS friend, to internship mentor, to professor, and finally thesis committee member. Thank you for being an incredible friend and mentor to me through every stage of our relationship. Jim, you have been one of my greatest advocates during my entire tenure at Tech. Thank you for sharing your wisdom, experience, and humor with me. Polo, you are caring and honest and I cannot

imagine the last three years without you. Thank you so much for inviting me out to Starbucks to talk about a new way to describe graph visualization transitions. My relationship with John can be summed up by the first time we ever met when I was looking at grad school programs: we spent fifteen minutes talking about aspects of visualization research I was interested in, fifteen minutes talking about aspects of visualization research he was interested in, and fifteen minutes talking about the Red Sox and Braves. John, thank you for everything.

Research for this thesis was funded by the National Science Foundation under Grants No. IIS-1320537 and IIS-1563816 and the XDATA program sponsored by DARPA and the Air Force Research Laboratory (AFRL). Prior portions of my PhD were funded by the National Science Foundation under Grant No. CCF-0808863 and a Department of Homeland Security Ph.D. Fellowship in Data Analysis and Visual Analytics. The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

TABLE OF CONTENTS

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **xii**

LIST OF FIGURES . . . . . **xiii**

I INTRODUCTION **1**

1.1 Term Definitions 5

1.2 Thesis Statement and Research Questions 8

1.3 Contributions and Impact 9

II BACKGROUND **11**

2.1 Visualization Toolkits and Grammars 11

2.2 Systems for Applying Techniques to Data 17

2.3 Systems for Specifying Visualization Techniques 19

2.4 Visualization Design Space Analysis 21

III A METHOD FOR GLO MODEL INDUCTION **22**

3.1 Seed Technique Identification 23

3.1.1 GLOv1 Identification 24

3.1.2 GLOv2 Identification 25

3.2 Transitions, GLOs, and the Transition Matrix 28

3.2.1 Semantic Substrates to PivotGraph 29

3.2.2 Force-Directed Layout to Matrix Plot 35

3.2.3 Graph-Level Operations and the Transition Matrix 41

3.2.4 Handling GLO Uncertainty 43

3.3 Inducing an Expected Data Model, Model of Visual Elements, and Set of Graph-Level Operations from the Transition Matrix 51

3.4 Augmenting the Operations Set 53

IV THE GRAPH-LEVEL OPERATIONS MODEL 55
4.1 Graph Data Model 55
4.2 GLO Visual Element Model 58
4.2.1 Glyphs 58
4.2.2 Generations 64
4.2.3 Canvases 68
4.2.4 GLO Display 69
4.3 Operation Sets 71
4.4 Language Properties of Graph-Level Operations 72
4.5 Differences Between GLOv1 and GLOv2 77
4.6 Specifying Techniques Using GLOs 80
V UTILITY OF GRAPH-LEVEL OPERATIONS 85
5.1 Easing the Engineering Challenge 85
5.1.1 Implementations 88
5.1.2 GLO-STIX 89
5.1.3 GLO-CLI 92
5.2 Enabling A Deeper Understanding of Techniques 93
5.2.1 Feature Space Analysis 94
5.2.2 GLO Distance 101
5.3 Identifying New Techniques 102
5.3.1 Approximate Measures of GLO Expressiveness 109
VI CONCLUSION 112
6.1 Contributions and Impact 112
6.2 Limits of GLOs 113
6.3 Future Research Directions 116
APPENDIX A — GLOV1 SEED TECHNIQUES 121
APPENDIX B — GLOV2 SEED TECHNIQUES 127
APPENDIX C — GLOV1 OPERATIONS SET 176

APPENDIX D — GLOV2 OPERATIONS SET 182

APPENDIX E — GLOV2 LITERATURE REVIEW RESULTS . 193

APPENDIX F — HIERARCHICAL CLUSTERINGS 200

REFERENCES 237

LIST OF TABLES

1 GLOv2 seed techniques 28

2 GLOv2 Constants 77

3 GLOv2 operations equivalent to GLOv1 operations. 80

4 Inverse GLOs required for GLOv2 GLOs. For each GLO in the first column that the technique specification contains, the corresponding inverse GLO(s) in the second column must be applied to return to the null state. 83

5 Number of GLOv2 seed technique specifications (out of 29) containing each GLOv2 operation. 84

6 GLOv2 operations that do not appear in any GLOv2 seed technique specifications. 84

7 Results of hierarchically clustering technique vectors created by ignoring optional parameters (no-flags). 96

8 Results of hierarchically clustering technique vectors with optional parameters (flags). 97

9 Results of hierarchically clustering technique vectors created by adding features for optional parameters (flags-xtra). 97

10 GLOv2 specification for modified NodeTrix display in Figure 62b. . . 106

11 GLOv2 specification for 'GLO' teaser technique in Figure 1. 108

12 GLOv2 operations unique to a single seed technique. 114

LIST OF FIGURES

1 Graph-Level Operations 1

2 Arc Diagram rendered in GLO.js. 3

3 Novel graph visualization technique combining the layout of PivotGraphs [244] with the interaction of Semantic Substrates [96]. The nodes in the graph (characters in a book) are aggregated by cluster label (rows) and gender (column). Edges are clustered by the cluster and gender of the source and target endpoint nodes and colored by the target node's cluster. These aggregated edges are only shown when the analyst mouses over an endpoint node. 4

4 GLOv2 EdgeMap B seed technique from [74]. 24

5 First example transition techniques 29

6 GLOv2 GraphDice seed technique from [36]. 32

7 Unmodified Semantic Substrates representation. 32

8 Semantic Substrates representation modified to show all edges. 33

9 Semantic Substrates representation modified to show all edges and position nodes on x by the discrete gender attribute. 33

10 Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, and show x axis labels. . 34

11 Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, and aggregate nodes into super-nodes. 35

12 Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, and size the super-nodes by the number of nodes they represent. 36

13 Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, size the super-nodes by the number of nodes they represent, and aggregate the edges by the edges' source gender, source cluster, target gender, and target cluster. 37

14 Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, size the super-nodes by the number of original nodes they represent, aggregate the edges by the edges' source gender, source cluster, target gender, and target cluster, and size the super-edges by the number of original edges they represent. In other words, a PivotGraph representation of the graph. 38

15 Second example transition techniques 38

16 Unmodified Force Directed Diagram representation. 39

17 Abstract Force Directed Diagram representation after evenly distributing nodes on x (a) and after evenly distributing nodes along the x axis sorted by cluster and aligning the nodes at the top of the display. . . 39

18 Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, and cloning the set of node glyphs. 40

19 Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, and evenly distributing the new nodes on y sorted by cluster without (a) and with (b) an inverted axis. 41

20 Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y sorted by cluster with an inverted axis, and aligning the new nodes to the left of the display. 42

21 Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y sorted by cluster with an inverted axis, aligning the new nodes to the left of the display, and drawing edges from the second set of nodes (on the left) to the first set of nodes (on top). 43

22 Abstract Force Directed Diagram representation after evenly distributing nodes on x, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y with an inverted axis, aligning the new nodes to the left of the display, drawing edges from the second set of nodes (on the left) to the first set of nodes (on top), and displaying edges as squares. In other words, the target abstract Matrix Plot representation. 44

23 Sample abstract techniques where ($*$, technique) transition matrix entries share operations. 45

24 Four abstract techniques where (∗, technique) transition matrix entries contain the *evenly distribute nodes on* {axis} GLO with different mandatory parameters and optional parameters. 46

25 Abstract Arc Diagram representations with different optional sorting parameters. 47

26 Representations with and without the optional within parameter. . . 48

27 Sample abstract techniques where (∗, technique) transition matrix entries use or do not use the group-by optional parameter. 49

28 Sample abstract techniques where (∗, technique) transition matrix entries use or do not use the group-by optional parameter including bounding boxes determined by the Circle Graph node positions. . . . 49

29 Sample abstract techniques where (∗, technique) transition matrix entries use a group-by optional parameter to display intra-group edges differently than inter-group edges. 50

30 The GLOv2 *position nodes by* {attr} operation positions node glyphs evenly along the axis with discrete parameters and relatively along the axis with continuous parameters. 52

31 GLOv2 GMap seed technique from [94]. 53

32 GLOv2 CiteVis seed technique based on [213]. 54

33 Demonstration of axis uncertainty. 54

34 GLOv2 seed techniques with differing node and edge glyph displays. . 59

35 GLOv2 EdgeMap B seed technique from [74]. 59

36 GLOv2 Citevis seed technique based on [213]. 61

37 GLOv2 Force-Directed Layout seed technique and GLOv2 GMap seed technique. The straight-line edge glyphs in the Force-Directed Layout have the show all edges interaction mode, while the straight-line edge glyphs in the GMap technique have the show faded interaction mode. 62

38 GLOv2 Semantic Substrates seed technique from [204] with edge glyphs in the show incident edges display mode. 62

39 GLOv2 List View seed technique from [199]. Edge glyphs utilize the show faded-and-incident interaction mode. 63

40 GLOv2 EdgeMap A and Edgemap B seed techniques from [74]. Both techniques utilize the in-out edges interaction mode where in edges of the selected node are displayed differently from out edges. 64

41 GLOv2 Edge-Label-Centric seed technique from [182]. The red straight-line edges are drawn from the source generation on the left to the target generation on the right through the waypoint generation of super-edge glyphs in the center. 67

42 GLOv2 MatLink seed technique from [118] demonstrates source and target node generations. 67

43 GLOv2 ScatterNet seed technique from [27] demonstrates axis labels. 69

44 GLOv2 MatrixExplorer seed technique from [116] demonstrates canvas partitioning within a GLO Display. 69

45 GLOv2 Attribute Matrix seed technique from [153] demonstrates filter-partitioning canvases and meta-axis labels. 70

46 GLOv2 GMap seed technique from [94] utilizing convex hulls. 72

47 GLOv2 DOSA seed technique from [232] utilizes the all-canvases optional parameter. 74

48 *Evenly distribute nodes on* $\{x\}$ with and without a within attribute. . 75

49 *Align nodes* {center} with and without a group-by attribute. 76

50 (a) Force-Directed Layout, (b) Force-Directed Layout after applying *hide edges*, (c) Force-Directed Layout after applying *hide edges* and *show all edges (group-by:* {cluster}*)* 76

51 (a) GLOv1 Adjacency Matrix seed technique with circles for edges and (b) equivalent Adjacency Matrix in GLOv2 with squares for edges. . . 79

52 Force-Directed Layout, Matrix Plot, and EdgeMap A techniques rendered in GLO.js. 81

53 Visualization software stack 86

54 The GLO-STIX interface. 90

55 The GLO-CLI interface. 92

56 Dendrogram results for three hierarchical clustering using three vectorization methods, Hamming distance, and average cluster comparison rendered using Matplotlib [129]. 99

57 GLOv2 NodeTrix seed technique from [117]. 100

58 GLOv2 seed techniques clustered by symmetric transition distance rendered with GLO.js. On the left, edges are colored by the one-way transition distance. On the right, edges are colored by the symmetric sum of the transition distances. Rendered using GLO.js 102

59 Clusters visible in the symmetric GLO Distance matrix. 103

60 Novel graph visualization technique combining the layout of PivotGraphs [244] with the interaction of Semantic Substrates [96]. 104

61 Example of a minor tweak to a seed technique by changing the Force-Directed Layout seed technique's edge display mode to curved lines. . 105

62 NodeTrix [117] GLOv2 seed technique and modified NodeTrix display with nodes colored by cluster, intra-cluster edges colored by a constant, faded intra-cluster edges, and highlighting intra-cluster edges incident to a selected node created using GLO.js. 105

63 Arc Diagram [243] created using GLOv1 within GLO-STIX. 107

64 Matrix Plot and MatLink [118] techniques rendered in GLO.js. Transitioning between these two techniques is more efficient without transitioning through an intermediate null state. 114

65 Single technique defined using two distinct specifications. 115

66 Examples of edge bundling from [120]. 118

67 Bring-and-Go interaction from [166]. 119

68 GLOv1 force-directed layout seed technique. 121

69 GLOv1 circle plot seed technique. 122

70 GLOv1 scatterplot seed technique. 123

71 GLOv1 semantic substrates [204] seed technique. 124

72 GLOv1 PivotGraph [244] seed technique. 124

73 GLOv1 adjacency matrix seed technique. 125

74 GLOv2 Force-Directed Layout seed technique from [138]. 128

75 Force-directed layout [138] rendered in GLO.js. 128

76 GLOv2 Matrix Plot seed technique from [34]. 129

77 Approximate matrix plot [34] rendered in GLO.js. 129

78 GLOv2 Cluster Circles seed technique from [69]. 131

79 Cluster circles [69] rendered in GLO.js. 131

80 GLOv2 Circle Graph seed technique from [207]. 132

81 Circle graph [207] rendered in GLO.js. 132

82 GLOv2 GeneVis A seed technique from [23]. 133

83 Genevis A [23] rendered in GLO.js. 134
84 GLOv2 GeneVis B seed technique from [23]. 135
85 Approximate Genevis B [23] rendered in GLO.js. 135
86 GLOv2 Arc Diagram seed technique from [141]. (Specifically the 'contributor coloring' subfigure.) 136
87 Arc diagram [141] rendered in GLO.js. 136
88 GLOv2 Matrix Browser seed technique from [262]. 137
89 GLOv2 Matrix with Bars seed technique from [205]. 139
90 GLOv2 MatrixExplorer seed technique from [116]. 140
91 Approximate MatrixExplorer [116] rendered in GLO.js. 141
92 GLOv2 NetLens seed technique from [140]. 143
93 GLOv2 Semantic Substrates seed technique from [204]. 145
94 Semantic Substrates [204] rendered in GLO.js. 145
95 GLOv2 PivotGraph seed technique from [244]. 146
96 PivotGraph [244] rendered in GLO.js. 147
97 GLOv2 MatLink seed technique from [118]. 148
98 Approximate MatLink [118] rendered in GLO.js. 148
99 GLOv2 List View seed technique from [199]. 150
100 List view [199] rendered in GLO.js. 151
101 GLOv2 Edge-Label-Centric seed technique from [182]. 152
102 GLOv2 Honeycomb seed technique from [106]. 154
103 Approximate Honeycomb [106] rendered in GLO.js. 154
104 GLOv2 GraphDice Segment seed technique from [36]. 156
105 GraphDice segment [36] rendered in GLO.js. 156
106 GLOv2 GraphDice seed technique from [36]. 157
107 Approximate 3x3 GraphDice [36] rendered in GLO.js. 157
108 GLOv2 GMap seed technique from [94]. 159
109 GLOv2 Attribute Matrix seed technique from [153]. 160
110 GLOv2 EdgeMap A seed technique from [74]. 161

111 Approximate EdgeMap A [74] rendered in GLO.js. 162
112 GLOv2 EdgeMap B seed technique from [74]. 163
113 Approximate EdgeMap B [74] rendered in GLO.js. 163
114 GLOv2 Hive Plot seed technique from [144]. 164
115 Hive Plot [144] rendered in GLO.js. 165
116 GLOv2 Hive Panel seed technique from [144]. 166
117 2x3 Hive Panel [144] rendered in GLO.js. 166
118 GLOv2 ScatterNet seed technique from [27]. 168
119 ScatterNet [27] rendered in GLO.js. 168
120 GLOv2 Citevis seed technique based on [213]. 169
121 Citevis [213] rendered in GLO.js. 170
122 GLOv2 DOSA seed technique from [232]. 171
123 Approximate DOSA [232] rendered in GLO.js. 171
124 GLOv2 NodeTrix seed technique from [117]. 173
125 Approximate NodeTrix [117] rendered in GLO.js. 173
126 SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and Hamming distance. 201
127 SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and Hamming distance. 202
128 SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and Jaccard distance. 203
129 SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and Hamming distance. 204
130 SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and cosine distance. 205
131 SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and Jaccard distance. 206
132 SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and Jaccard distance. 207
133 SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and cosine distance. 208

134 SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and cosine distance. 209

135 SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and Hamming distance. 210

136 SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and Jaccard distance. 211

137 SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and cosine distance. 212

138 SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and Hamming distance. 213

139 SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and Hamming distance. 214

140 SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and Hamming distance. 215

141 SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and Hamming distance. 216

142 SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and cosine distance. 217

143 SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and cosine distance. 218

144 SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and Jaccard distance. 219

145 SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and Jaccard distance. 220

146 SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and Jaccard distance. 221

147 SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and cosine distance. 222

148 SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and Jaccard distance. 223

149 SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and cosine distance. 224

150 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and Hamming distance. . 225

151 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and Hamming distance. 226

152 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and Hamming distance. 227

153 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and cosine distance. . . . 228

154 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and Jaccard distance. . . 229

155 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and cosine distance. . . 230

156 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and Jaccard distance. . 231

157 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and Hamming distance. . . 232

158 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and cosine distance. . . 233

159 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and Jaccard distance. . 234

160 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and Jaccard distance. . . . 235

161 SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and cosine distance. 236

CHAPTER I

INTRODUCTION

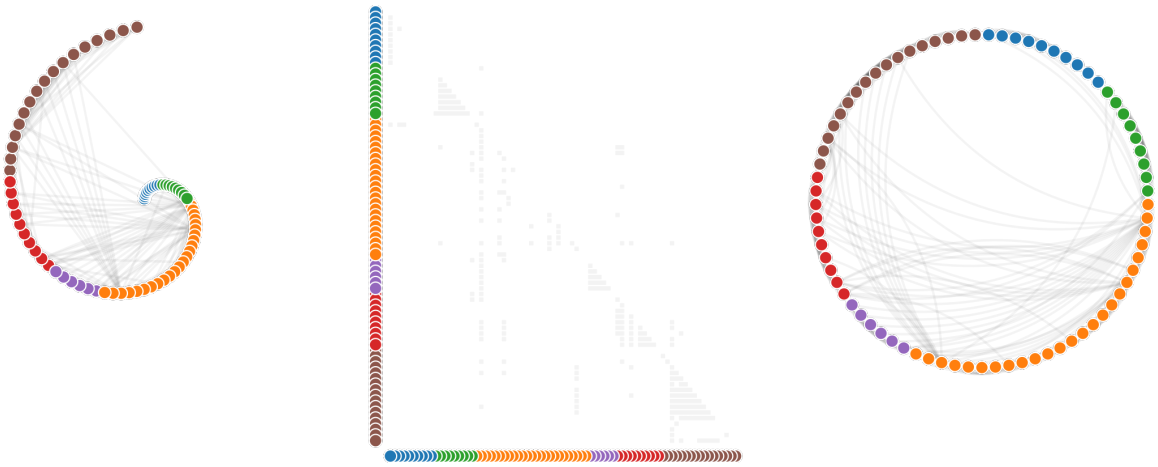


Figure 1: Graph-Level Operations

More and more the world is being described as networks—as connections between people, places, and ideas. Businesses share technology and ideas [27], friends communicate with each other [108], and authors frequently use pairs of words together [235]. Networks (or graphs, as the data structure is often called) provide a richer model than simply understanding each item in isolation. Identifying corporations that produce a critical component, or mutual friends amongst acquaintances, or topically-related terms are all key tasks for those analyzing, exploring, or studying these domains and all benefit from an understanding of the underlying connections. Meanwhile, data visualization has long provided tools for better making sense of data [52, 84]. Visualization harnesses the human visual processing system's ability to rapidly translate representation into cognition. Graph visualization, as one might expect, applies the tools and methods of visualization to graph data [119].

No single representation of a graph (visual or otherwise) is the perfect fit for every

graph analysis task. Over the years, researchers have developed a wide variety of graph visualization techniques for helping analysts solve their wide variety of analysis tasks. Though the breadth of techniques represents solutions to a panoply of tasks, it introduces a new issue: complexity. So many techniques introduces complexity in comparing techniques in an objective way and engineering complexity of implementing so many techniques. In this dissertation, I introduce a class of models for graph visualization, graph-level operations models (or GLO models) [216], as an elegant solution to this complexity. The crux of a GLO model is recognizing that there are common features among graph visualization techniques. Similar to understanding a watch by taking it apart, by identifying these commonalities we can better understand the techniques themselves. We can also use these shared properties to abstract away many of the details usually required to specify techniques, instead recognizing an effective "mid-level" between low-level graphics code and high-level techniques.

A graph-level operations model (GLO model) is a domain-specific language (DSL) defined around these commonalities. A GLO model consists of two parts: a model of the visual elements of graph visualization and a set of functions (called graph-level operations, operations, or GLOs) for manipulating those elements. Each operation encapsulates a manipulation of some aspect of a graph visualization. Such aspects include:

- the position of node and edge glyphs
- visual properties of glyphs (e.g. color and size)
- glyph interactivity (e.g. highlighting incident edges or neighbors of a selected node)
- the underlying data (e.g. through aggregating nodes or edges)

For example, one operation may position all of the glyphs representing the nodes along an axis based upon an attribute of the nodes; another might adjust all of the edge glyphs to use counter-clockwise Bézier curves rather than using straight lines

between the nodes associated with the edge; a third might be to apply a force-directed layout algorithm to the node glyphs.

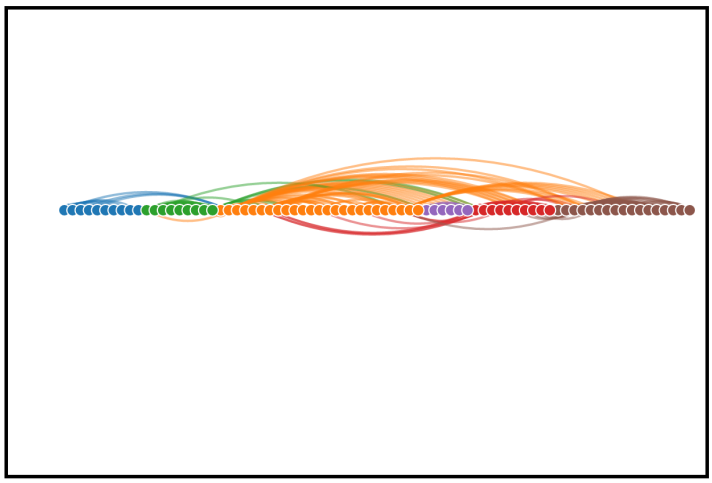


Figure 2: Arc Diagram rendered in GLO.js.

Using GLOs, one can then define graph visualization techniques using ordered lists of these operations. For example, the Arc Diagram [243, 141] depicted in Figure 2 is defined as:

- display nodes as circles
- size nodes by constant
- color nodes by {*node-color-attr*}
- align nodes {*middle*}
- display edges as curved lines
- size edges by constant
- color edges by {*target.node-color-attr*}
- show all edges
- evenly distribute nodes on x (sort-by:{*sort-attr*})

These software-environment-independent definitions can then be used by developers to simplify adding additional visualization techniques to graph analysis systems,

thereby providing analysts with a wider variety of tools and solutions. These definitions also enable researchers exploring the space of graph visualizations to have a concrete means of comparing techniques, which simplifies identifying clusters amongst techniques or identifying aspects of visualizations that correlate to better results on analysis tasks. Finally, having a set of building-blocks enables visualization researchers to identify novel, effective graph visualization techniques. As a very simple example, one can combine the grid-based layout, node and edge aggregation, and curved edge display of a PivotGraph [244] with the interaction of only showing incident edges of Semantic Substrates [204] to create a novel visual representation to see interactions between clusters while minimizing the occlusion of edges (Figure 3).

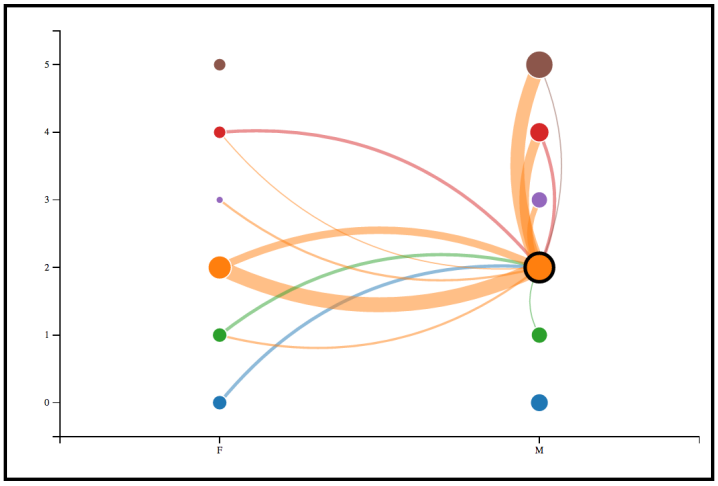


Figure 3: Novel graph visualization technique combining the layout of PivotGraphs [244] with the interaction of Semantic Substrates [96]. The nodes in the graph (characters in a book) are aggregated by cluster label (rows) and gender (column). Edges are clustered by the cluster and gender of the source and target endpoint nodes and colored by the target node's cluster. These aggregated edges are only shown when the analyst mouses over an endpoint node.

This dissertation presents the methods and results of my work inducing two graph-level operations models from sets of canonical graph visualization seed techniques. I presented the initial work on GLOs at IEEE Infovis 2014 [216]. In that work, I and my colleagues identified the model necessary to specify 6 hand-picked seed techniques. In this dissertation, I refer to the results of that work as **GLOv1**. In subsequent

research, I performed a similar induction process with a set of 29 seed techniques resulting from a literature review of 430 graph visualization publications. I refer to model resulting from that work as **GLOv2**.

For the remainder of this chapter, I will specifically define common terminology that I will be using throughout this dissertation. I will then state the thesis of my research and the research questions that I addressed over the course of this work. Finally, I will present the contributions the work presented here.

1.1 Term Definitions

Before I move forward, let me specify terminology I will be using throughout this dissertation. For those that work with graphs and with visualization toolkits, these definitions will be standard, but I include them here for completeness. In a few cases, this is to clarify which definition I will be using when there are differing common interpretations of a term.

- **Graphs** are a data structure consisting of a set of **nodes** (or **vertices**) and a set of relationships between these nodes called **edges** (or **links**). The nodes associated with an edge are called **endpoints** of the edge. A **subgraph** is a subset of these nodes and a subset of these edges. Graphs can be **directed** (each edge has a distinct **source** node and **target** node) or **undirected** (each edge has two associated nodes, but the order does not matter). In the case of directed graphs, an edge is an **out-edge** of the source and an **in-edge** of the target. In the case of undirected graphs, each edge is both and in-edge and an out-edge of its associated vertices. The **in-nodes** of a vertex are the source nodes of all of the vertex's in-edges and the **out-nodes** are all of the target nodes of the vertex's out-edges. The **neighborhood** of a vertex is the union of the vertex's in-nodes and out-nodes. The **degree** of a node is the size of its neighborhood, while its **in-degree** and **out-degree** are the sizes of its in-node

set and out-node set, respectively. A **self-edge** is an edge where both endpoints are the same node. Nodes and edges might have **attributes** (or **properties**) that define more information about the data.

- A **visualization technique** consists of the method for mapping **raw data** to **visual primitives** (or **glyphs** or **marks**) and positioning those primitives into a **representation** (or **visualization**) as well as a specification for any **interactive capabilities**. For example, a bar chart creates a rectangular glyph for each data item and maps the length of that bar to a chosen quantitative attribute of the data item. The bars are aligned along a single axis and then evenly distributed over the perpendicular axis. A bar chart often does not have any interaction associated with it, though an alternative technique might include highlighting the bar when the user hovers over it. A second alternative might show a tooltip when the user mouses over the bar. A **graph visualization technique** is a visualization technique where the input data is expected to be a graph data structure.
- An **Embedded Domain-Specific Language** is a focused API built upon an existing programming language (called the **host** or **host language**). While the specifics can differ, the embedded language often uses the host language's operators and syntax, but with additional method names, operators, or reserved words. In other cases, such as regular expressions or number formatting languages, the embedded language is independent of the host language, except that the parser for the embedded language is implemented in the host language in order to interact with the rest of the code. A GLO model is of the latter variety: a set of functions that each implementation can adapt to the host language's conventions.
- **Low-level graphics code** manipulates the graphical primitives of a host language's graphics system. For example, manipulating Scalable Vector Graphics

(SVG) elements in a web-based visualization system or Swing elements in a Java-based visualization system. This code is written in the host language. **Low-level visualization toolkits** are domain-specific languages for manipulating these primitives. These languages provide a layer of abstraction, simplifying the code necessary to specify aspects of a visualization. Examples include D3.js and prefuse.

- The **expressiveness** of a language or model describes how much the language or model can represent. In other words, what can the language or model define and what cannot be defined using the model.
- A language is **portable** if code written in the language can be run in a variety of environments. Since I will be discussing an embedded domain-specific language, I will be referring to portability between host software environments such as Javascript with SVG graphics, Javascript with Canvas graphics, Java with Swing graphics, or python with tcl/tk graphics. In other words, portable DSL code be run on interpreters written in a variety of host languages.
- Finally, throughout the visualization software stack are a variety of different classes of user, of which three are notable with respect to GLOs. A **programmer** or **developer** writes the code that will either generate visualizations or the software that presents visualizations within a larger application. A **designer** defines visualization techniques, i.e. specifies how data is mapped to marks. An **analyst** is someone who wishes to generate insights from a collection of data. In other words, an analyst is a user of a visualization. These three categories are not mutually exclusive. Some designers are skilled programmers; some analysts have strong design skills; many analysts have programming experience. However, when discussing various tasks, I will be specific about which type of user needs to accomplish that task and target the solution to them.

1.2 Thesis Statement and Research Questions

Thesis Statement The graph-level operations (GLO) model consists of a visual element model and a set of high-level functions that harness commonalities between graph visualization techniques. This model enables effectively describing graph visualizations, simplifying graph visualization engineering, more deeply understanding graph visualization techniques, and discovering novel graph visualization techniques.

RQ1 What are the elements of a portable, expressive, high-level model for graph visualization? In Chapter 3, I describe a method for inducing a high-level model for graph visualization (a GLO model) from a set of canonical graph visualization seed techniques and describe two such sets of seed techniques. In Chapter 4, I describe the resulting GLO models (GLOv1 and GLOv2) each consisting of a model of visual elements and a set of operations for manipulating those elements, and I define how to use GLOs as a specification language for graph visualization techniques.

RQ2 What is the utility of a such a model? In Chapter 5, I describe three use cases of a GLO model: simplifying graph visualization engineering, more deeply understanding the design space of graph visualization techniques, and identifying novel graph visualization techniques. In Chapter 6, I consider further applications of GLO models and research opportunities introduced by GLO models.

RQ3 Does the model work in practice as well as in theory? In Chapter 5, I describe Javascript and SVG implementations of the GLOv1 and GLOv2 models and describe two applications built using the implementations (GLO-CLI and GLO-STIX).

RQ4 What are the bounds on the model's expressiveness? In Chapter 4, I show how a GLO model can express its seed techniques. In Chapter 5, I describe how

GLOs can express novel techniques and I approximately quantify the expressiveness of the GLOv1 and GLOv2 models and my GLO.js Javascript implementation of GLOv2. In Chapter 6, I briefly discuss possible limits on the expressiveness of the GLO model and the generalizability of the model.

1.3 Contributions and Impact

This dissertation contributes to (and provides a direct impact on) five facets of graph visualization research and practice: models, analysis methods, software, techniques, and education.

Models I present a novel class of graph visualization model, the graph-level operations model (GLO model). I introduce a method for inducing a model from a set of canonical seed techniques in Chapter 3 as well as two instances of GLO models (GLOv1 and GLOv2) and a means of defining techniques using the model in Chapter 4. I include definitions for the 6 hand-picked GLOv1 seed techniques and the 29 GLOv2 seed techniques selected by means of a review of 430 graph visualization publications in Appendices C and D. All of the following additional contributions stem from the GLO model.

Analysis Methods The GLO model represents a giant leap forward in our ability to easily and effectively compare and cluster graph visualization techniques. To demonstrate this, I introduce GLO-based methods for reducing techniques to vector representations as well as a novel GLO-based distance metric for techniques in Chapter 5. Furthermore, GLO models open numerous avenues for future research into the design space of graph visualization. I list a sample of these avenues in Chapter 6.

Open-Source Software GLO models significantly reduce the engineering overhead of incorporating a wide range of graph visualization techniques into graph analysis

software. I have developed the GLO.js graph visualization toolkit to easily incorporate a large variety of graph visualization techniques into web-based graph analysis software. I have used the toolkit to build the GLO-STIX GUI application and GLO-CLI command-line application for using visualization to explore a network. These three packages are described in Chapter 5 and are available as open-source software at `https://github.com/chadstolper/glo`.

Techniques Graph-level operations greatly simplify identifying novel graph visualization techniques. Throughout this dissertation (especially in Chapter 5), I introduce a number of novel techniques and provide their definitions using GLOs.

Education Graph-level operations have the potential to revolutionize graph visualization education through demonstrating the variety and interconnectedness of graph visualization techniques. The model could also have a sizable impact on discrete mathematics education through enabling demonstrations of graph theory properties and algorithms. I briefly discuss this potential in Chapter 6.

CHAPTER II

BACKGROUND

Because of the effect that graph-level operations (GLOs) have throughout the visualization development stack, the related work for this this covers a wide breadth of visualization research. The set of operations and GLO.js implementation fall under visualization systems, or toolkit, research. The applications that I have built on top of the GLO layer fit a variety of visualization application categories, including visualization creation systems and graph analysis tools. The analysis I conduct using GLOs comparing and clustering graph visualizations falls under visualization design space analysis research.

I am going to limit my discussions to just those systems that draw visualizations from data. In other words, while a skilled artist could likely create any static visualization using Microsoft Paint, Adobe Illustrator [8], or (for graphs) Microsoft Visio [165], I will not be including these in my discussion.

2.1 Visualization Toolkits and Grammars

Graph-level operations are intended to act as a semantically-meaningful middle-layer between the low-level graphics environment and top-level applications. Thus, it is important to understand the state-of-the-art at the low-level, in order to understand why having such a middle-layer is worthwhile.

Some of the earliest work on a visualization language was Mackinlay's APT system [159]. Mackinlay was attempting to automate the choice of visualization based upon the structure of the (tabular) input data. In doing so, he built a framework for describing visualization techniques, especially bar charts and scatterplots. The

work also introduces concepts relevant to any visualization framework: expressiveness (what can the framework represent) and composition (can techniques be combined, such as a scatterplot matrix or node-link diagram of matrix displays ala NodeTrix [117]).

Building off of this foundational work, a number of visualization toolkits have been developed. In most cases, the designers of these toolkits set out to abstract away the graphical details of the host language they were implemented in. One of the more influential toolkits was Heer et al.'s Prefuse [113]. Prefuse was built in contrast to prior toolkits such as the Infovis Toolkit [83] and the XML Toolkit [30]. Those systems allowed a developer to incorporate specific techniques into their applications, but did not enable him or her to implement new techniques. Prefuse enabled the developer to implement new techniques either through composition or from raw primitives and marks. Prefuse was designed around Card, Mackinlay, and Shneiderman's reference model [52]: raw data is *data transformed* into data tables that are then *mapped* onto visual structures that are then *view transformed* into views. Prefuse was built on top of Java and Java2d, though it was eventually ported to ActionScript/Flash as well (the port was called flare [111]). Prefuse contained a variety of components for creating graph and tree visualizations, including force-directed layouts, circular layouts, physics simulation modules, and view distortions such as fisheye distortions. The social network visualization system Vizster [108] is one of the graph systems built using the toolkit.

JUNG [173] is another Java-based library. Rather than being designed primarily for visualization, JUNG was designed to be a general purpose graph library. JUNG (and similarly the python library NetworkX [105]) has a large focus on managing the graph data structure itself. Unlike NetworkX, however, JUNG does have a strong visualization component for easily enabling node-link diagrams within Java2d or Java3d

with various pre-defined layout algorithms. A developer still has the freedom to implement their own layout algorithms on top of the framework as well, or even write their own visualization code and just use the underlying graph model. While NetworkX does not support visualization itself, the popular python visualization library Matplotlib [129] does. Matplotlib (a library originally based on Matlab's plotting functionality [160]) supports drawing node-link diagrams and adjacency matrices of NetworkX graph structures.

Adar's Gython language [7] (a component of the GUESS system) is similar to JUNG in being a toolkit for managing graph data structures and visualizing them. Gython is built on top of Jython (python implemented in Java). Gython includes a wide range of syntax support for graph-specific queries such as alice<->bob, alice->bob, alice<-bob, and alice?bob to select (respectively) all undirected or bidirected edges between the nodes (or node-groups) alice and bob, all edges from alice to bob, all edges from bob to alice, and all edges between alice and bob regardless of directionality. Gython also (explicitly) differs from JUNG and prefuse in treating visual properties and data properties equivalently. The developer can adjust or check the value of the node or edge in the same manner as he or she can adjust or check the color or size of the glyph representing the node or edge.

With the trend of moving away from desktop applications to web applications, the developers of prefuse and flare set out to write a Javascript- (as opposed to Java-) based visualization toolkit. Bostock and Heer's first Javascript toolkit was Protovis [40]. Soon after, they recognized a number of issues with Protovis, and ceased development in favor of D3.js [41]. Protovis used a set of custom mark types, whereas D3 was designed to be general and work with existing web technologies, especially Scalable Vector Graphics (SVG) [240], the W3C-standardized XML vector graphics format. D3 allows the developer to map arrays of data to elements in the webpage's document object model (or DOM) and set properties of those elements

based on the data. In addition to this base functionality, D3 includes a number of helper functions and layouts such as numerical and color scales, code for easily creating axes, and force-based graph layouts.

These toolkits (prefuse, flare, JUNG, Gython, Protovis, D3, and Matplotlib) are all embedded domain-specific languages (or DSLs) [128] for visualization. In this role, they all provide abstractions beyond what their host languages (respectively: Java, ActionScript, Java, Jython, Javascript, Javascript, and python) provide for developers incorporating visualization into their applications or building standalone visualization applications. However, each of these systems is very closely entangled with the host language. Prefuse is written to make sense to a Java developer and depends on features built into the Java language. D3 is written in Javascript and is structured to make sense to a Javascript developer and depends on features built into the Javascript language. There are clear advantages to going this route. Most importantly, if someone is building an application in a language, her or she is likely comfortable with the paradigms and built-in functionality associated with that language. There is also the benefit that comes with any DSL that the library does not need to 'reinvent the wheel' with standard functionality. The downside is that by depending on the host language as much as they do, the code generated by these systems is inherently unportable. An interesting graph visualization implemented in D3 is only available in a Javascript-based system and the same holds true with prefuse and Java. (One toolkit that acts as an exception to this rule is The Visualization Toolkit (or VTK) [197]. VTK is an API for developing scientific visualization systems using the dataflow architecture. VTK was originally written for C++, but the API has also been ported to Java and Python through bindings that call the C++ code.)

Visualization grammars do not have these host-language constraints. One of the most successful visualization grammars is Wilkinson's Grammar of Graphics [247].

The GoG breaks down techniques into seven stages: variables, algebra, scales, statistics, geometry, coordinates, and aesthetics. The variable, algebra, scales, and statistics stages cover the data component of the technique while the geometry, coordinates, and aesthetics cover the 'visual' component. (The grammar does not include interaction specification.) Graph visualization in the GoG is handled by the 'network geometry' that generates a node-link diagram of the nodes and edges in the graph. The most popular instantiation of the grammar is Wikham's ggplot2 plugin [245, 246] for the R statistics package. A related technology is VISO [238], an ontology for visualization. The ontology includes a vocabulary for specifying heuristic rules, information about the user, and information his or her environment.

Another, more recent, example of language-independent visualization specification languages is the Vega visualization grammar [225]. Analogous to how DSLs are built on top of a programming language, Vega is built on top of a markup language, namely JSON. Unlike the toolkits described above, Vega specifications were never designed to be written 'by hand'. Rather, Vega was designed to act as a *file format* equivalent to docx or svg. In that sense, Vega is not written by a developer, but rather by another system (such as Word writing docx files or Illustrator generating svg files). The Vega specification that a system writes consists of the data (either the raw data tables or the location of those tables), transforms on the data (such as normalizing a property or finding unique values), scales, axes, legends, and finally marks (how the data is mapped to glyphs in the display). These Vega files can then be rendered in any graphics environment for which someone has written a Vega renderer. (Vega's website currently lists support for two W3C standard web-based graphics environments: HTML Canvas and SVG). In addition to data transformations, the Vega specification supports what it calls *view encoding transforms*. One of these view encoding transforms is a force-directed layout that takes an array of nodes and an array of edges and generates the x,y coordinates of the node-link diagram. (This is

also how D3 handles its force-directed layout.) While the first release of Vega had minimal support for interaction (the display properties of a glyph could be updated when the mouse hovered over it), the designers have implemented a more thorough interaction library using the reactive programming paradigm [195]. In addition to Vega, Vega-lite [253] is built atop Vega and uses smart-defaults to significantly shrink the size of specifications.

The goal of my work with GLOs is to provide the same functionality that Vega provides for tabular visualization techniques, but for graph visualization techniques. Rather than treating network visualization as composed of simply node-link diagrams and shoehorning it into a framework designed for tabular data, GLOs are explicitly for visualizing graph data structures. (Of course, by considering tabular data as graph data without any connections between the data items we can always shoehorn tabular data visualization into a graph-focused framework.) Furthermore, like the Grammar of Graphics, one of the goals of this work is to better understand the techniques themselves by understanding how they are put together.

I am also approaching the structure of the framework differently than the Vega designers. While the high-level nature of the GLOs means that the expressiveness of GLOs may never be equivalent to that of Vega, there are still advantages to the GLO approach. One is that GLO specifications are shorter and more 'human-readable' than equivalent Vega specifications would be. Another is that GLO models push many of the low-level details of rendering the visualization to the interpreter. This enables the interpreter instance to implement each operation in the best manner for the target language and graphics system.

Finally, a brief note on two other toolkits. Raphael [26] and Processing [1] are not visualization libraries explicitly, but have both been used for visualization. Raphael is a Javascript library for drawing vector graphics using SVG elements designed for beginning programmers. Processing is a programming language and environment for

building interactive art. Processing's primary implementation is in Java, though the creators also have a Javacript-based implementation (most, but not all, code in one implementation ports to the other). Processing has been immensely popular amongst artists and other creatives, with a very shallow learning curve compared to most other visualization libraries.

2.2 Systems for Applying Techniques to Data

At the application level, there are a number of systems that allow a user to apply a variety of pre-defined visualization techniques to their tabular data. The most ubiquitous of these tools is Microsoft's Excel spreadsheet software [164]. Excel has long supported a wide variety of techniques including barcharts, linecharts, scatterplots, and pie charts. Polaris [217], and its present commercial form Tableau [219], go beyond Excel in a number of ways. First, it supports a wider variety of techniques such as maps and treemaps. Second, it uses an underlying language (called VizQL [158]) to understand the data-field signatures of different techniques. For example, a scatterplot expects two quantitative data fields; a scatterplot matrix expects n quantitative fields. Using this language, Tableau is able to suggest optimal visualization techniques based on the analyst's choice of data fields to show (a feature they call *Show Me*). The Spotfire system [223] and the ManyEyes site [237] similarly supports a wider range of techniques than Excel and limit techniques to only those where the field data types make sense. SageBook [62] allows an analyst to apply (data-appropriate) techniques created using SageBrush (described below). Common to all these systems though is a notable lack of support for graph data structures. Each can treat a graph as a table of nodes or a table of edges and then visualize that, but none of them actually support any of the visualization techniques that show the structure of the network.

At the other end of the spectrum are those system explicitly for visualizing graphs.

Some early systems include UCINet [39], Pajek [29], and Tulip [19]. Two systems that have gained popularity in recent years are Gephi [28] and NodeXL [209]. There are also two commercial graph visualization and analysis systems: Centrifuge [56] and ToughGraph Navigator [224]. (All of these systems are stand-alone except for NodeXL, which is an extension for Microsoft Excel.) All of these systems are for both analyzing networks (e.g. calculating statistics such as betweenness centrality [87, 88] and PageRank [176]) and for visualizing the network as a node-link diagram. The systems allow the analyst to customize the display such as setting what attributes of the nodes and edges should map to the various visual properties such as the size of the nodes, the color (hue, value, saturation) of the nodes and edges, and the thickness of the edges. Each includes a number of different force-directed layout algorithms such as Fruchterman-Reingold [90] or Kamada-Kawai [138]. Gephi and NodeXL were also designed to allow a developer to easily implement alternative layout algorithms. For example, the user-built Gephi plugin library includes an implementation of the hive plot [144] technique as a layout option. NodeXL's developers have experimented with techniques beyond node-link diagrams, such as Dunne and Shneiderman's motif simplification [71]. Two other graph visualization and analysis systems of note are Ploceus [153, 154] and Orion [110, 114]. Both systems provide an interface for converting tabular database records into graphs and then manipulating the graph structure through a direct manipulation interface. Both then allow the analyst to view the current state of the network using node-link visualization techniques. Orion (unlike Ploceus, Gephi, and NodeXL) also includes the ability to show a network as an adjacency matrix or a scatterplot. Most recently, the DOSA system [232] provides a system for exploring large graphs by showing summaries of selected subgraphs while simultaneously showing the full subgraph in a different display. One of the advantages of a technology like GLOs is to make it easier for engineers to add these additional visualization techniques (and potentially more) to these analysis systems

without needing to learn a one-off API or development environment (such as Gephi's or NodeXL's). This work will also allow traditional tabular visualization systems (such as Excel and Tableau) to more easily integrate network visualization into their existing products.

2.3 Systems for Specifying Visualization Techniques

While the toolkits in Section 2.1 allow a programmer or developer to specify techniques in code, and the systems in Section 2.2 allow an analyst to apply a predefined technique to their data, the systems in this category allow a non-programmer to specify new or customized techniques. In a sense, these systems enable someone to build templates of visualizations to be applied in the future. Many, though not all, of these systems tend to fall into one of two categories: those that use the dataflow or pipeline model and those that use a drag-and-drop/direct manipulation metaphor.

The data-pipeline or dataflow model allows a user to visually build pipelines from data to visualization using modules with specific inputs and outputs. These modules might include normalizing a data feature or mapping features to visual attributes. Early examples of the category are the AVS [231] and apE [73] scientific visualization systems, followed by Data Explorer [155], SCIRun [178], IRIS Explorer [85], VTK [197], and VisTrails [31]. VisIt [61] and VisMashup [192] represent the third generation of the dataflow metaphor. VisMashup provides a more usable front-end for VTK by simplifying VTK's pipeline modules into more readable displays as well as VisMashup supporting multi-view systems through multiple pipelines with synchronized input parameters. The Tioga-2 [10] and Impure [174] systems applied the dataflow model to non-scientific visualization. Graph-level operations draw heavily from the dataflow system's use of encapsulation. Each task in the visualization-creation process is operationalized. However, the pipeline systems also operationalize the very low-level tasks such as database accesses. It is these low-level tasks that a

GLO model abstracts away from the designer. Furthermore, GLOs are designed to be as order-independent as possible, as opposed to pipeline systems that are rigorously sequential with fixed input and output specifications.

The system that first brought the drag-and-drop/direct manipulation metaphor for visualization specification was SageBrush, a component of the SageTools [190] suite. SageBrush employs the drag-and-drop metaphor to enable an analyst to add marks to a canvas and then set properties of the marks by dragging data features onto handles of the marks. The designer could then save the technique as a template that could be used by the other components of SageTools (SageBook and SAGE). The Delaunay system [64], built for creating custom visualizations of database data, continued the trend. Delaunay uses a similar declarative drag-and-drop metaphor for visualization creation as SageBook, combined with a constraint solver for realizing the specification. Delaunay provides support for recursive definitions of visualizations, thus making techniques such as node-link tree diagrams easier to specify. DataSplash [256] was designed explicitly for building semantic zoom applications, allowing the user to define different views at different levels of zoom elevation. Flexible Linked Axes (FLAs) [63] takes a slightly different tack, allowing a user to add, remove, and manipulate only axes using drag-and-drop. The user drags axes onto a canvas, selects the data features to which the axes are mapped, and then links the axes together using either lines (equivalent to parallel coordinate plots) or dots (equivalent to scatterplots). Visualization Primitives [208] uses the drag-and-drop metaphor, but includes support for a greater number of features of each graphical primitive than previous systems as well as more easily supporting compound visualizations such as stacked bar charts. Lyra [196] works similarly to Visualization Primitives, though is a more complete system. Lyra is most interesting in that it saves its templates as Vega specifications. The Vega-Lyra relationship mirrors the relationship between GLOs and the applications built on it. Lyra (and applications) allow a designer to specify

techniques that can then be used in any system that supports the standardized Vega (or GLO) format.

Finally, and graph-specifically, the GUESS system [7] (discussed earlier in respect to its Gython language) provided the user with the ability to change the visual properties of the nodes and edges in a node-link visualization in one pane through interacting with a command-line interpreter in another. Furthermore, the interpreter and display pane were linked, in that mousing over a node's or group of nodes' identifier in the interpreter highlighted the node or group of nodes in the display. This provided the analyst with more control than applying a preset layout algorithm (though the analyst could always call those from the interpreter as well). GUESS also had some standard tabular visualization techniques (such as bar charts and pie charts) available to render, using separate panes for each chart type. The GLO-CLI described in Section 5.1 is directly based on this aspect of GUESS.

2.4 Visualization Design Space Analysis

There is a body of work in visualization on understanding the design space of visualizations. Li et al. [152] provide a good summary of the research in their research on analyzing the design space of tree visualizations. They define a feature space for tree visualizations and cluster a variety of techniques from literature using the feature space. Schulz et al. [198] represents the most relevant work to GLOs. The authors define the design space of implicit hierarchy visualizations (e.g. treemaps and icicle plots) according to four dimensions: dimensionality (2D or 3D), Node Representation, Edge Representation, and Layout. They produced a tool that allows a designer to load a hierarchical dataset and select values for each dimension. The tool then generates the technique with those values.

CHAPTER III

A METHOD FOR GLO MODEL INDUCTION

In order to take advantage of a model of graph visualization, one must first determine the elements of that model. In this chapter, I present and describe a repeatable method for inducing a model of graph visualization from a set of exemplar graph visualization seed techniques. The method is designed to identify recurring features within the set of techniques at a practical level of complexity. In other words, the resulting model should not be so fine-grained as to be tedious, but also not so coarse-grained as to not be expressive. The method relies on one or more human identifiers to determine steps to change the glyphs of one technique into the glyphs of another technique (i.e. to **transition** from one technique to another). This is conceivably problematic since there are infinitely many ways to transition between any two techniques by transitioning through intermediate states that have nothing to do with the two techniques (such as hiding all of the edges and then showing them again when both techniques have all of their edges permanently displayed). However, when identifiers aim towards identifying a small set of atomic units of graph visualization and avoiding unrelated states, the method works well in practice. Furthermore, the method helps define how to describe techniques using the resulting model and guarantees that you can describe each member of the set of seed techniques using the resulting model. (I discuss these latter points in Chapter 4.)

At a high level, the method is broken down into four steps:

1. Identify a set of exemplar seed techniques.
2. Determine the set of reusable, high-level operations needed to transition from each technique in the set to each technique in the set (including self-transitions).

3. Use those transitions to identify a visual element model and set of graph-level operations that manipulate the visual element model.
4. Augment the set of operations.

In the following sections, I walk through each of these steps, including examples from the GLOv1 and GLOv2 induction processes.

3.1 Seed Technique Identification

The first step of the induction process is to identify a set of techniques commonly used for graph visualization—the **seed techniques**. These techniques should provide a good cross-section of network visualization features. Equivalent to a training set in a machine learning scenario, features that do not appear in the seed techniques will not be represented in the resulting model. In order to maximize the variety of techniques, one strategy is to identify the seed techniques through a comprehensive search of graph visualization literature and tools. An alternative strategy might be to seed the method with a set of techniques that must be modeled (for example, in a software package). The former strategy is more useful from both a theoretical perspective (such as understanding the design space of visualization techniques) and for future-proofing software packages towards adding additional techniques.

This step also includes identifying the inherent features of each technique that define the technique. In other words, if an instance of a technique uses a specific color scheme or an unusual shape, an identifier must determine whether the scheme or shape is inherent to the technique or specific only to the given instance of the technique. Using abstract forms of techniques enables the resulting visual elements model and operations set to be simpler, while still expressing the critical features of the techniques.

For example, the critical components of the Edgemap B technique shown in Figure 4 are that the nodes are evenly distributed and aligned along the middle of the

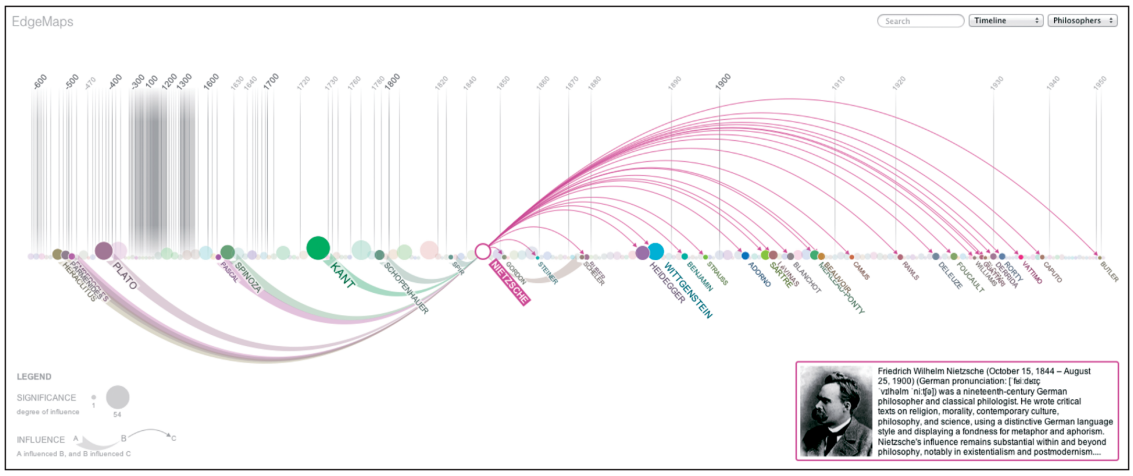


Figure 4: GLOv2 EdgeMap B seed technique from [74].

display, nodes are colored by a property and sized relatively by a property, edges are only shown when an analyst mouses over an endpoint node and are displayed differently based on whether the edge is an in- or out-edge of the node, edges are colored the same as their source node, and when the analyst mouses over a node the neighbors of that node are highlighted. I chose to abstract away the specific styling of the edge glyphs (that in-edges and out-edges are rendered differently is more critical aspect of the technique), the color schemes for the nodes and edges (that the nodes are colored by a discrete attribute is more critical than the choice of blue, purple, and pink as the colors), and the labels (since they are closely tied to time rather than a generic continuous attribute). I also chose to abstract away the detail-on-demand panel. With GLOv2, I wanted to focus on the different ways that techniques visually represent nodes and edges. Therefore, I abstracted away features that involved additional glyphs (such as details-on-demand panels).

3.1.1 GLOv1 Identification

During the GLOv1 induction process, I (and my coauthors of [216]) seeded the method with six techniques: Force-Directed Layouts, Circle Plots, Scatterplots, Semantic

Substrates [204], PivotGraphs [244], and Adjacency Matrices. (Descriptions and depictions of these can be found in Appendix A.) These techniques were hand-picked to provided good coverage of 2D graph visualizations while intentionally excluding techniques (most notably Arc Diagrams) in order to demonstrate the expressiveness of GLOs. (As I show in Section 5.3, Arc Diagrams can still be represented using the GLOv1 model.)

3.1.2 GLOv2 Identification

The GLOv1 seed technique set has two notable limitations. First, there are only six techniques, which limits the expressiveness of the resulting model. Second, the set of seed techniques was hand-picked, rather than chosen through a repeatable method.

In order to identify a broader set of seed techniques (with the purposes of identifying a more expressive GLO model), I conducted a literature review of graph visualization techniques. I seeded my literature review with papers with graph-related terms from the IEEE Infovis, IEEE Vis/SciVis, and IEEE VAST conferences using the Visualization Publication Dataset [131]. I then added publications on graph visualization techniques referenced by these works, and then publications on graph visualization techniques referenced by those works, and so on. In all, I reviewed 430 graph visualization publications.

Through the course of the review, I identified seven categories of 'techniques': graph visualization techniques, tree visualization techniques, directed acyclic graph visualization techniques, dynamic graph visualization techniques, graph visualization interactions, graph visualization display options, and graph visualization systems. Graph visualization interactions and graph visualization display options are not complete techniques, but rather specific interactions or display options that could be applied to any number of other techniques. An example of a graph visualization display option is edge bundling [257, 120]. Edge bundling is not a technique to display

the data of a graph, but rather an additional step that can be applied to any graph or tree visualization technique that displays its edges as lines. Graph visualization systems are applications that implement one or more graph visualization techniques. In the case where those techniques are novel, I included the system in the list of systems and the technique in the list of techniques. For example, the Ploceus system [153] utilizes the novel Attribute Matrix technique.

Having completed the literature review, I set out to determine the seed techniques for what would become GLOv2. I immediately ruled out the graph visualization systems, as they are not techniques, but rather collections of techniques (which are included in the other categories as appropriate). I next ruled out the dynamic graph visualization systems as dynamic graphs represent an additional dimension of complexity over the standard graph data model.

When it came to directed acyclic graph (DAG) visualization techniques and tree visualization techniques, I chose to exclude these techniques as well. Put simply, all graphs (including trees and DAGs) can be visualized using graph visualization techniques whereas general graphs can only be visualized using DAG and tree visualization techniques by designating one of the nodes as the root and (in the case of trees) removing edges from the graph. One of the core properties of GLOv1 was that operations apply to every node equivalently. In tree and DAG visualizations, many aspects of the techniques depend on considering nodes differently based on their distance from the root.

I eventually chose to ignore the graph visualization interactions and visualization display options as well. The sub-techniques in these two categories fit well under the premise of graph-level operations model—they are discrete modifications that apply equally to all elements of the visualization. However, I chose to focus the model towards full graph techniques, rather than these **technique-independent operations**. Some of the display options and interactions in fact appear in the chosen seed

techniques and are represented in the GLOv2 model in Chapter 4. (For example, displaying convex hulls around clusters occurs in the GMap seed technique and selective highlighting of neighbors occurs in the EdgeMap A and B seed techniques.) On the other hand, many of the display modifications and interactions do not occur in the final set of seed techniques and are therefore not represented in the GLOv2 model. I will discuss technique-independent operations further in Chapter 6.

Finally, I pared down the approximately 55 general-purpose graph visualization techniques to 29 seed techniques. First, I removed the techniques that begin by reducing the graph to a tree (usually using a minimum-spanning tree algorithm or hierarchical clustering algorithm) and then visualizing the graph using a tree visualization technique as these are practically tree visualization techniques. (These included SPF [14], MO-Tree [42], Space-Filling Curves [167], Treemap of hierarchy [168], Treemaps with Links [82], Similarity trees [177], ArcTree [171], TreePlus [149], Grouse [17], GrouseFlocks [15], and TreeNetViz [98].) I then removed any three-dimensional techniques in order to focus on two-dimensional techniques. (3D techniques included Ask-Graphview [5], 3D node-link using stereoscope [242], Graph Surfaces [6], Cityscape [57], State-Transition Graphs [233], Landscape [45], and WilmaScope [9]).

Lastly, there were a class of techniques that were highly dependent on knowing the topology of the network to calculate node and edge positions. I chose to exclude all but one technique this class. These techniques included C-Group [139, 27], B-Matrix [22], compressed adjacency matrix [68], and Edge-Compression [72]. In addition, two techniques (SegmentView [27] and JauntyNet [137]) are similar to each other in that nodes are positioned based on their attributes, which are represented in a circle along the radius of the visualization display. Rather than only including glyphs representing nodes and edges, these two techniques introduce glyphs representing attributes. Positioning glyphs relative to the position of these attribute glyphs is similar to the case where the topology is driving the layout.

Force-Directed Layout [138]	Edge-Label-Centric [182]
Matrix Plot [34]	Honeycomb [106]
Cluster Circles [69]	GraphDice Segment [36]
Circle Graph [207, 93]	3x3 GraphDice [36]
GeneVis A [23]	GMap [94]
GeneVis B [23]	Attribute Matrix [153]
Arc Diagram [243, 141]	EdgeMap A [74]
Matrix Browser [262]	EdgeMap B [74]
Matrix with Bars [205]	Hive Plot [144]
MatrixExplorer [116]	2x3 Hive Panel [144]
NetLens [140]	ScatterNet [27]
Semantic Substrates [204, 18]	Citevis [213]
PivotGraph [244]	DOSA [232]
MatLink [118]	NodeTrix [117]
List View [199, 214]	

Table 1: GLOv2 seed techniques

The one topology-dependent technique (and its variants) that I left in the set was the Force-Directed Layout. The decision to do so was that, as the most-used network visualization technique, any model of graph visualization should be able to describe Force-Directed Layouts. This decision would eventually result in a stand-alone *apply force-directed algorithm to nodes* GLO.

The final result of this selection process was a final set of 29 graph visualization seed techniques that can be found in Table 1 Appendix B. Lists of the non-system techniques not chosen as seed techniques can be found broken down by category in Appendix E.

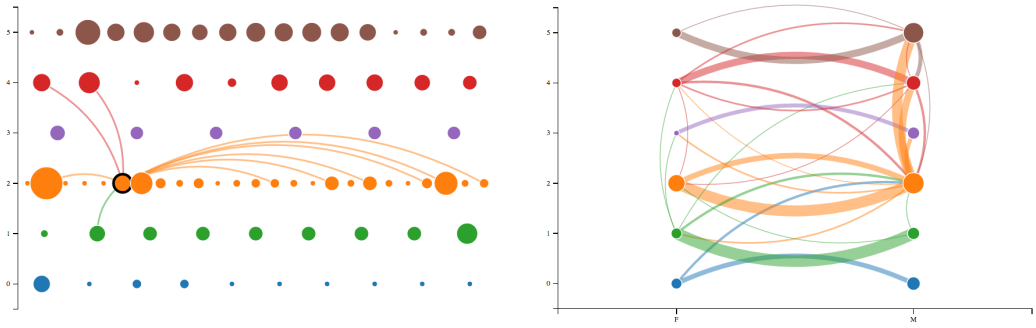
3.2 Transitions, GLOs, and the Transition Matrix

Having identified a set of seed techniques, the second step is to identify the high-level operations necessary to transition from an instance of each technique in the set to another instance of each technique in the set. (This includes transitioning from an instance of a technique to an alternate instance of the technique.) In other words, what is *different* between the two techniques/instances? Is the only difference the position of the elements? Or should the glyphs be drawn differently? Does the underlying data have to be manipulated (such as through aggregation)?

The goal of this second step is to generate a **transition matrix** of the seed techniques. Each (i, j) value of this matrix is a list of operations necessary to transition from the ith seed technique to the jth seed technique. Let me demonstrate this process with two such transitions (i.e. transition matrix entries).

3.2.1 Semantic Substrates to PivotGraph

For the first example transition, consider the abstract Semantic Substrates [204] representation of a graph in Figure 5a and the abstract PivotGraph [244] representation of the same data in Figure 5b.[1]



(a) Semantic Substrates [204] (b) PivotGraph [244]

Figure 5: First example transition techniques

The graph data[2] represented concerns book characters and includes discrete attributes gender and cluster and quantitative properties such as the degree of each character in the network. In the abstract Semantic Substrate representation, glyphs representing nodes are positioned along the y axis by a discrete attribute (here a cluster label) and are colored by the same attribute. The node glyphs are distributed along the x axis within the clusters (more precisely, within the groups defined by the discrete attribute). Axis labels are shown for the y axis using the discrete attribute values. The node glyphs are sized by a quantitative attribute (here the degree of the

[1]Note that these are the GLOv1, not GLOv2, seed technique versions of these two techniques.

[2]Les Misérables character co-occurrence graph included with D3.js based on Donald Knuth's jean.dat file available at http://www-cs-staff.stanford.edu/ uno/sgb.html

node). Edges are displayed as constant-sized, curved line glyphs. Each edge glyph is colored by the same discrete attribute that the nodes are colored by in order to have the same color as their target nodes' glyphs (i.e., here the lines are colored by the cluster of their target node). However, edge glyphs are only shown when the analyst interacts with an endpoint node of the edge. (For example, the analyst is interacting with the node represented by the glyph circled in black in the figure).

In the abstract PivotGraph representation, node glyphs are similarly positioned on the y axis by a discrete attribute and axis labels for the y axis are shown. The node glyphs are colored by a property (discrete or continuous), here by the discrete cluster property. Each edge's curved line glyph is colored by a property (discrete or continuous), here (as with the Semantic Substrates abstract technique) by the cluster of the edge's target node. Unlike the abstract Semantic Substrates representation, the abstract PivotGraph does not display a glyph for every node and edge in the graph. Rather, the technique displays glyphs representing multiple nodes and edges. These **super-nodes** and **super-edges** represent aggregations of the nodes and edges based on properties of the data. In the abstract PivotGraph, nodes are aggregated by two discrete properties (here, cluster and gender). In other words, each super-node glyph represents a super-node that is an aggregate of all of the nodes that share both the same cluster and gender properties. Edges are then aggregated based on the values of those same two discrete properties of both their source and target endpoints. Therefore in this display, each super-edge glyph represents a super-edge aggregating all of the edges in the graph whose source nodes have the same cluster and gender and whose target nodes have the same cluster and gender. The super-node glyphs are then positioned on the x axis based on the second discrete attribute (here the gender of the super-nodes) and axis labels are shown for the x axis. The super-node and super-edge glyphs are then sized by an attribute. Here, the attribute is how many original nodes or edges the glyph represents.

Having defined the two abstract techniques, let me now demonstrate transitioning from the Semantic Substrates representation to the PivotGraph representation. Each step of the transition involves recognizing a difference between the two techniques and altering the representation to resolve the difference.

During this process, one can choose between two approaches. The first is to take a **best-case approach**, where anything that can stay the same between the two techniques does. For example, since both representations have the nodes aligned on the y axis by an attribute, that does not need to change. An second approach is a **worst-case approach**. Under this approach, one assumes that though the two displays position nodes on the y axis, they do so by different attributes and thus this must be included in the transition. During the GLOv1 identification process, my colleagues and I used the worst-case approach. In practice, however, this approach generates unnecessarily long lists of differences when transitioning between different techniques. Therefore, for GLOv2, I followed the **best-case approach** when transitioning between instances of two different techniques. (However, when determining the steps to transition between two arbitrary instances of the same technique I used the worst-case approach. Some operations are only necessary for specific techniques. For example, transitioning between two arbitrary instances of the 3x3 GraphDice GLOv2 seed technique (Figure 6) requires a transition step that does not occur in any other transition.)

In Figure 7, I begin with the abstract representation of the Semantic Substrates technique described above.

The first difference between the two abstract techniques is that the PivotGraph representation displays every edge glyph, not simply those of edges incident to a single node. Therefore one can show all of the edges, resulting in the display in Figure 8.

The second difference is that the node glyphs are not distributed across the display in the PivotGraph, but rather are positioned on x according to a discrete attribute.

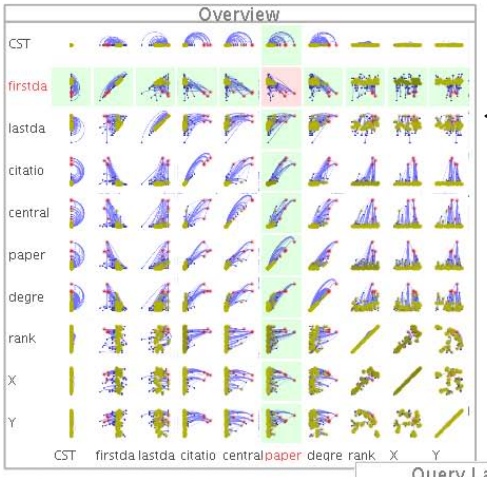


Figure 6: GLOv2 GraphDice seed technique from [36].

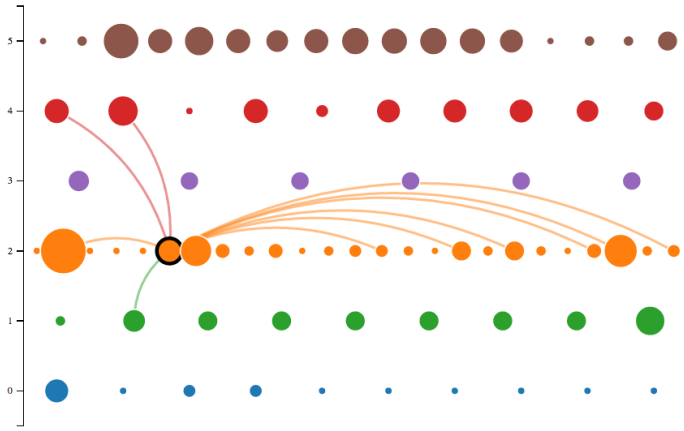


Figure 7: Unmodified Semantic Substrates representation.

Taking the display in Figure 8 and positioning the node glyphs along the x axis based on a discrete attribute (namely the gender of the nodes) results in the display in Figure 9.

The third difference is that the PivotGraph has axis labels displayed for the x axis. Figure 10 is Figure 9 with those labels displayed.

The fourth difference is the PivotGraph's use of glyphs representing super-nodes rather than a glyph for each node. In Figure 10, the nodes that share the same cluster and gender are stacked at the same grid position. Since the nodes in the Semantic Substrates display were sized by a continuous property (namely, their degree) and

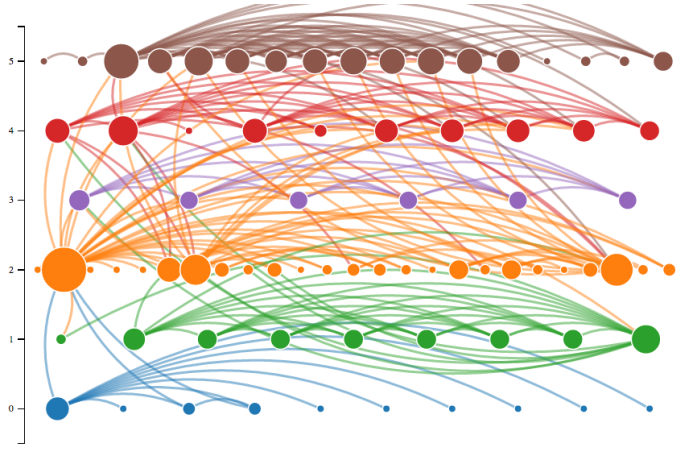


Figure 8: Semantic Substrates representation modified to show all edges.

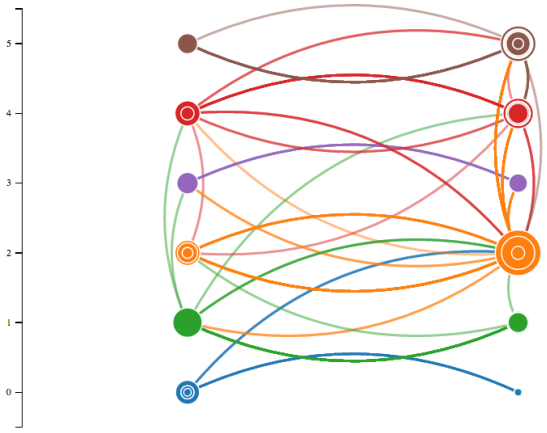


Figure 9: Semantic Substrates representation modified to show all edges and position nodes on x by the discrete gender attribute.

this has not been modified, the stack of node glyphs form bullseye-like displays. Aggregating the nodes represented by each stack (i.e. nodes that have the same values for both axes' discrete variables, in this case cluster and gender) into super-nodes (and the associated glyphs) results in the display in Figure 11. The size of each super-node glyph is an average of the sizes of the constituent nodes.

The fifth difference is that the super-node glyphs are not sized by their degree in the abstract PivotGraph, but rather by the number of original nodes that the super-node represents. Sizing the super-node glyphs in Figure 11 by the number of nodes

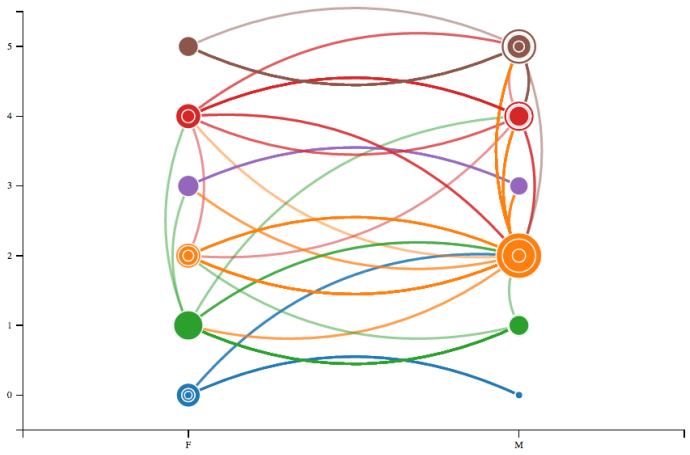


Figure 10: Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, and show x axis labels.

they represent results in the display in Figure 12. (This change is rather subtle with this dataset, but compare the size of the cluster-3/female super-node glyph in the two figures.)

The sixth difference is that the PivotGraph also displays super-edge glyphs. The difference between Figure 12 with individual edge glyphs and Figure 13 with super-edge glyphs is subtle. The reason for this is that the edge glyphs used by the Semantic Substrates abstract technique are constant-sized. This means that unlike the "bulls-eye stacks" of unaggregated node glyphs, the "stacked" edge glyphs are hidden behind the top-most drawn edge glyph in each stack. In Figure 13, each stack is aggregated into a single glyph by aggregating edges that share the same discrete properties. These discrete properties are not of the edges themselves, but rather of the endpoint nodes of each edge. Thus, for this display, those edges with source nodes having the same gender and cluster values and target nodes each having the same gender and cluster values are aggregated together into super-edges and represented by super-edge glyphs.

The seventh and final difference between the Semantic Substrates abstract technique and the PivotGraph abstract technique is that rather than the constant-sized

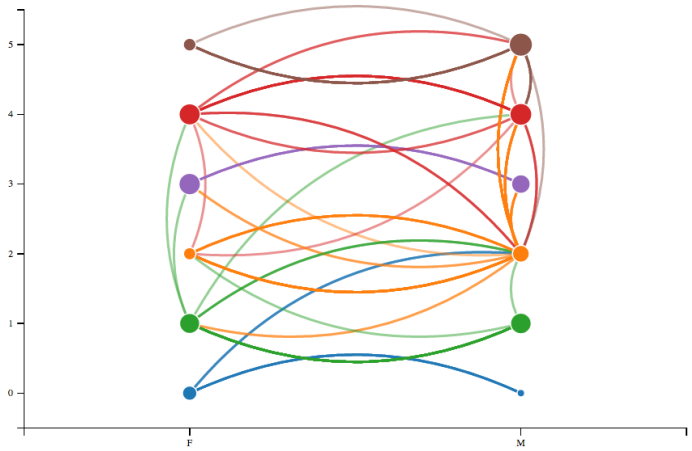


Figure 11: Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, and aggregate nodes into super-nodes.

edge glyphs of the Semantic Substrates display, the PivotGraph display's super-edge glyphs are sized by the number of original edges aggregated to form the super-edge. Figure 14 is Figure 13 with the super-edges sized in this way. In fact, Figure 14 is the target abstract PivotGraph representation from Figure 5b.

3.2.2 Force-Directed Layout to Matrix Plot

As a second example transition, consider the Force-Directed Layout and Matrix Plot abstract techniques in Figure 15.[3]

The graph data displayed in the abstract techniques is the same as the prior transition, and once again the nodes are colored by the discrete cluster attribute. In the Force-Directed Layout, the nodes are represented as constantly-sized circles positioned using a force-directed algorithm and edges are represented as constant-colored, constant-sized straight-lines between their endpoint nodes' glyphs. In the Matrix Plot,

[3]Note that these are not the GLOv2 seed technique versions of these two techniques. Unlike in both seed techniques, the node glyphs in both abstract techniques are colored by an attribute (namely, the cluster of the nodes) in order to make the steps of the transition clearer. Furthermore, in order to simplify the transition and highlight the need for cloning representations, the nodes in the Matrix Plot are represented by circle glyphs instead of the GLOv2 seed technique version's textual labels, and the edges in the Matrix Plot are displayed as constantly-colored squares, rather than relatively-colored squares as they are in the GLOv2 seed technique.

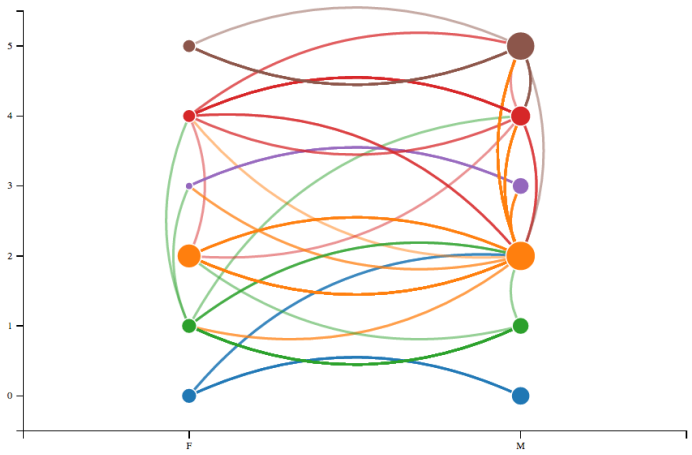


Figure 12: Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, and size the super-nodes by the number of nodes they represent.

nodes are represented as constant-sized circles colored by an attribute (in this case, the cluster of the node). The nodes are represented in both an evenly distributed row along the top of the display as well as in an evenly distributed column along the left of the display such that the left-most node of the top row matches the top-most node of the left column. Each edge is represented as a constant-size, constantly-colored square positioned at the y coordinate of the edge's source node's representation on the left and the x coordinate of the edge's target node's representation along the top.

Once again, let me walk through the steps to resolve the differences between these two representations. I start with the Force Directed Diagram representation in Figure 16.

The first difference between the two techniques is the position of the nodes. Rather than positioned by a force-directed algorithm, the Matrix Plot has the nodes aligned along the top and left of the display. In order to form the row at the top using the node glyphs of the Force-Directed Layout, I first evenly distribute the nodes along the horizontal x axis. (I also signal that the nodes should be sorted by an attribute of the nodes (namely, the cluster attribute) so that the nodes of each cluster are adjacent

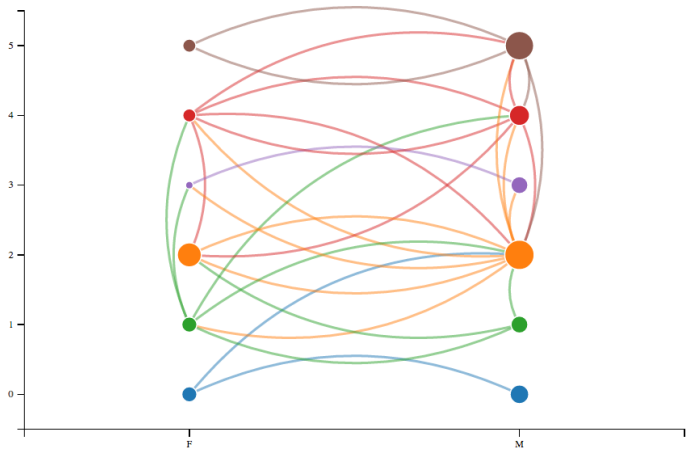


Figure 13: Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, size the super-nodes by the number of nodes they represent, and aggregate the edges by the edges' source gender, source cluster, target gender, and target cluster.

to each other.) I then align the node glyphs to the top of the display. The results of these two steps can be seen in Figures 17a and 17b. These two steps were separated (rather than simply aligning nodes on top in a row as one step) in order to increase the reusability of each step. Generalized steps can be used in multiple transitions. As I discuss below, many different transitions involve evenly distributing nodes along the x axis though they do not necessarily include aligning the nodes along the top of the display.

The second difference between the Force-Directed Layout and Matrix Plot is that the Matrix Plot includes a second set of node glyphs. Rather than a single glyph representing each node in the underlying data, the Matrix Plot has two glyphs for each node. Therefore, another glyph for each node must be introduced into the display. One way to do this would be to introduce a completely new set of glyphs. However, what form would this new set of glyphs take? Where would these new glyphs be positioned? A simple answer to these questions is that the new glyphs could be positioned where the current glyphs are positioned and be of the form and

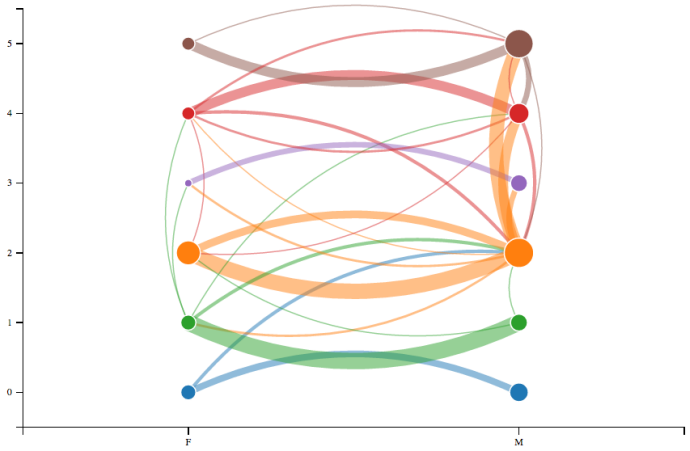
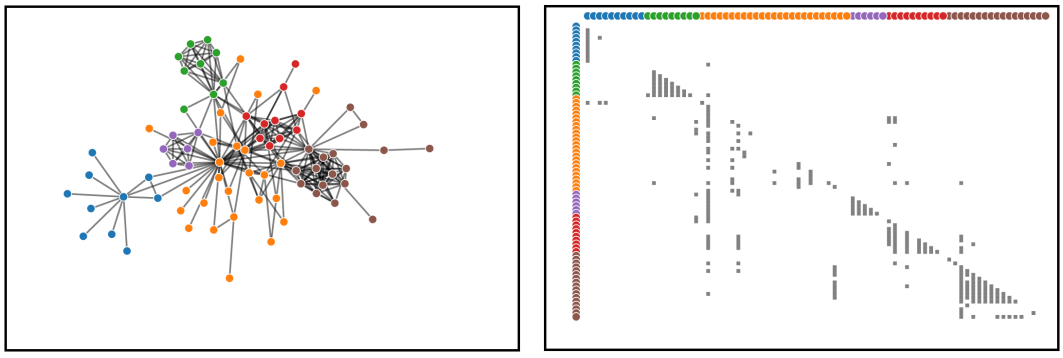


Figure 14: Semantic Substrates representation modified to show all edges, position nodes on x by the discrete gender attribute, show x axis labels, aggregate nodes into super-nodes, size the super-nodes by the number of original nodes they represent, aggregate the edges by the edges' source gender, source cluster, target gender, and target cluster, and size the super-edges by the number of original edges they represent. In other words, a PivotGraph representation of the graph.

style of the existing set of glyphs. This would leave no ambiguity about the extra set of glyphs. In other words, in order to introduce another set of node glyphs into the display I can **clone** the current set of node glyphs to create a duplicate set of glyphs. The result of this step is shown in Figure 18. As a 2d image, this representation is identical to the representation in Figure 17b since the second set of glyphs is directly on top of (in the z axis) the first set of glyphs.



(a) Force-Directed Layout (b) Matrix Plot

Figure 15: Second example transition techniques

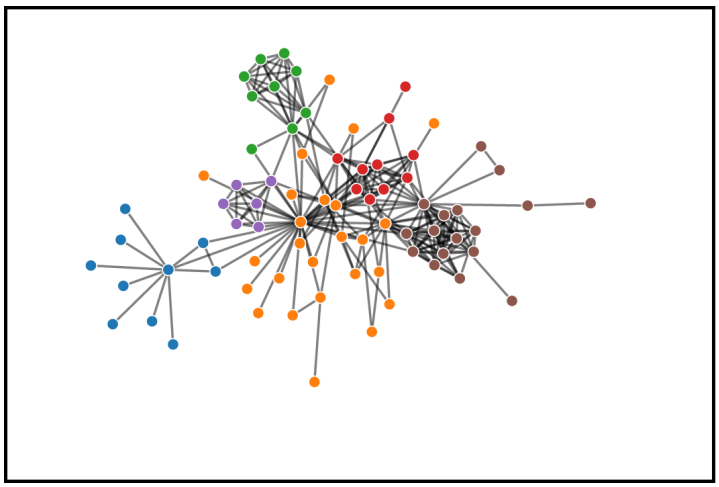


Figure 16: Unmodified Force Directed Diagram representation.

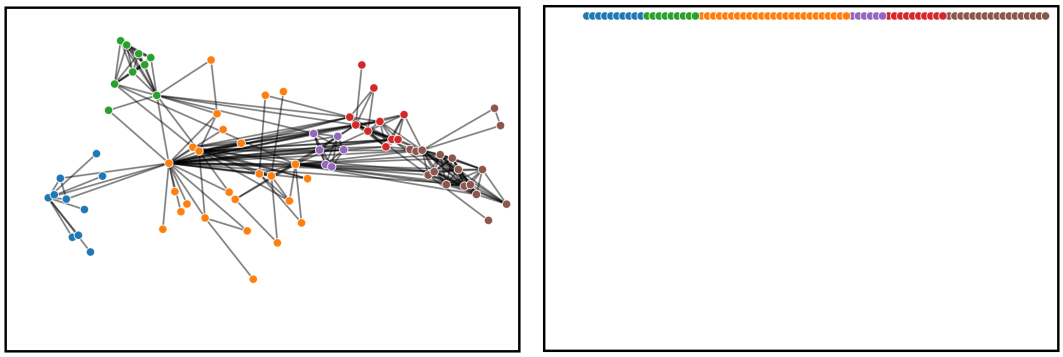


Figure 17: Abstract Force Directed Diagram representation after evenly distributing nodes on x (a) and after evenly distributing nodes along the x axis sorted by cluster and aligning the nodes at the top of the display.

With the second set of glyphs created, they must then be repositioned to form the column on the left of the Matrix Plot display. A logical next step is then to evenly distribute the new node glyphs along the vertical y axis. Given the Matrix Plot in Figure 15b, the node glyphs should be positioned such that the "blue" cluster is at the top of the column and the "brown" cluster at the bottom. If we were to assign values 1-77 to the nodes in the top row based on their position from left to right, then the column on the left should have nodes ordered from 1-77 downward. In other words, the origin point (0,0) of the display is considered the top-right corner. While this is the case for many matrix-based techniques (e.g. the Matrix Plot [34], MatLink [118], and

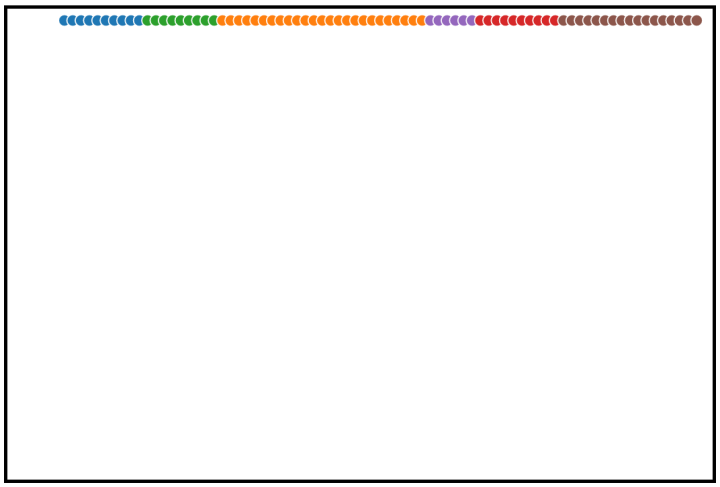


Figure 18: Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, and cloning the set of node glyphs.

Honeycomb [106] GLOv2 seed techniques), this is not the case for other techniques (e.g. the Scatternet [27], and GraphDice Segment [36], and DOSA [232] GLOv2 seed techniques) which assume an origin at the bottom-left corner. I chose to use the latter (bottom-left) origin point and therefore simply evenly distributing the second set of nodes sorted by cluster along the y axis would result in the display in Figure 19a, which is not the intended result. Instead, I evenly distribute the nodes along a "flipped" or **inverted** y axis (i.e. with the origin at the top rather than the bottom of the display) resulting in the display in Figure 19b.

I finish positioning the second set of nodes by aligning them to the left of the display, resulting in the display in Figure 20.

The third major difference between the Force-Directed Layout and the Matrix Plot is that the edges are not represented as lines, but rather as squares positioned based on the locations of the source and target endpoints of the edge in the left-side column and top row, respectively. In Figures 16-20, the edges have been drawn as straight lines between the nodes in the top row. Thus, one change that has to be made is to draw the edges from the left-hand set of nodes to the top set of nodes.

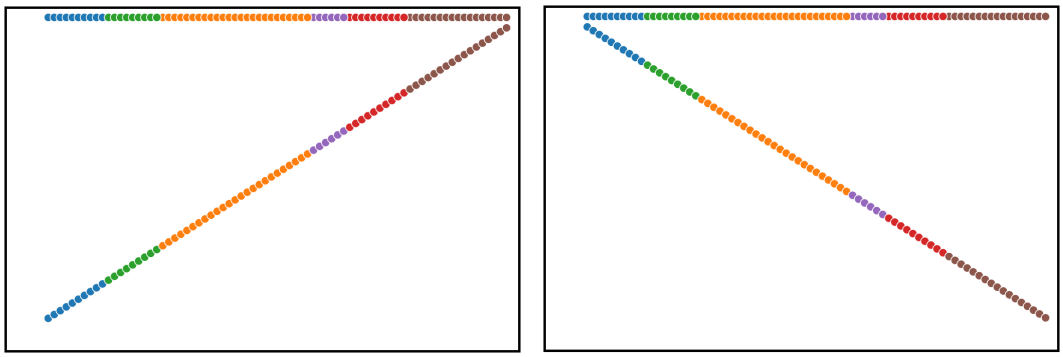


Figure 19: Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, and evenly distributing the new nodes on y sorted by cluster without (a) and with (b) an inverted axis.

Doing so results in the display in Figure 21.

The final change is therefore to display the edges as squares, rather than as straight lines. Doing so results in the target abstract Matrix Plot technique displayed in Figure 22.

3.2.3 Graph-Level Operations and the Transition Matrix

Each of the individual changes made to the displays during these transitions can be summarized as a **graph-level operation** (or **GLO**). Each operation represents a change to either the glyphs displayed in the visual representation (such as positioning the nodes by a property, aligning the nodes, or cloning the nodes), a change to the underlying data of the representation (such as aggregating nodes into super-nodes or aggregating edges into super-edges), or a change to an interaction in the display (such as showing every edge rather than only some edges based on analyst interaction).

One can thus represent the seven-step (Semantic Substrates, PivotGraph) transition as the following list of GLOs[4]:

- *show all edges*
- *position nodes on* $\{x\}$ *by* {attribute}

[4]I will discuss the {*parameters*} of the GLOs in the next section.

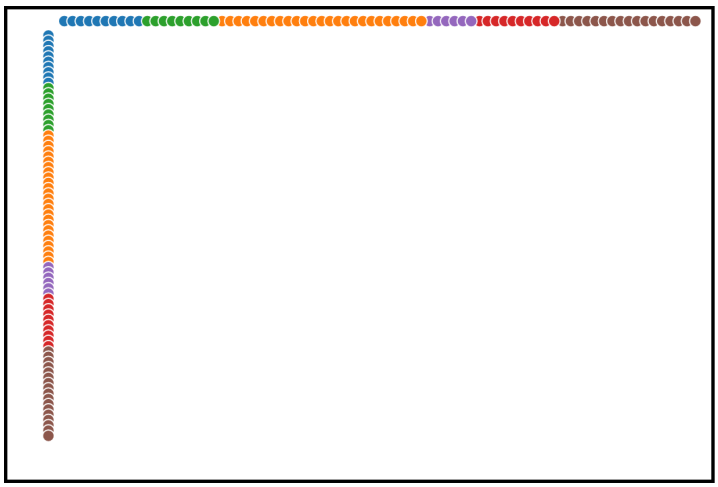


Figure 20: Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y sorted by cluster with an inverted axis, and aligning the new nodes to the left of the display.

- *show* {x} *axis*
- *aggregate nodes by* {discrete attribute(s)}
- *size nodes by* {attribute}
- *aggregate edges by* {discrete attribute(s)}
- *size edges by* {attribute}

The transition matrix entry for (Semantic Substrates, PivotGraph) would then be this list of operations.

Equivalently, one can represent the seven-step (Force-Directed Layout, Matrix Plot) transition as the following list of GLOs:

- *evenly distribute nodes on* {x} *(sort-by:*{attr}*)*
- *align nodes* {top}
- *clone nodes*
- *evenly distribute nodes on* {y} *(sort-by:*{attr}*, invert:true)*
- *align nodes left*
- *set source generation* {1} #i.e., draw edges from the second set of node glyphs (0-indexed)

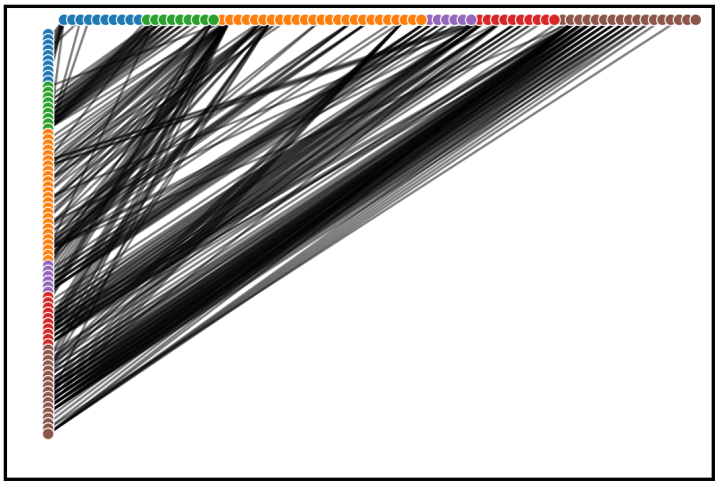


Figure 21: Abstract Force Directed Diagram representation after evenly distributing nodes on x sorted by cluster, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y sorted by cluster with an inverted axis, aligning the new nodes to the left of the display, and drawing edges from the second set of nodes (on the left) to the first set of nodes (on top).

- *display edges as squares*

and therefore the transition matrix entry for (Force-Directed Layout, Matrix Plot) would be this list of operations.

By identifying the operations to transition to and from each pair of techniques in the seed technique set (including self-transitions), one generates a complete transition matrix.

3.2.4 Handling GLO Uncertainty

Over the course of identifying the transitions for each ordered pair of techniques (i.e., each entry in the transition matrix), ideally operations are reused. For example, consider the four displays in Figure 23. Many (∗, CiteVis) and (∗, Semantic Substrates) transition matrix entries include the *position nodes on* $\{y\}$ *by* {attribute} GLO. Many (∗, MatLink) and (∗, Arc Diagram) entries include the *evenly distribute nodes on* $\{x\}$ operation. (I say "many" because not all (∗, Semantic Substrates) entries will contain the same operations. For example, the (CiteVis, Semantic Substrates) entry would

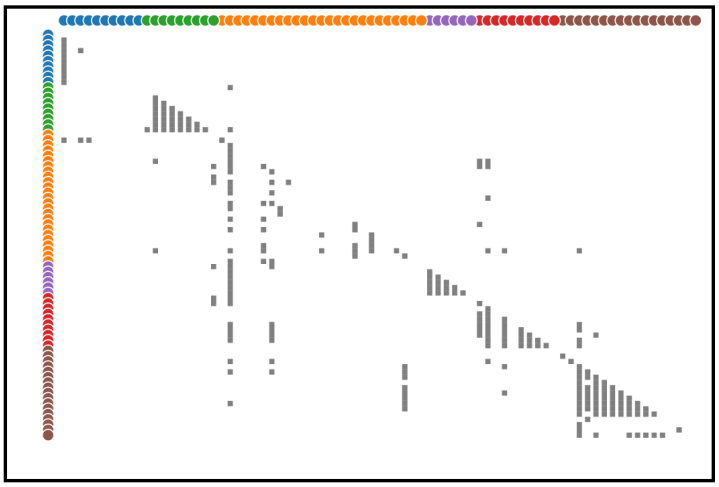
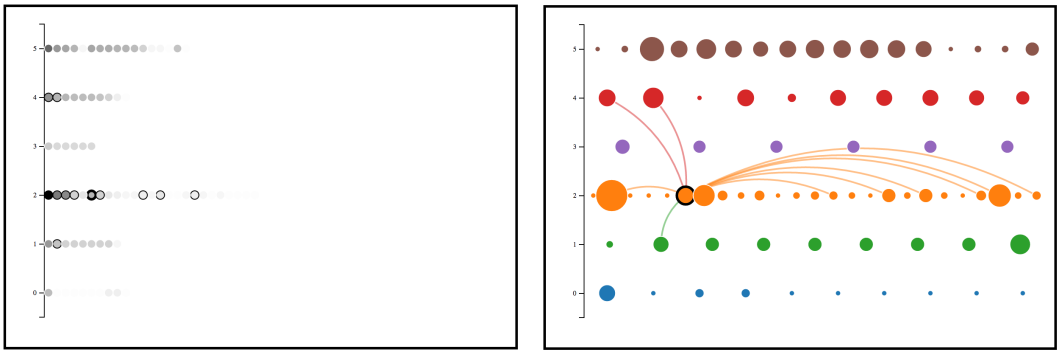


Figure 22: Abstract Force Directed Diagram representation after evenly distributing nodes on x, aligning the nodes at the top of the display, cloning the set of node glyphs, evenly distributing the new nodes on y with an inverted axis, aligning the new nodes to the left of the display, drawing edges from the second set of nodes (on the left) to the first set of nodes (on top), and displaying edges as squares. In other words, the target abstract Matrix Plot representation.

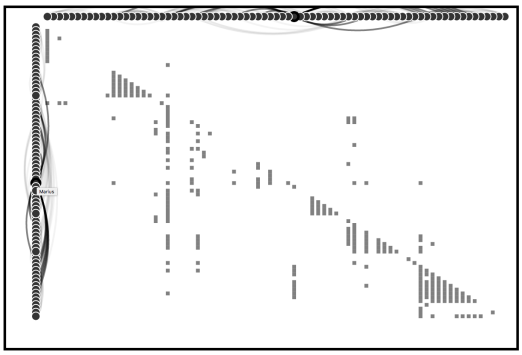
not include the *position nodes on* {y} *by* {attribute} GLO since the nodes in that display are already positioned along the y axis by an attribute.)

Positioning nodes by an attribute of the data is clearly distinct from evenly distributing nodes along an axis. Notably, the former requires a parameter specifying the attribute while the latter does not. On the other hand, sometimes deciding whether two changes should be considered the same operation can be unclear. Should positioning nodes by a discrete attribute and positioning nodes by a continuous attribute be considered the same operation or different? During the GLOv1 identification process, I and my colleagues erred on the side of distinct operations. Therefore, entries in our GLOv1 transition matrix separated operations for positioning nodes by discrete versus continuous attributes or for positioning nodes along Cartesian axes versus polar axes. During the GLOv2 identification process, I initially bootstrapped the set of operations with the GLOv1 operations. However, I eventually chose to err on the side of minimizing the number of distinct operations. This led to some GLOv1 operations being combined into a single GLOv2 operation. For example, entries in my

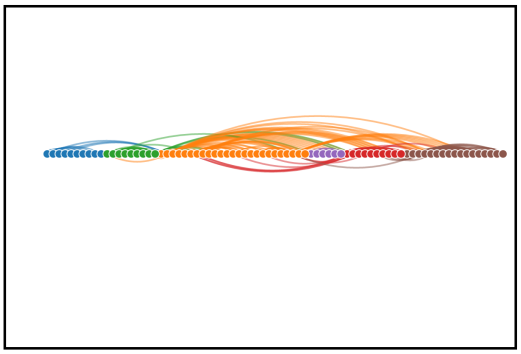


(a) CiteVis [213]

(b) Semantic Substrates [204]



(c) MatLink [118]

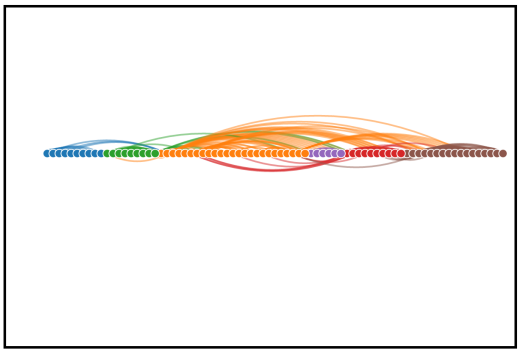


(d) Arc Diagram [243, 141]

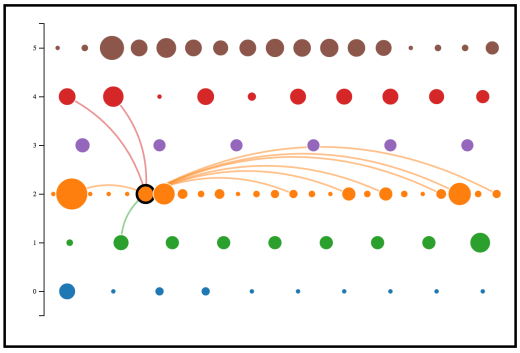
Figure 23: Sample abstract techniques where (∗, technique) transition matrix entries share operations.

GLOv2 transition matrix contain a single operation for positioning nodes along any axis (Cartesian x and y or polar ρ and θ) by any attribute (discrete or continuous). (I will discuss the other resulting differences between the GLOv1 and GLOv2 operations sets in Chapter 4 once I have presented the two models.)

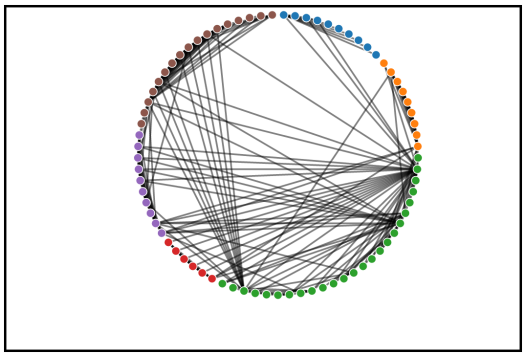
As part of the goal of minimizing the number of distinct operations, two interesting sub-cases arose. In the abstract Arc Diagram in Figure 24a, all of the nodes are evenly distributed horizontally across the display. In the abstract Semantic Substrates display in Figure 24b, the nodes are also evenly distributed horizontally across the display, but not as a single set of nodes. Instead, the nodes are evenly distributed within each row (where the row of a node is determined by a discrete property of the node). Similarly, while the abstract Circle Graph in Figure 24c evenly distributes the nodes along the angular polar (θ) axis, the abstract Cluster Circles display in



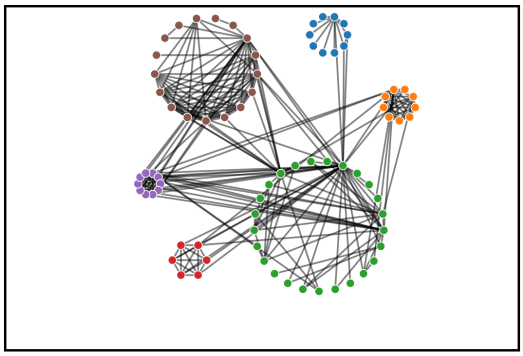
(a) Arc Diagram [243, 141]



(b) Semantic Substrates [204]



(c) Circle Graph [207]

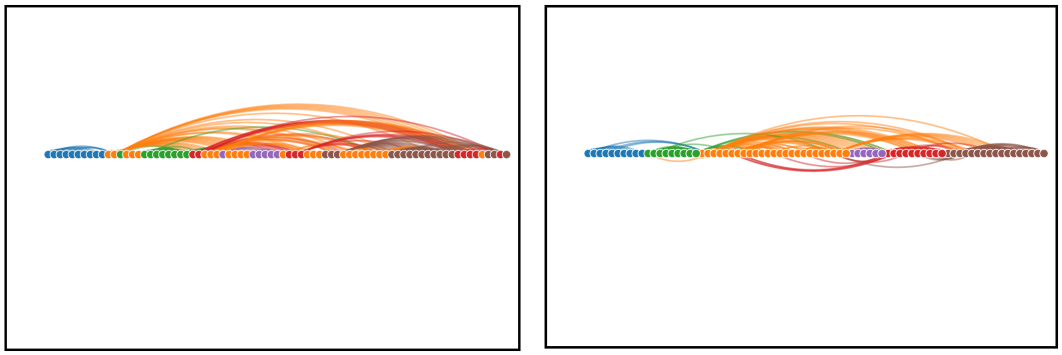


(d) Cluster Circles [69]

Figure 24: Four abstract techniques where (∗, technique) transition matrix entries contain the *evenly distribute nodes on* {axis} GLO with different mandatory parameters and optional parameters.

Figure 24d evenly distributes the nodes in each cluster around center points scattered around the display.

In all four abstract techniques, the operation being performed is the same (evenly distributing nodes along an axis) but how the operation is performed is clearly different. Between the Arc Diagram and the Circle Graph, the difference is simply which axis the GLO is applied to, and thus this axis becomes a **mandatory parameter** of the operation. Mandatory parameters are used when the operation cannot be performed without the information provided by the parameter. Other mandatory parameters include the {*attribute*} used to size nodes by an attribute in many (∗, Semantic Substrates) transition matrix entries and the {*discrete attribute(s)*} used to aggregate nodes or edges in many (∗, PivotGraph) entries.



(a) No sorting parameter (b) Sorted by discrete cluster attribute

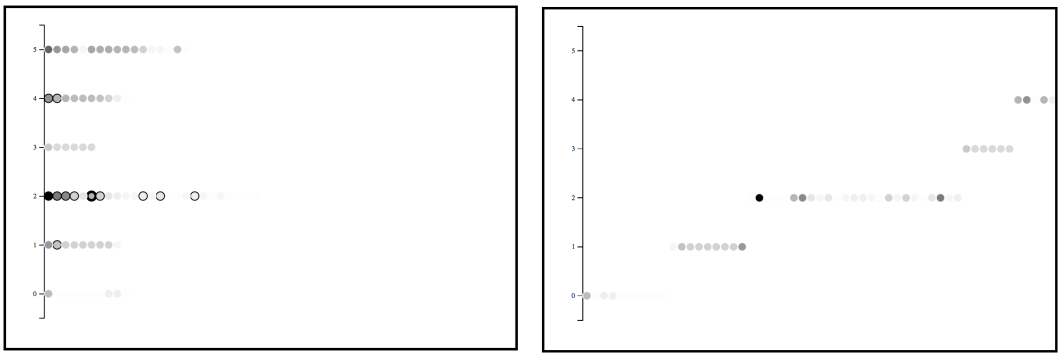
Figure 25: Abstract Arc Diagram representations with different optional sorting parameters.

In contrast to mandatory parameters, consider the two different Arc Diagram representations in Figure 25. Both displays are identical, except that the representation on the left has the nodes sorted seemingly randomly, while on the right the nodes are sorted by cluster (identified through color). The sort order on the left is not actually random; it is simply the order of the nodes as stored in memory. This sort ordering provides a useful default case when an alternative sort order is not provided. In other words, the attribute by which the nodes are sorted during the evenly distribute operation is an **optional parameter**. Another optional parameter is a binary **flag** signaling to **invert** the ordering of the sort. This optional parameter is used when positioning nodes by an attribute in order to 'flip' an axis (e.g. position nodes with larger values on the left and smaller values on the right when positioning nodes on the x axis).

On the other hand, while both the Arc Diagram and Semantic Substrates techniques in Figure 24 have the same x mandatory axis parameter, they are still different. In the Semantic Substrates display, the distribute nodes operation is being performed on the nodes of each cluster independent of all the other nodes. Note that this variation is not unique to the abstract Semantic Substrates technique. The abstract CiteVis representation in Figure 26a has nodes stacked on the x axis within each row

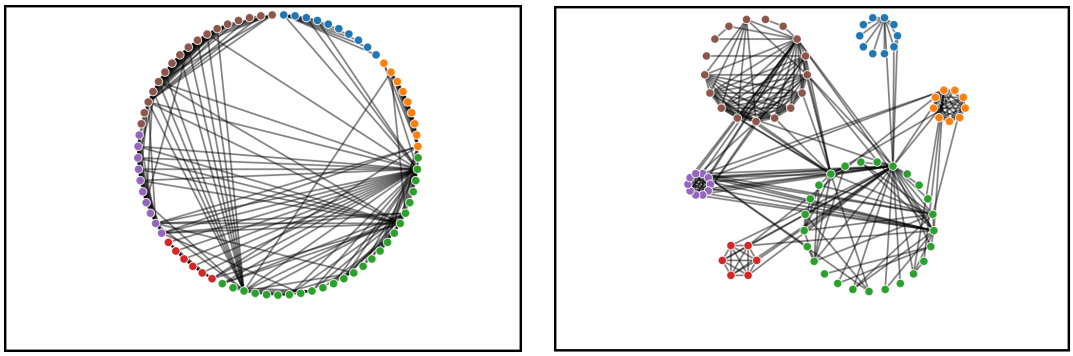
(again, determined by the discrete cluster attribute) whereas the display Figure 26b shows what would happen if the nodes were stacked on the x axis independent of the row. In GLOv1, I and my colleagues chose to treat the Arc Diagram and Semantic Substrates cases as distinct (i.e., one was evenly distributing nodes on an axis and the other was evenly distributing nodes on an axis within a discrete attribute). With the goal of minimizing the number of distinct operations in GLOv2, these two variations were considered the same operation (i.e. evenly distribute nodes on axis), but the Semantic Substrates case requires an optional **within parameter**. Using a within parameter applies the operation to each subgroup (as defined by the nodes that share values of the parameter attribute) independently. This same parameter can be used to align left all of the stacks in the CiteVis abstract technique and evenly distribute each of the rows in the Semantic Substrates abstract technique.

The difference between the Circle Graph and Cluster Circles abstract techniques (Figure 27) is distinct from the difference between Semantic Substrates and Arc Diagrams. The nodes in each row of the Semantic Substrates technique are still distributed across the entire display. In other words, the coordinate-space used to position the nodes within each row in the Semantic Substrates display is the same coordinate-space used to position the nodes in the single row in the Arc Diagram display. This is not the case with the two circle-based techniques. Instead, while the



(a) CiteVis [213] (b) Without within parameter

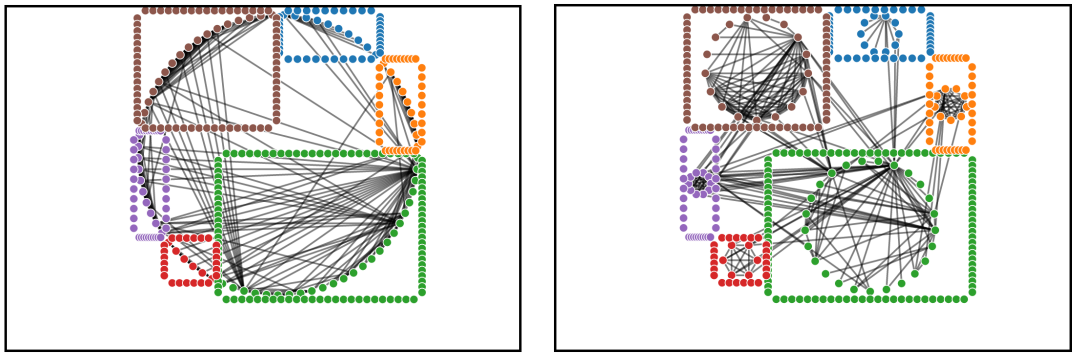
Figure 26: Representations with and without the optional within parameter.



(a) Circle Graph [207] (b) Cluster Circles [69]

Figure 27: Sample abstract techniques where (∗, technique) transition matrix entries use or do not use the group-by optional parameter.

Circle Graph distributes the nodes along a θ axis centered at the center point of the entire display space, the Cluster Circles representation distributes the nodes of each category along angular θ axes of sub-displays centered throughout the full display. In fact, the center points of each of these sub-displays is the center point of the bounding box of the nodes of each cluster in the Circle Graph (see Figure 28). Just as the within

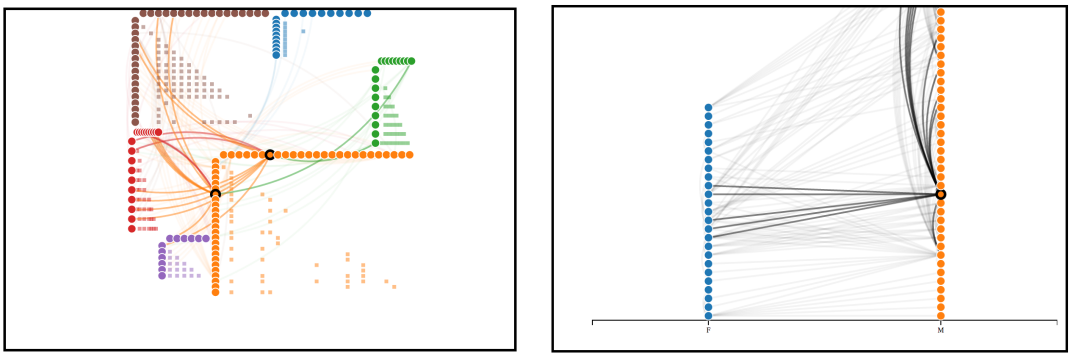


(a) Circle Graph [207] (b) Cluster Circles [69]

Figure 28: Sample abstract techniques where (∗, technique) transition matrix entries use or do not use the group-by optional parameter including bounding boxes determined by the Circle Graph node positions.

optional parameter signals to apply an operation to subsets of nodes using the full coordinate-space (i.e., the full display's x, y, ρ, and θ axes), a **group-by parameter** applies an operation to subsets of nodes within sub-coordinate spaces of the overall coordinate-space. Thus many (∗,Cluster Circles) transition matrix entries include an

Evenly distribute nodes on {axis} GLO adjusted by a discrete group-by parameter (in this case the cluster of the nodes).



(a) NodeTrix [117] (b) List View [199]

Figure 29: Sample abstract techniques where (∗, technique) transition matrix entries use a group-by optional parameter to display intra-group edges differently than inter-group edges.

Operations in transitions to the NodeTrix abstract technique in Figure 29a also utilize optional group-by parameters to position the nodes. But this abstract NodeTrix technique and the abstract List View representation in Figure 29b both also display edges differently depending on whether the edge's endpoints are in the same group or different groups (represented in these displays by the color of the nodes). In the NodeTrix display, intra-group edges are displayed as squares while inter-group edges are displayed as curved lines. In the List View display, intra-group edges are displayed as curved lines while inter-group edges are displayed as straight lines. Just as I attempted to minimize the number of distinct operations used in the GLOv2 transition matrix's entries, I also chose to minimize the number of optional parameters. Therefore, I chose to overload the group-by optional parameter to handle this case. Including a group-by parameter with an operation that changes how edges are displayed (such as a GLO to *display edges as squares*) only applies the operation to the intra-group edges. Thus, to transition to the List View technique might require first displaying edges as straight lines (without a group-by parameter), and then displaying the edges as curved lines (with the group-by parameter).

3.3 Inducing an Expected Data Model, Model of Visual Elements, and Set of Graph-Level Operations from the Transition Matrix

Once a transition matrix has been generated for a set of seed techniques, this transition matrix can then be used to identify both a visual element model of graph visualization and the set of operations to manipulate the visual element model. The matrix also implies the data model that these two components expect.

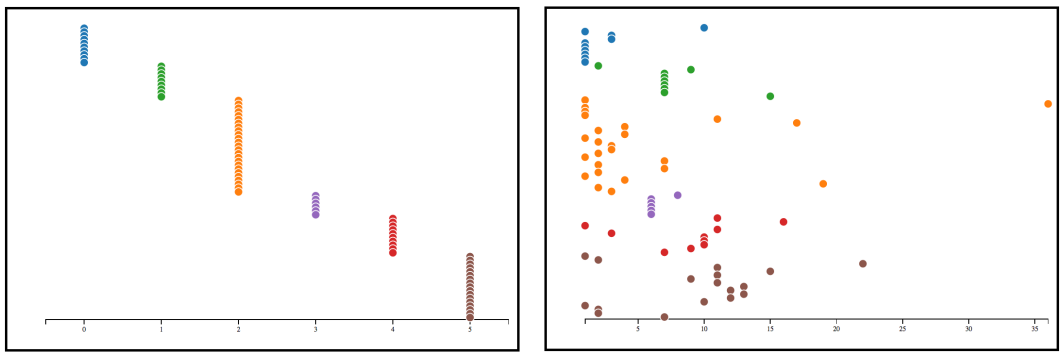
The operations set is the easiest to identify: the set is simply the union of the set of operations in each entry of the matrix. In other words, since each entry is a list of operations, one simply reduces each list to a set and then unions all of the sets together. As optional parameters such as the within or group-by parameters are ignored during this set-generation step, all of the optional parameters used in the matrix are then collected.

Identifying the visual element model is more subtle. The operations in the matrix entries imply a visual element model that the operations manipulate. For example, operations change representations of nodes and edges, which mandates that the model being manipulated have some notion of a glyph for nodes and edges. Another examples is that operations in many entries refer to axes, namely the Cartesian horizontal x axis, the Cartesian vertical y axis, the polar radial ρ axis, and the polar angular θ axis. Identifying all of these elements produces the GLO model's visual element model.

The transition matrix also helps to identify overarching properties of a model's set of operations. Most importantly, in both GLOv1 and GLOv2 each operation applies to all of the nodes and edges in a given set of nodes or edges. The set consists of either a glyph for each node or edge in the backing graph or glyphs of super-nodes and super-edges created from those nodes and edges. For example, during the (Force Directed Diagram, Matrix Plot) transition, aligning the second set of nodes to the left

aligned the entire set. The GLOv1 and GLOv2 models refer to these sets of nodes as **generations**. Even operations that affect groups of nodes independently from each other, such as operations modified with a group-by or within optional parameter, still apply to every node or edge in the generation.

This also begins to imply an underlying graph data model that the operations expect. If operations apply to every node (or every edge), this requires that every node/edge have the same set of attributes (though with likely different values of those attributes) in order to avoid referencing an attribute that any one element does not have. In addition, the operations sometimes differentiate discrete attributes (whether ordinal or categorical) from continuous (quantitative) attributes. For example, the *position nodes by* {attribute}operation positions nodes relatively along an axis for continuous attributes and into evenly distributed sets for discrete attributes (see Figure 30). Furthermore, aggregating edges by properties of their endpoints implies data visibility requirements (namely that edges can know the decorations of their endpoints).



(a) Discrete (b) Continuous

Figure 30: The GLOv2 *position nodes by* {attr} operation positions node glyphs evenly along the axis with discrete parameters and relatively along the axis with continuous parameters.

3.4 Augmenting the Operations Set

The final step is to augment the set of operations derived from the transition matrix with obvious missing operations. Given a large and varied enough seed technique set, this step should not result in many additional operations. In the case of GLOv2, this required the addition of only three operations compared to the sixty-nine operations that occurred in transitions.

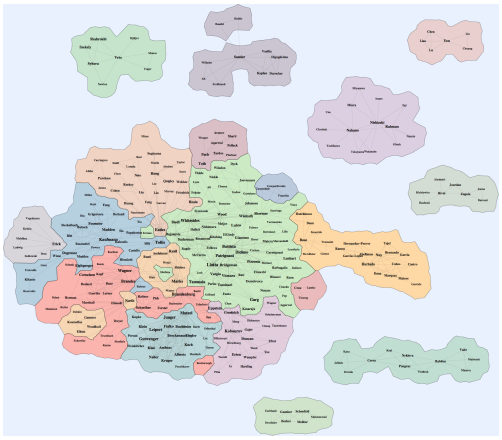


Figure 31: GLOv2 GMap seed technique from [94].

First, during the GLOv2 identification process, transitions to the GMap seed technique (Figure 31) included drawing convex hulls around nodes and then the operation *color convex hulls by* {attribute}. Operations for coloring nodes and edges by attributes also appeared in transitions, as did operations for coloring node and edge glyphs by constants. Thus, for consistency, the *color convex hulls by constant* GLO was added to the set.

Second, many transitions to the CiteVis seed technique (Figure 32) required the *position nodes evenly stacked* {direction} operation. The paired operations *evenly distribute nodes on* {axis} and *position nodes on* {axis} *by* {attribute} were similarly required for various transitions. However, a stacking GLO equivalent to the relative positioning *position nodes on* {axis} *by* {attribute} did not appear in any transitions. Thus, I added a *position nodes stacked* {direction} *by* {attr} operation that stacks

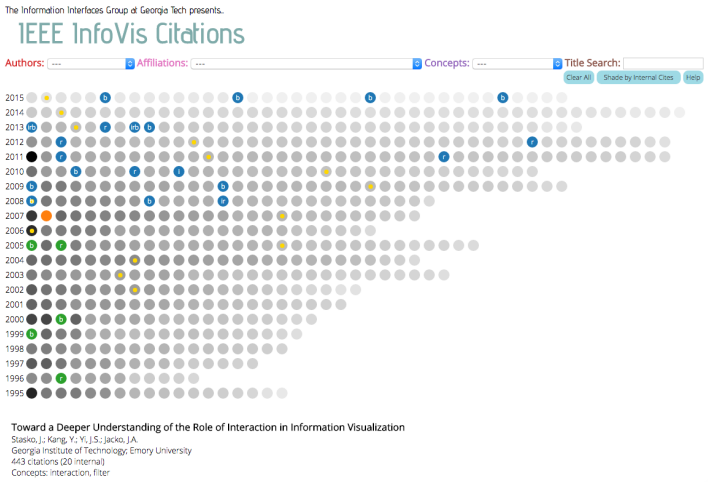


Figure 32: GLOv2 CiteVis seed technique based on [213].

the nodes distances apart relative to an attribute of the data.

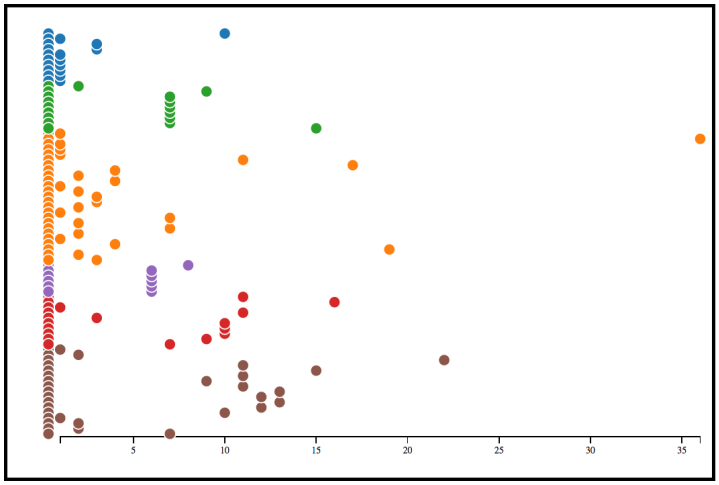


Figure 33: Demonstration of axis uncertainty.

Third, no GLOv2 seed technique that includes axis labels has multiple generations of nodes referring to the same axis. This meant that among these techniques there was no uncertainty as to which generation of nodes any given axis labels should refer. However, consider the display in Figure 33. The axis labels should refer to the generation of nodes positioned by an attribute rather than the left-aligned generation, but that may need to be communicated through a GLO. Thus, the *set* {axis} *axis node generation* {num} GLO was added to the operations set to provide this information.

CHAPTER IV

THE GRAPH-LEVEL OPERATIONS MODEL

In Chapter 3, I described a method for inducing a model of graph visualization from a set of seed techniques and two sets of seed techniques consisting of 6 and 29 techniques. In this chapter, I present the models that resulted from applying the method to the two sets of seed techniques (GLOv1 and GLOv2, respectively). As mentioned in the last chapter, each model consist of two components: a visual elements model and the set of graph-level operations for manipulating that visual elements model.

I begin by describing the data model that the GLO models expect. I then describe the GLO visual element model including glyphs, generations, canvases, axes, and the GLO Display. I then present the two sets of operations discuss the differences between them. I will then describe properties of GLOs that enable them to function as a domain-specific graph visualization language. Finally, I describe how we can use lists of operations to describe techniques, starting with the seed techniques.

4.1 Graph Data Model

Before discussing the graph-level operations models directly, let me first describe the graph data model that the models expect. At the highest level, the GLO models expect a graph consisting of a set of nodes and a set of edges between those nodes. Note, however, that such a graph can have no nodes and no edges or the graph can have nodes but no edges. The latter of these two cases is schematically equivalent to tabular data if each row of the data is considered a node.

Each node must have an **identifier (id)** unique to the node relative to the other nodes' identifiers and these identifiers must be sortable. Equivalently, each edge must also have a unique, sortable identifier. These identifiers are used as the default sorting

attribute for GLOs (such as *evenly distribute nodes on* {axis}) when no optional sort-by parameter is provided. Two simple and effective identifier schemes are to label the nodes and edges with the address (e.g. 0x440B...0x44AF) or index of the element's position (e.g. 1...n) in memory.

Each edge must have some reference to its endpoints. These references must be in the form of a source reference and a target reference. For directed graphs, these references map directly to the source and target nodes of the edge. Undirected graphs, however, must be converted to directed graphs. There are two simple algorithms to perform this conversion. The first algorithm is to set one endpoint of each undirected edge to be the source and the other to be the target (for example, by selecting the node with the lower id as the source). The advantage of this algorithm is that the degree of each node in the adjusted graph is equal to the degree of each node in the unadjusted graph and the number of edges in both graphs remains the same. On the other hand, the in-degree and out-degree of each node is not guaranteed to remain the same. The second algorithm is to create a second copy of each non-self-edge in the graph, with each pair of otherwise-duplicate edges having reverse source and target references. This second algorithm reverses the advantages and disadvantages of the first algorithm—the original nodes and the new nodes have the same in-degrees and out-degrees, but the nodes in the new graph have different degrees than their counterparts and the total number of edges has changed.

Each node and edge can be decorated with **attributes** (or **properties**). These attributes can be **discrete attributes** (where the values of the attribute are drawn from a finite set of values) or **continuous attributes** (where the values of the attributes are drawn from an infinite set or series of values). (Using Card, Mackinlay, and Shneiderman's taxonomy [52], discrete attributes cover both nominal and ordinal data types and continuous attributes are quantitative data types.) Unless explicitly specified, continuous or discrete attributes can be passed as attribute parameters.

Certain GLOs (and optional parameters) do explicitly depend on accepting only discrete attributes. Namely, aggregation operations require any number of discrete attributes to be passed, the filter-partition operation takes a discrete parameter, and the group-by and within optional parameters must be discrete attributes. All of these operations and optional parameters concern partitioning the set nodes into categories or clusters, which can be done without additional information using discrete attributes. In contrast, to reduce a continuous attribute to a discrete attribute requires binning the values. However, this introduces uncertainty. Most importantly, how many bins should be used? To circumvent this uncertainty, both models expect that if binning a continuous attribute is necessary, that the attribute be pre-binned and represented as a discrete attribute before any operation is applied.

As I discuss later in this chapter, every operation applies to glyphs representing every node or edge in the graph. Therefore, any nodes or edges lacking an attribute that is referenced will lead to an unpredictable response. Therefore, all nodes must have the same set of attributes and all edges must have the same set of attributes. Of course, the values of these attributes are not expected to be the same. This requirement can be limiting, for example, if the data for a certain attribute of a specific node or edge does not exist. For discrete attributes, missing values can be represented as a "missing" (or **null**) value that will simply be considered as another value of the attribute. Missing continuous attributes pose a larger challenge. Unlike discrete attributes, a null value for continuous attribute would cause issues whenever a visualization of the data is rendered. Thus, each missing value must be replaced. The replacement value could be a statistic (such as min, max, mean, median, or mode) over the non-missing values or could be a pre-determined value (such as -1 among otherwise non-negative values). Regardless, all of these replacements must be performed before a GLO is applied using the data.

Finally, from a data visibility perspective, the GLO models expect that each node

can access the values of its own attributes, its in- and out-edges, and the nodes making up its in- and out-neighborhoods. (Note that there is no expectation that a node can access the attributes of these incident edges and neighboring nodes, only the edges and nodes themselves.) Conversely, the model expects that each edge can access the values of its own attributes, its two endpoint nodes, and the attributes of its endpoints.

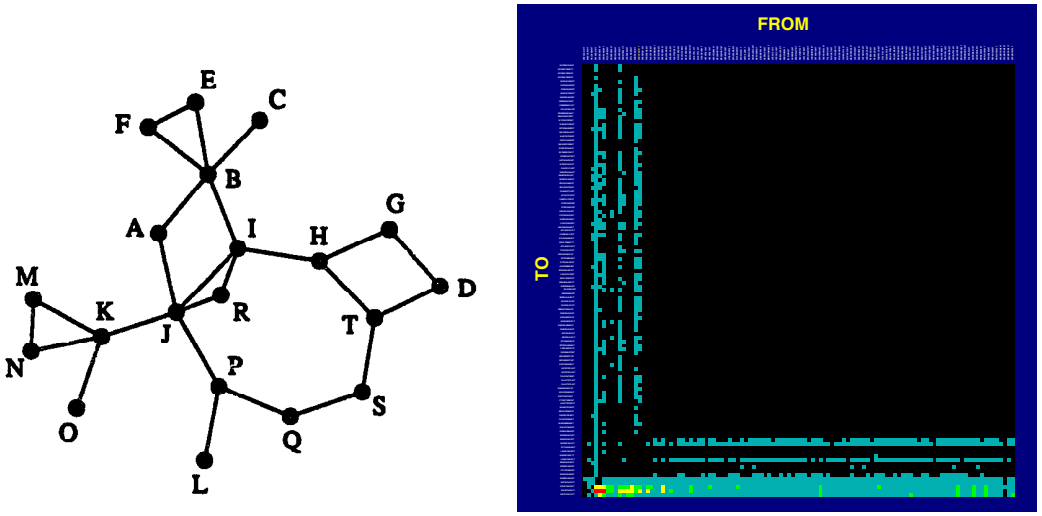
4.2 GLO Visual Element Model

As I described in the previous chapter, various changes are made to representations in order to transition between them. The elements (and their properties) manipulated through those changes describes a visual element model. In this section I will describe the GLO visual element model induced from the GLOv2 transition matrix. This visual element model serves as a super-set of the GLOv1 visual element model. (I will postpone describing the precise differences between the two visual element models until after I have presented both the GLOv2 visual elements model and both operations sets.)

4.2.1 Glyphs

The primary unit of the GLO visual element model is the **glyph**. A glyph is a visual representation of a given node or edge in the graph data, called the **backing node** or **backing edge** of the glyph. Nodes are represented by **node glyphs**, while edges are represented by **edge glyphs**.

Glyphs can be displayed in a variety of ways. For example, consider the Force-Directed Layout and Matrix Plot GLOv2 seed techniques in Figure 34. The Force-Directed Layout's nodes are represented as circles, while its edges are represented as straight lines. Alternatively, the Matrix Plot's nodes are represented as textual labels, while its edges are represented as small squares colored by an attribute of the data.



(a) Force-Directed Layout from [138] (b) Matrix Plot from [34]

Figure 34: GLOv2 seed techniques with differing node and edge glyph displays.

Node glyphs under GLOv2 have one of four **display modes**: circles, squares, bars, or textual labels. Edge glyphs under GLOv2 have one of six: straight lines, curved lines, squares, textual labels, bars, or right angles. Furthermore each node and edge glyph has a size and a color.

Beyond its display mode, each glyph has three additional visual properties: **position**, **size**, and **color**.

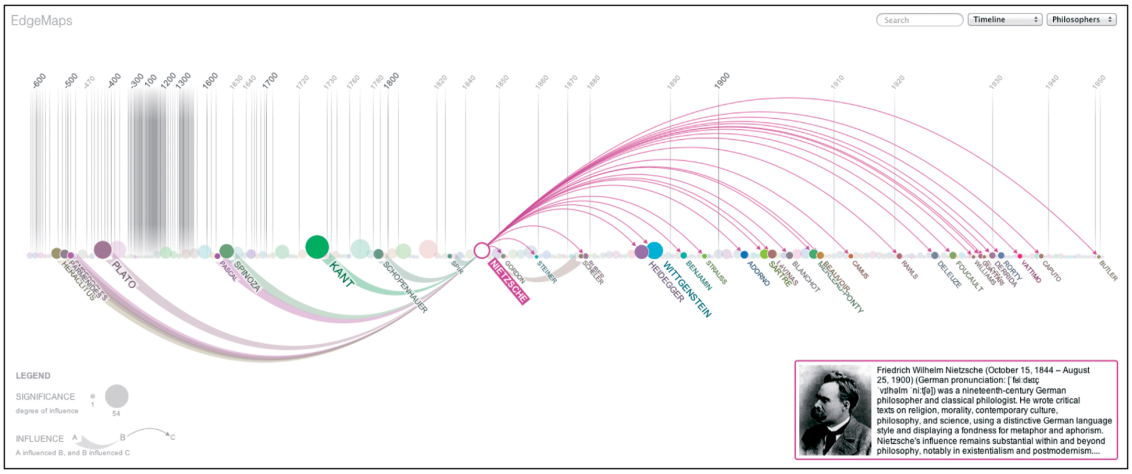


Figure 35: GLOv2 EdgeMap B seed technique from [74].

In addition, each glyph has an **interaction mode**. Interaction is enabled through

a **selected node** model. For example, in the GLOv2 Edgemap B seed technique in Figure 35, the Nietzsche node is the selected node. In this display, the node glyphs' interaction mode (highlight neighbors) causes node glyphs to highlight if the backing node is a neighbor of the selected node and fade out if the backing node is not a neighbor of the selected node. The interaction mode of the edge glyphs in this display (show in-out edges) signals for edge glyphs to be visible if the backing edge is incident to the selected node and hidden otherwise.

The GLOv2 model does not explicitly specify how the selected node is determined. Instead, this is left to each implementation of the model. For example, the Javascript GLOv2 implementation I present in the next chapter (GLO.js) considers the node backing a node glyph the analyst is mousing over the selected node. Alternate selection mechanisms could include mouse clicking, finger tapping, or even eye tracking. However, the GLOv2 model does explicitly require that either no nodes are selected or a single node is selected. The model explicitly does not support multi-selection. Consider the Edgemap B seed technique again. The edges are displayed such that in-edges of the selected node (thin lines) are displayed differently than the out-edges of the selected node (thin lines). These edge glyphs are not sized by an attribute of the edges, but rather by their relationship to the selected node. If there were two selected nodes that were endpoints of the same edge, this would lead to an undefined state for the edge where it was both an in-edge and an out-edge of a selected node. (I will discuss in Chapter 6 how multi-selection within a GLO model is an open challenge.)

GLOv2 supports three interaction modes for node glyphs: no interaction, highlight neighbors, and highlight-in-out-neighbors.

In the no interaction case, selecting a node (again, however the implementation has defined selection) has no effect on the representation. In the highlight neighbors mode, node glyphs are shown fully rendered when no node is selected and node glyphs are in some way highlighted when they are neighbors of a selected node. In the case

of the Edgemap B seed technique above, this was handled by reducing the saturation of non-highlighted nodes. However, a given GLO implementation can perform this highlighting however it deems fit. For example, in my GLO.js implementation, a thin black ring is drawn around highlighted nodes. The critical aspect is that the neighborhood of the selected node be visually distinct from the non-neighbor node glyphs in the generation.

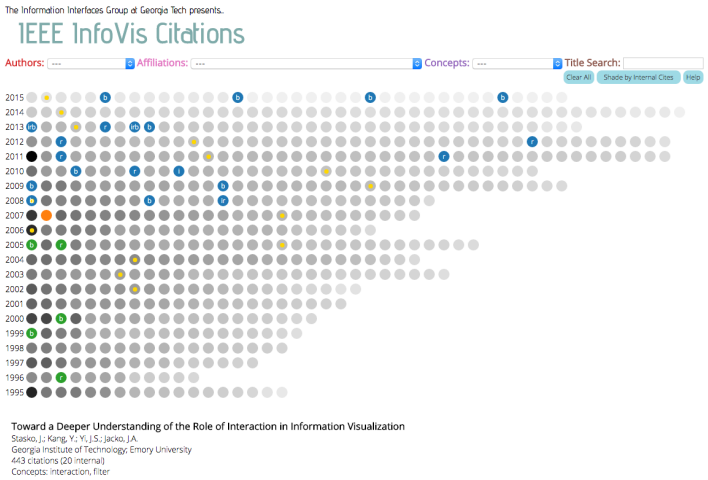
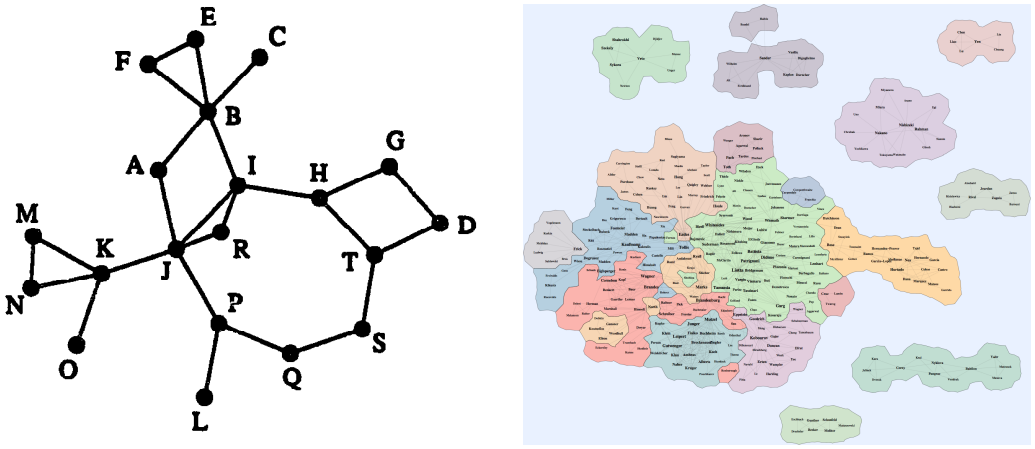


Figure 36: GLOv2 Citevis seed technique based on [213].

In the highlight-in-out-neighbors case, exemplified through the CiteVis GLOv2 seed technique in Figure 36, nodes are highlighted differently if they are in-nodes or out-nodes of the selected node. In the seed technique, this is handled by coloring in-nodes green and out-nodes blue. However, once again, the specific manner in which the node are visually modified is left to the implementation.

Edges glyphs in GLOv2 have six interaction modes: show none, show all, show faded, show incident, show in-out, and show faded and incident.

The show none, show all, and show faded interaction modes remove any interaction from the edge glyphs. In the show none case, all edge glyphs in the generation are hidden. This is the edge interaction mode of the CiteVis seed technique above. Note that the edge glyphs are not removed when they are hidden. Properties of the glyphs can still be manipulated and will be reflected should the interaction mode



(a) Force-Directed Layout from [138] (b) GMap from [94]

Figure 37: GLOv2 Force-Directed Layout seed technique and GLOv2 GMap seed technique. The straight-line edge glyphs in the Force-Directed Layout have the show all edges interaction mode, while the straight-line edge glyphs in the GMap technique have the show faded interaction mode.

change. In the show all case, each edge glyph is displayed fully rendered. In the show faded case, every edge glyph is displayed with a lower saturation. For example, in Figure 37, compare the fully-rendered straight-line edges of the Force-Directed Layout seed technique (show all edges interaction mode) with those of the GMap seed technique (show faded interaction mode).

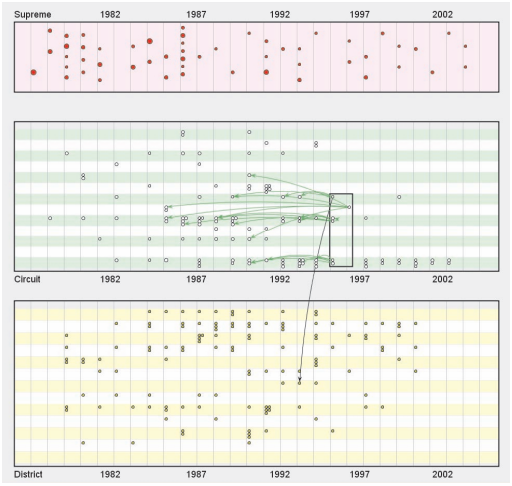


Figure 38: GLOv2 Semantic Substrates seed technique from [204] with edge glyphs in the show incident edges display mode.

In the show incident interaction mode, an edge glyph is only displayed when one of the endpoints of the edge glyph's backing edge is selected. This is the interaction mode used by the Semantic Substrates GLOv2 seed technique in Figure 38.

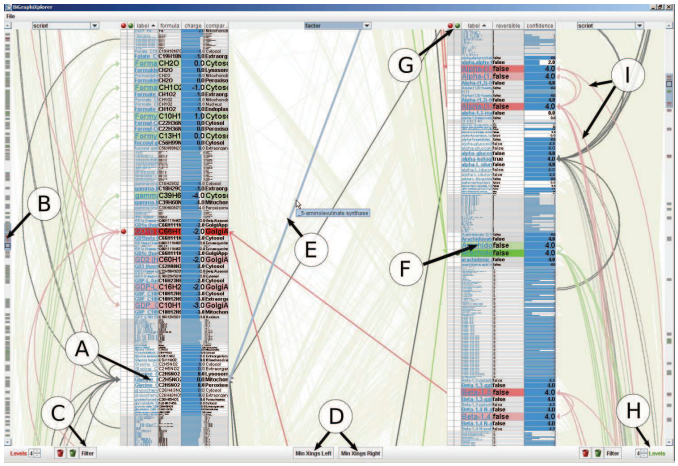
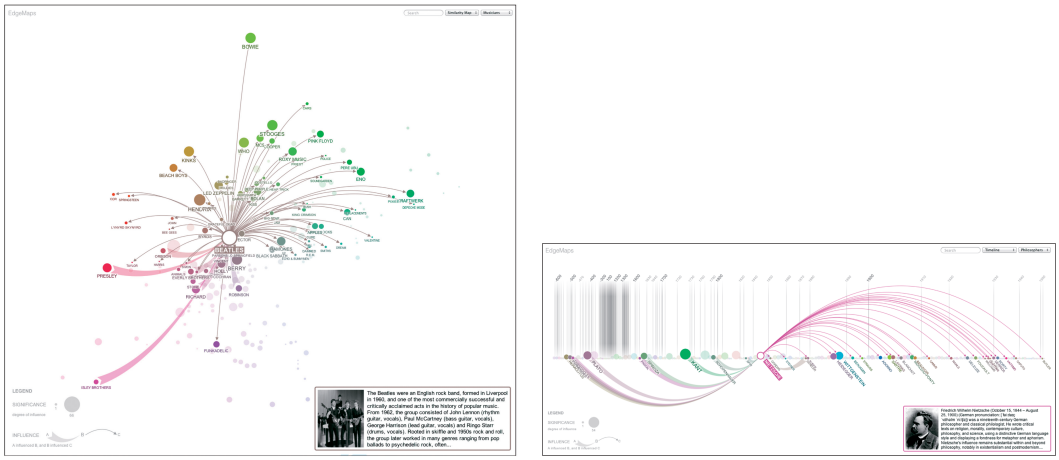


Figure 39: GLOv2 List View seed technique from [199]. Edge glyphs utilize the show faded-and-incident interaction mode.

The List View GLOv2 seed technique in Figure 39 utilizes the show faded-and-incident interaction mode. In the show faded-and-incident interaction mode, edge glyphs are displayed as faded (i.e. with a lower saturation) both when there is no selected node or when a selected node is not an endpoint of the edge glyph's backing edge. In the case where the backing edge is an endpoint of the selected node, then the edge glyph is drawn fully rendered.

Finally, the Edgemap A and Edgemap B GLOv2 seed techniques in Figure 40 demonstrate the show in-out interaction mode. Like the show incident interaction mode, under the show in-out interaction mode, edges are hidden unless there is a selected node. When a node is selected, edge glyphs representing in-edges of the node are rendered one way, while out-edges of the node are rendered another way. It is again left to each implementation how the edges are rendered differently.



(a) Edgemap A (b) Edgemap B

Figure 40: GLOv2 EdgeMap A and Edgemap B seed techniques from [74]. Both techniques utilize the in-out edges interaction mode where in edges of the selected node are displayed differently from out edges.

4.2.2 Generations

At the next level of the visual element model, glyphs are grouped into **generations**. To summarize the logic behind generations from Chapter 3, changes between techniques are described as changes to sets of glyphs, not to individual glyphs. Specifically, operations that manipulate the visual element model operate on sets of one glyph per node or one glyph per edge. Therefore, in a given generation, there is one glyph per node or edge in the graph. I.e., in a given **node generation** there is a single node glyph for each **backing node** in the underlying graph data, and in a given **edge generation** there is a single edge glyph for each **backing edge** in the underlying graph data.

The backing nodes and edges of a generation may not necessarily be the nodes and edges of the underlying graph data. As I demonstrated during the (Semantic Substrates, PivotGraph) transition, nodes and edges can be **aggregated** into **super-nodes** and **super-edges**. These super-elements represent multiple nodes or edges. A generation of **super-node glyphs** backed by super-nodes will therefore have fewer glyphs than the total number of nodes in the original graph and equivalently for

super-edge glyphs.

When nodes or edges are aggregated to form super-nodes and super-edges, the attributes of the aggregated nodes and edges are shared by the super-element. In other words, super-elements have the same data attributes as the original graph's nodes and edges. For continuous attributes, super-elements summarize the original element's attributes using a summary statistic such as mean, median, min, or max. The choice of summary statistic is therefore a mandatory parameter of aggregation GLOs. For discrete attributes, the super-elements have the same values as the most common value among the original elements. If there is a tie between values, the value associated with the element with the lowest identifier among the elements with one of the candidate values is chosen.

Every node glyph and edge glyph backed by an element of the original graph includes an immutable **count attribute** with a value of 1. This value signifies that the glyph represents a single element of the original graph. Super-elements have count attributes equal to the number of original elements they represent. For example, if three original nodes are aggregated into a super node, then the super-node has a count attribute value of 3. Techniques such as the PivotGraph GLOv2 seed technique depend on this value to properly size its super-node-backed node glyphs. Furthermore, should super-elements be aggregated, the new (super-)super-elements have count attributes equal to the sum of the count attributes of the super-elements.

Super-nodes and super-edges can be **deaggregated** to restore the nodes and edges or super-nodes and super-edges (and the associated glyphs) that made up the aggregates. Note that the restored node and edge glyphs may have been modified. Operations performed on a generation of super-node glyphs or super-edge glyphs are reflected in the glyphs of the original nodes and edges. For example, any operations that change properties of a super-glyph (display mode, position, size, color, or interaction mode) will be reflected in the restored nodes. For example, if a super-glyph

is colored red by an operation, then all of the node glyphs aggregated to form the super-glyph will be restored as red-colored.

In the (Force-Directed Layout, Matrix Plot) transition in the previous chapter, I showed that some techniques (such as Matrix Plots) require multiple glyphs to represent the same graph elements. Furthermore, I showed that these additional generations of glyphs can be introduced into a display unambiguously by **cloning** an existing generation. Cloning a generation creates a new generation, with its own set of glyphs, each of which is a duplicate of one of the original generation's glyphs. With the exception of a **default generation**, generations can only be created through cloning. A default generation consists of a single generation of unaggregated elements. Notably, the new generation's glyphs are still backed by the same underlying nodes, edges, super-nodes, or super-edges of the generation being cloned. In other words, the glyphs in cloned generations can still have properties such as size and color derived from data attributes.

Recall that interaction is handled by a selected node model, as opposed to a selected node glyph model. This means that any interaction with a glyph representing a node will simulate interaction on (i.e., brush to) all other glyphs representing the node.

Just as node and edge glyphs have properties, so too do edge generations. Specifically, each edge generation has a **source node generation** and **target node generation**. Consider the Edge-Label-Centric GLOv2 seed technique in Figure 41. The technique consists of two generations of nodes, one on the left and one on the right. Edges (represented by the red line glyphs) are drawn from the glyph representing the source node in the left generation to the glyph representing the target node in the right generation. In order to correctly draw these edges, the edge generation knows its **source node generation** and **target node generation**. Edges displayed

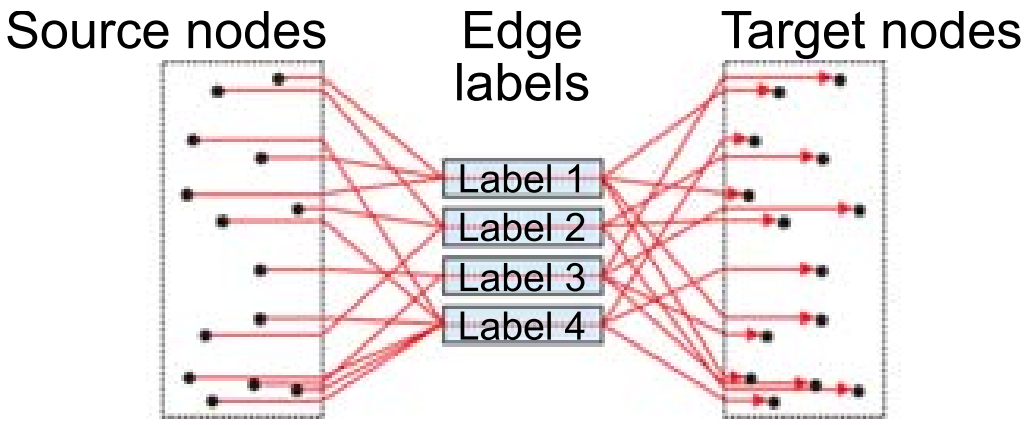


Figure 41: GLOv2 Edge-Label-Centric seed technique from [182]. The red straight-line edges are drawn from the source generation on the left to the target generation on the right through the waypoint generation of super-edge glyphs in the center.

as straight or curved lines or right-angles can then be drawn between the appropriate source and target glyphs. Edges displayed as squares or attribute labels can be positioned relative to the source and target node glyphs.

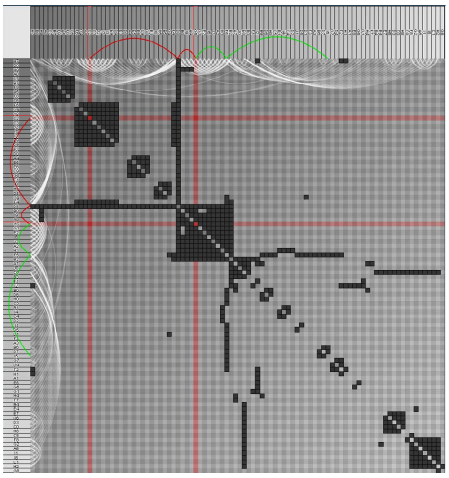


Figure 42: GLOv2 MatLink seed technique from [118] demonstrates source and target node generations.

For example, consider the MatLink GLOv2 seed technique in Figure 42. One generation of curved edges is drawn with the nodes on the left-side as both the source and target generation. Similarly, the other generation of curved edges is drawn with the nodes on the top as both the source and target generation. In contrast, the nodes

displayed as squares making up the majority of the display are drawn such that they share a y coordinate value with their source nodes on the left and share an x coordinate value with the target node in the top generation. Therefore, the left-hand node generation is the source generation of the square edge generation, while the top node generation is its target generation.

An additional property of edge generations are **waypoint generations**. A waypoint generation is a different edge generation through which edge glyphs with line-based display modes (straight lines and curved lines) are drawn. For example, in in Figure 41, the generation of super-edge glyphs with display mode labels in the center of the display is an edge waypoint generation of the generation of red straight-line edge glyphs. By default, edge generations do not have a waypoint generation.

4.2.3 Canvases

The third level of the GLOv2 visual element model is the **canvas**, on which the glyphs of each generation are drawn. Each canvas has four **axes**: x, y, ρ, and θ. The x and y axes map to the Cartesian coordinate system (i.e. lower-left origin). Each canvas has three preset values for the x axis (left, center, and right) and three preset values for the y axis (top, middle, and bottom). Conversely, the radial ρ and angular θ axes define a polar coordinate system around the (center,middle) point in the Cartesian coordinate system. Single preset (**constant**) values for ρ and θ are defined by each implementation of the GLOv2 model. The GLO.js implementation uses the positive y direction as its constant θ value and quarter the shorter of the x or y axis lengths as its constant ρ value.

Canvases can also display axis labels along their edges. (For example, see the Scatternet GLOv2 seed technique in Figure 43.) If node glyphs are repositioned along an axis by a different attribute, the axis labels update accordingly. Similar to each edge generation have source and target node generations, each axis on each

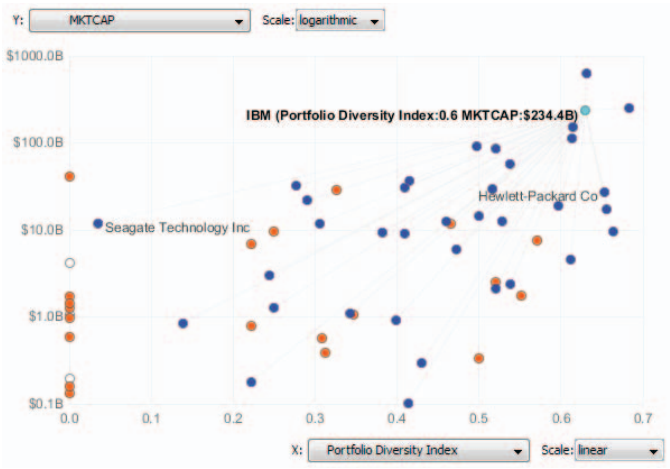


Figure 43: GLOv2 ScatterNet seed technique from [27] demonstrates axis labels.

canvas also has an associated node generation. In this way, each axis knows when to update the labels.

4.2.4 GLO Display

The highest level of the GLO visual element model is the **GLO Display**. Unlike canvases, generations, and glyphs, there is only one GLO Display. The GLO Display consists of canvases arranged in a grid along the x and y axes. Similar to how new node and edge generations can be created by cloning existing node and edge generations, new canvases can be created by **partitioning** existing canvases.

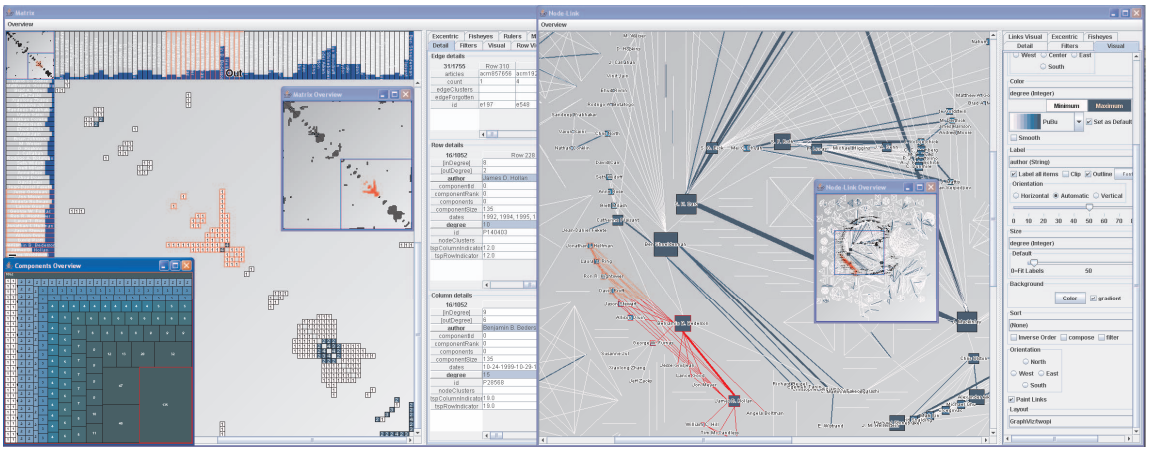


Figure 44: GLOv2 MatrixExplorer seed technique from [116] demonstrates canvas partitioning within a GLO Display.

Partitioning a canvas divides the area of the GLO Display originally designed to the canvas into n equal-sized partitions along an axis. Each partition is a clone of the original canvas except for the smaller dimensions. The partitions each contain clones of all the node and edge generations in the original canvas. Once a canvas is partitioned, each of the canvases can be modified independently. For example, to transition to the MatrixExplorer GLOv2 seed technique in Figure 44 requires partitioning the GLO Display into two canvases. After the partitioning, each of the two canvases can be adjusted independently, one into a Force-Directed Layout and the other into a Matrix Plot.

Recall that interaction is derived from selected nodes (rather than selected node glyphs). Thus, interaction does not simply brush to other glyphs backed by the selected node in a single canvas, but rather to node glyphs backed by the selected node in all canvases. In this way, the GLO visual elements model brushing and linking between different representations, such as the two displays that make up the MatrixExplorer.

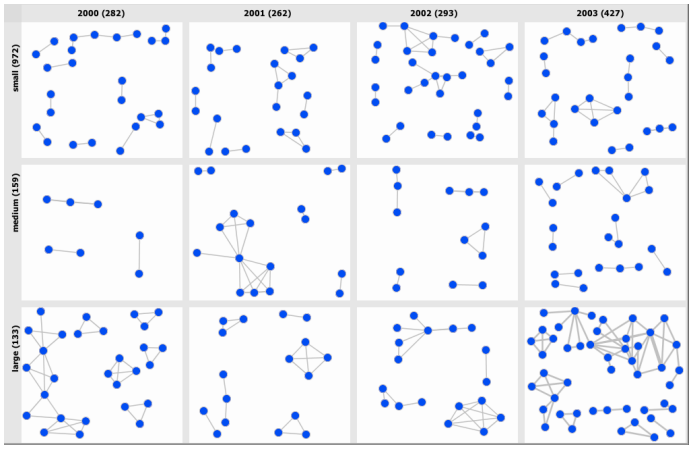


Figure 45: GLOv2 Attribute Matrix seed technique from [153] demonstrates filter-partitioning canvases and meta-axis labels.

While partitioning canvases into a specific number of smaller canvases enables

linked representations, other techniques make use of Tufte's principle of small multiples [229]. For example, consider the Attribute Matrix GLOv2 seed technique in Figure 45. The Attribute Matrix displays a grid of small multiples of Force-Directed Layouts, where each small multiple consists of the subgraph induced by the nodes that have the same pair of properties. GLOv2 supports this small multiples functionality through **filter-partitioning**. Filter-partitioning not only partitions the canvas, but does so by splitting each node generation on the canvas based on a discrete value of the data. Similar to axis labels on canvases, the GLO Display can display **meta-axes** when the canvases themselves are data-driven in this way.

4.3 Operation Sets

While the visual element model represents one component of a full GLO model, the other component is a set of graph-level operations that modify an instance of the visual element model. These operations represent the various steps used to transition between techniques during the identification process, such as sizing nodes by an attribute, cloning a generation, or displaying edges as squares.

Rather than list the operations here, I instead refer the reader to the full set of 34 GLOv1 operations in Appendix C and the full set of 72 GLOv2 operations in Appendix D.

In this space, let me instead briefly describe the categories of operations in each model.

The 34-element GLOv1 operations set can be broken down into five overarching categories. **Positioning** operations adjust the coordinate positions of nodes glyphs in the GLO Display. There are operations that modify **element properties**, such as how edges are displayed or whether nodes are constant-sized or sized by an attribute. This category includes operations for specifying interaction modes. **Cloning** operations allow the visualization to represent the same set of nodes with multiple glyphs

(and remove those extra glyphs.) The model provides operations for **aggregating** nodes and edges into super-nodes and super-edges. Finally, GLOv1 enables showing and hiding **axes labels** that update as the positions of the nodes are changed.

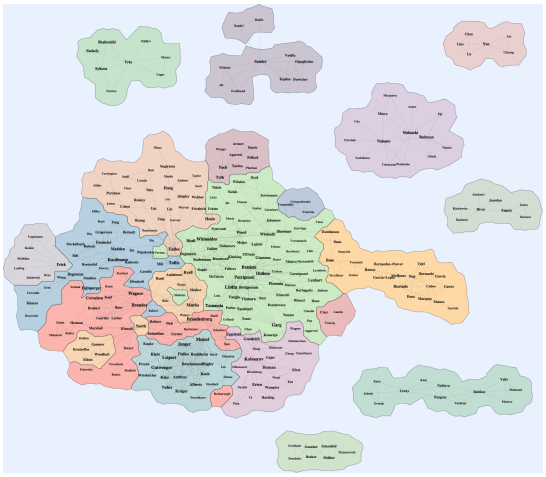


Figure 46: GLOv2 GMap seed technique from [94] utilizing convex hulls.

The 72-element GLOv2 operations set can be broken down into eight categories. Some are the same as GLOv1, such as those for adjusting node (and edge) glyph **positions** and other **visual properties**. GLOv2 also supports **cloning** nodes (and edges), **aggregating** nodes (and edges), and displaying **axes**. However, GLOv2 also adds support for **partitioning canvases** within the GLO Display, drawing translucent **convex hulls** around groups of nodes (e.g. see the GMap GLOv2 seed technique in Figure 46), and a wider variety of **interaction** operations to change the interaction modes of the glyphs.

4.4 Language Properties of Graph-Level Operations

If one considers a set of operations as an application programming interface (API), 72 unique GLOv2 operations (functions) over 8 categories (classes) is actually comparatively small. As I discuss in the next chapter, utilizing GLOs as an API is one of the primary advantages of GLO models. Thus, in this section, I want to define properties of the domain-specific language that such an API represents.

First, each canvas has an **active node generation** and an **active edge generation** and each GLO Display has one or more **active canvases**. Operations that act on a canvas will only act on the active canvas(es). Operations that act on a generation will (by default) only act on the active generation (node or edge as appropriate) of the active canvas(es). This is, in my opinion, the most important feature of the GLO DSL—operations apply to every glyph in a generation. No operations act on a single glyph. In other words, if the *size nodes by* {attr} GLO is applied, it is applied to the entire active *generation* rather than to a specific glyph.

As I discussed in the previous chapter, operations can have **mandatory parameters** as well as **optional parameters**. For example, the *evenly distribute nodes on* {axis} operation has a mandatory {*axis*} parameter that can be any of the four axes supported by GLOv2: x, y, ρ, or θ. This GLO also recognizes two optional parameters. A **sort-by** attribute parameter that distributes the nodes in order of the nodes' attribute values. An **invert** flag parameter reverses the order of the sorting. The value of the {*direction*} parameter in GLOs such as *align nodes* {direction} can be top, middle, bottom, left, right, or center.

With two notable exceptions, all mandatory parameters take a single value. The exceptions to this are the parameters of the *aggregate nodes by* {discrete attributes} *using* {method} and *aggregate edges by* {discrete attributes} *using* {method} GLOs. Rather than a single attribute parameter, these two operations can take one or more discrete attributes as a set of parameters. This enables aggregation by more than a single attribute such as in the PivotGraph transition in the previous chapter.

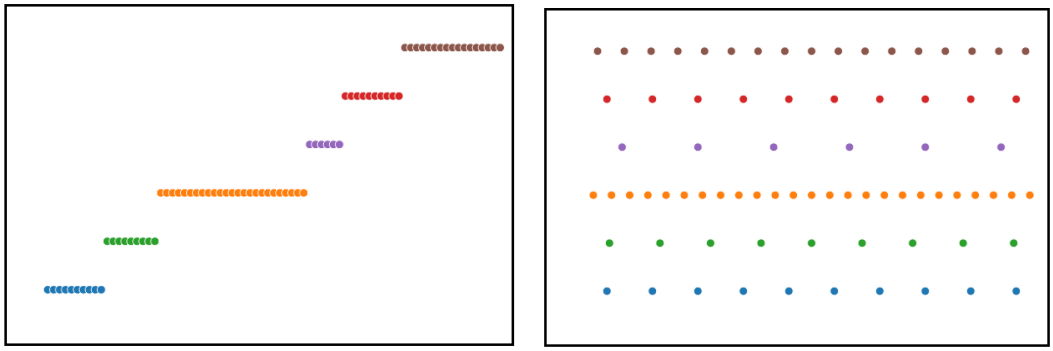
In the GLOv2 operations list in Appendix D, mandatory parameters are defined for those operations that require them. For consistency, any optional parameters may be passed to any operation. However, operations that do not utilize a given optional parameter can simply ignore that parameter. For example, the *clone nodes* operation would ignore any sort-by or invert optional parameters passed to it.

Figure 47: GLOv2 DOSA seed technique from [232] utilizes the all-canvases optional parameter.

Two specific optional parameters are utilized for every operation. Recall that GLOs only apply to either the active canvas or the active generation of the active canvas depending on the GLO. There are instances, however, when applying an operation to more than only a single canvas or generation is useful. For example, in the DOSA GLOv2 seed technique in Figure 47, one might need to color the node glyphs in both canvases by the same property. In these cases, either or both of two optional parameters can be passed to adjust the GLO's scope: the **all-generations flag** and **all-canvases flag**. The all-generations flag applies the operation to every generation (node or edge, as appropriate) in the active canvas(es). The all-canvases flag applies the operation to the active generation in every canvas in the GLO Display. Both flags together apply the operation to every generation (node or edge, as appropriate) in every canvas.

As I discussed in the previous chapter, sometimes a generation of glyphs must be considered not as a single unit, but as a collection of pair-wise distinct clusters. In the graph data, these clusters are represented as a single discrete attribute of the nodes with different values for each cluster. GLOs utilize two optional parameters to handle these cluster cases: the **within attribute** the **group-by attribute**.

The within attribute signals that the operation should consider each cluster distinctly, but utilize the entire canvas's coordinate space. For example, Figure 48 shows the difference between evenly distributing nodes on x without and with a within attribute. Without the attribute, the distribution is performed amongst all of the glyphs. With the attribute, the distribution is performed independently for each cluster (here represented by glyph color).

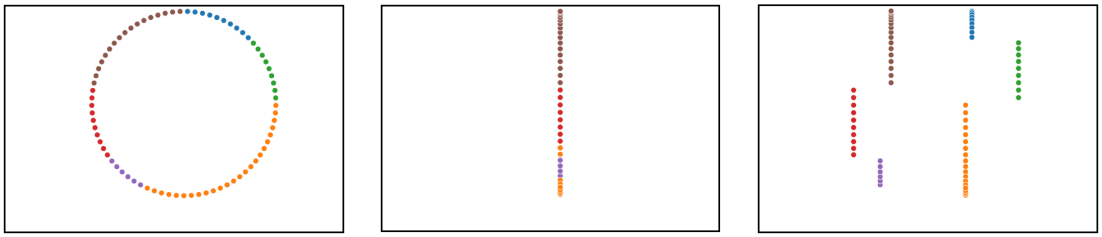


(a) without within attribute (b) with within attribute

Figure 48: *Evenly distribute nodes on* $\{x\}$ with and without a within attribute.

While the within attribute enables positioning glyphs using the canvas's coordinate space, a group-by attribute enables positioning glyphs using sub-coordinate spaces of the canvas. The coordinate space for each cluster is defined as the bounding-box of the the clusters' glyphs' current positions. Figure 49a displays the nodes positioned in a circle, sorted by cluster (again represented by glyph color). In Figure 49b, the *align nodes* {center} GLO is applied without an optional group-by attribute. Note that the nodes are all aligned to the center of the canvas. In Figure 49c, the same GLO is applied, but with the cluster attribute passed as a group-by parameter. The glyphs are each aligned to the center of the bounding box of each cluster in Figure 49a.

As I explained in the previous chapter, the group-by parameter is overloaded to also be used to modify only intra-cluster edges. For example, Figure 50 demonstrates applying a *hide edges* GLO (to set the interaction mode of the entire active edge



(a) inital layout (b) without group-by attribute (c) with group-by attribute

Figure 49: *Align nodes* {center} with and without a group-by attribute.

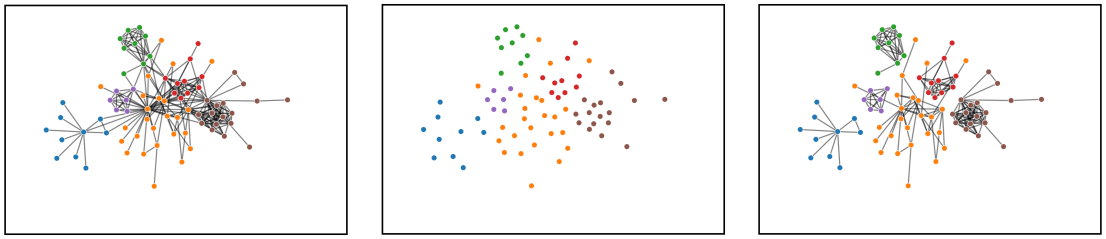


Figure 50: (a) Force-Directed Layout, (b) Force-Directed Layout after applying *hide edges*, (c) Force-Directed Layout after applying *hide edges* and *show all edges (group-by:* {cluster}*)*

generation to show none) followed by a *show all edges* GLO with a group-by attribute (to change the interaction mode of only intra-cluster edges to show all). The effect of these two operations is to show the edges whose endpoints share a cluster and hide edges between nodes in different clusters.

Note that this last example reiterates an import point: each node or edge glyph has only a single interaction mode, display mode, size, and color. Therefore, any operations that modify these values overwrite the prior value. Therefore, the order that GLOs are applied matters. The result of applying GLOs in one order may be quite different than the result of applying the same GLOs in an alternate order.

A number of GLOs refer to or depend on various pre-defined constants. For example, the GLOv2 operations to *color nodes by constant* or *align nodes* {direction}. Table 2 lists the constants that must be defined by each GLOv2 implementation. The values of the 'Canvas $*$' constants are functions of the dimensions of the canvas. An implementation may tweak these values, such as to avoid clipping scale axes with

Canvas Left	The left of the given canvas
Canvas Center	The center of the given canvas
Canvas Right	The right of the given canvas
Canvas Top	The top of the given canvas
Canvas Middle	The middle of the given canvas
Canvas Bottom	The bottom of the given canvas
Default ρ	ρ value to use when positioning by constant on ρ
Default θ	θ value to use when positioning by constant on θ
Default node size	Node size to use when sizing nodes by constant
Default edge size	Edge size to use when sizing nodes by constant
Default stack distance	Distance between nodes to use when stacking nodes evenly
Default node color	Node color to use when coloring nodes by constant
Default edge color	Edge color to use when coloring nodes by constant
Default convex hull color	Color to use when coloring convex hulls by a constant

Table 2: GLOv2 Constants

aligned nodes. (I have chosen to do this in the GLO.js implementation.) The values of the 'Default $*$' constants, however, are left to each implementation. As I discuss in the next chapter, this provides implementations with additional customization without sacrificing the important aspects of techniques.

Finally, GLOs are atomic. No operation depends on another operation having been applied in order to determine the effect of the operation. In other words, the result of applying a GLO is always well-defined. As I will discuss in the next section, this GLO independence is a strict property of GLOv2 but notably not a property of GLOv1.

4.5 Differences Between GLOv1 and GLOv2

Recall that the visual element model described above is the visual element model for GLOv2. GLOv1's visual element model is a subset of GLOv2's. The GLOv1 visual element model consists of a single canvas with any number of node generations (including generations of super-node glyphs) and a single edge generation. GLOv1 does not support changing the color of nodes, simply the size. GLOv2 extended GLOv1's visual element model to include multiple edge generations, multiple canvases, colorable node and edge glyphs, convex hulls, and a larger number of display and interaction modes for both node and edge glyphs.

With respect to the operations sets, as Table 3 shows, of the 72 GLOv2 GLOs, 22

are equivalent to the 34 GLOv1 operations. The reason for this reduction is four-fold.

First, GLOv2 includes optional parameters. In GLOv1, *evenly distribute nodes on x or y* and *evenly distribute nodes on x or y by* {attribute} are considered two distinct operations. In GLOv2, these are both covered by *evenly distribute nodes on* {axis} GLO, which can take an optional sort-by parameter and/or an optional invert parameter.

Second, GLOv1 differentiates between operations for positioning by categorical and continuous attributes. *Substrate nodes on x or y by* {categorical attribute} and *position nodes on x or y relatively by* {continuous attribute} are two distinct operations. In GLOv2, these are simply both covered by *position nodes on* {axis} *by* {attribute}.

Third, GLOv1's has distinct polar coordinate operations. In fact, GLOv1 has separate operations for radial operations and angular operations. In contrast, GLOv2 treats the four axes (x,y,ρ,θ) equivalently with a single set of operations, with the axis a mandatory parameter of any relevant operation.

Fourth, GLOv1 does not support multiple parameters. As I mentioned above, the GLOv2 operations *aggregate nodes by* {discrete attributes} *using* {method} and *aggregate edges by* {discrete attributes} *using* {method} GLOs allow for multiple discrete attribute parameters to be passed. Under GLOv1, aggregating by a single attribute or two attributes are two distinct operations. Furthermore, under the GLOv1 model nodes cannot be aggregated by three or more attributes.

Note that in Table 3 the GLOv1 GLO *display links as circles* is marked as equivalent to the GLOv2 GLO *display edges as squares*. Under GLOv1, the edges in matrix displays were shown as circles (e.g. see Figure 51). GLOv2 replaced these circles with squares, as that is how they are traditionally represented.

There are two more critical differences between the two operation sets that I

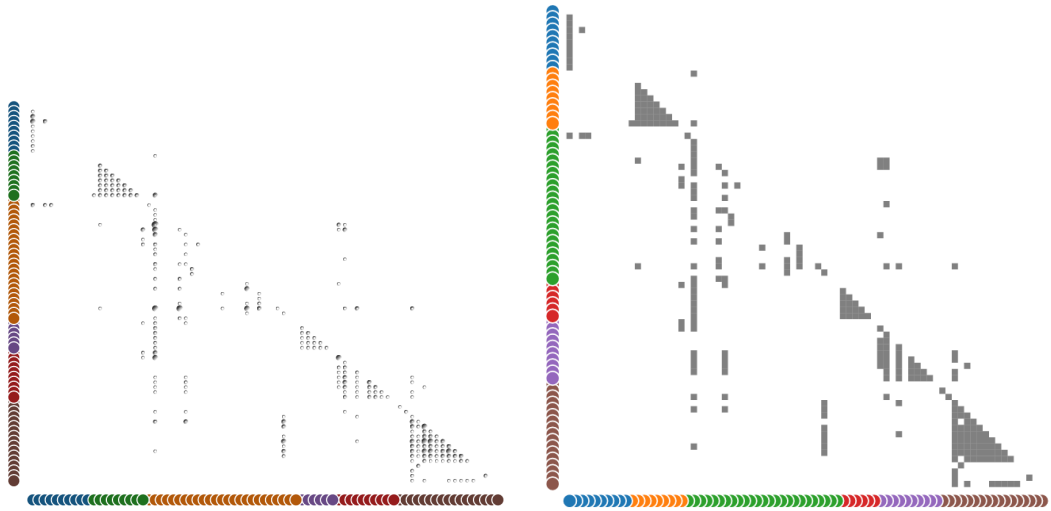


Figure 51: (a) GLOv1 Adjacency Matrix seed technique with circles for edges and (b) equivalent Adjacency Matrix in GLOv2 with squares for edges.

wish to highlight. The first critical difference relates to the GLOv1 operations *substrate nodes on x or y by* {categorical attribute} and *evenly distribute nodes within substrates*. Under GLOv2, GLOs do not have any dependencies on other GLOs—each GLO is well-defined regardless of the current state of the GLO Display. Under GLOv1, the *evenly distribute nodes within substrates* operation depends upon the *substrate nodes* operation being called first and setting a global 'substrate' variable. GLOv2 replaces this global variable with the within optional parameter. Without this dependency, GLOv2's operations are fully independent of each other.

The second critical difference is replacing the GLOv1 operation *apply* {algorithm} *to the nodes* with the GLOv2 operation *apply force-directed algorithm to nodes*. In Chapter 5, I describe how a distinct force-directed algorithm GLO allows for a more precise comparison between techniques than a catch-all algorithm GLO.

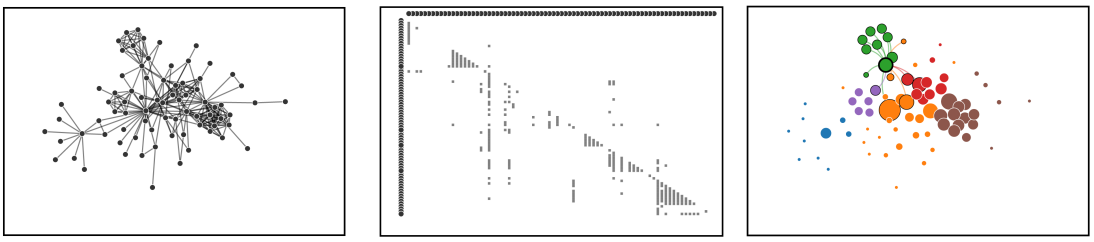
The remaining 50 of the 72 GLOv2 operations are for either manipulating the extended visual model (e.g. *clone edges* and *select canvas* {num}) or for providing additional expressiveness. As mentioned above, GLOv2 includes a larger variety of node and edge glyph display modes and interaction modes.

GLOv1	GLOv2 Equivalent
align nodes {*left, center, right, top, middle, bottom*}	align nodes {*dir*}
evenly distribute nodes on x or y by {*attribute*}	evenly distribute nodes on {*axis*} (by {*attr*})
evenly distribute nodes on x or y	evenly distribute nodes on {*axis*} (by {*attr*})
substrate nodes on x or y by {*cat. attribute*}	position nodes on {*axis*} by {*attr*}
evenly distribute nodes within substrates	evenly distribute nodes on {*axis*} (by {*attr*})
position nodes on x or y relatively by {*cont. attribute*}	position nodes on {*axis*} by {*attr*}
evenly distribute nodes radially by {*attribute*}	evenly distribute nodes on {*axis*} (by {*attr*})
evenly distribute nodes radially	evenly distribute nodes on {*axis*} (by {*attr*})
position nodes radially by {*cont. attribute*}	position nodes on {*axis*} by {*attr*}
substrate nodes radially by {*cat. attribute*}	position nodes on {*axis*} by {*attr*}
evenly distribute nodes along plot radius by {*attribute*}	evenly distribute nodes on {*axis*} (by {*attr*})
evenly distribute nodes along plot radius	evenly distribute nodes on {*axis*} (by {*attr*})
position nodes along plot radius by {*cont. attribute*}	position nodes on {*axis*} by {*attr*}
substrate nodes along plot radius by {*cat. attribute*}	position nodes on {*axis*} by {*attr*}
position nodes along plot radius by constant	position nodes on {*axis*} by constant
apply {*algorithm*} to the nodes	apply force-directed algorithm to nodes
size nodes by constant	size nodes by constant
size nodes relatively by {*cont. attribute*}	size nodes by {*attr*}
display all links	show all edges
display selected links	show incident edges
hide links	hide edges
display links as straight	display edges as straight lines
display links as curved	display edges as curved lines
display links as circles	display edges as squares
clone active generation	clone nodes
select generation k	select node generation {*num*}
set source generation k	set source generation {*num*}
set target generation k	set target generation {*num*}
remove generation k	remove node generation {*num*}
aggregate by {*cat. attribute*}	aggregate nodes by {*discrete*} using {*method*}
aggregate by {*cat. attribute*} and {*cat. attribute*}	aggregate nodes by {*discrete*} using {*method*}
deaggregate generation k	deaggregate nodes
show x or y axis	show {*axis*} axis
hide x or y axis	hide {*axis*} axis

Table 3: GLOv2 operations equivalent to GLOv1 operations.

4.6 Specifying Techniques Using GLOs

Since each GLO is a step of a transition, one should be able to describe a technique by the GLOs necessary to transition to it. However, what is the source state of such a transition? Transitioning to a technique from different source states might lead to drastically different definitions. If the initial state was, for example, a Force-Directed Layout in Figure 52a, then the definition of the Edgemap A technique in Figure 52c would not include the *apply force-directed algorithm to nodes* GLO since the nodes would already be in those positions. However, if the initial state was a Matrix Plot technique in Figure 52b, then the definition for the Edgemap A technique would have to include either the *remove all cloned nodes* GLO or the *remove node generation* {num} GLO.



(a) Force-Directed Layout (b) Matrix Plot (c) Edgemap A

Figure 52: Force-Directed Layout, Matrix Plot, and EdgeMap A techniques rendered in GLO.js.

In order to avoid this inconsistency, a state can be defined such that transitions from the state to a given technique are consistent. I call this state the **null state**. The null state is defined as a GLO Display with the following properties:

- a single unpartitioned canvas,
- a single generation of unaggregated, unrotated node glyphs in the 'no interaction' mode,
- a single generation of unaggregated edge glyphs,
- no edge waypoints (implied by only a single generation of edge glyphs),
- no convex hulls drawn,
- no axes drawn,
- and no meta-axes drawn.

Defining the null state in this way has two notable advantages.

First, this state avoids the need for any **inverse GLOs**:

- *remove canvas num*
- *remove all partitions*
- *deaggregate nodes*
- *deaggregate edges*
- *unrotate nodes*
- *remove all cloned nodes*

- *remove all cloned edges*
- *remove node generation num*
- *remove edge generation num*
- *hide axis axis*
- *hide meta axis axis*
- *hide convex hulls*
- *remove all edge waypoints*
- *stop highlight neighbors*

Appearing in many transition matrix entries, inverse GLOs more often than not communicate information about the source technique rather than helping describe the target technique. For example, the *remove all cloned nodes* operation of a (Matrix Plot, Edgemap A) transition mostly communicates that the Matrix Plot has multiple generations.

Second, this null state is highly under-constrained. The positions, display modes, sizes, and colors of glyphs are not defined. The interaction mode of the edges is also undefined. This mandates that GLOs for setting these properties be included in a technique's definition.

Using this null state, one can define a technique by the ordered list of GLOs necessary to transition from a null state to the technique. This list is a **GLO specification** of the technique. For example, here is the GLO specification for the EdgeMap A seed technique:

- display nodes as circles
- size nodes by {*node_size_attr*}
- color nodes by {*node_color_attr*}
- display edges as curved lines
- size edges by constant
- color edges by {*source.node_color_attr*}

GLO	Inverse GLO(s)
partition canvas on {axis} *(by* {num}*)*	*remove all partitions*, *remove all cloned nodes*, *remove all cloned edges*
filter partition canvas on {axis} *by* {discrete}	*remove all partitions*, *remove all cloned nodes*, *remove all cloned edges*
show meta {axis} *axis*	*hide meta axis axis*
clone nodes	*remove all cloned nodes*
clone edges	*remove all cloned edges*
set edge waypoint edge generation {num}	*remove all edge waypoints*
aggregate edges by {discrete} *using* {method}	*deaggregate edges*
aggregate nodes by {discrete} *using* {method}	*deaggregate nodes*
highlight in-out neighbors	*stop highlight neighbors*
highlight neighbors	*stop highlight neighbors*
rotate nodes {num} *degrees*	*unrotate nodes*
show convex hulls	*hide convex hulls*
show {axis} *axis*	*hide axis axis*

Table 4: Inverse GLOs required for GLOv2 GLOs. For each GLO in the first column that the technique specification contains, the corresponding inverse GLO(s) in the second column must be applied to return to the null state.

- show in-out edges
- highlight neighbors
- apply force-directed algorithm to nodes

Note that this specification includes variables (node_size_attr and node_color_attr) passed as parameters to the operations. GLO specifications are effectively functions that call GLOs and therefore can take their own parameters.

Using a GLO specification, one can easily determine the inverse GLOs necessary to return to the null state from the technique. In order to return to the null state from a given technique, for each GLO on the left-side of Table 4 that appears in the specification, one must apply the corresponding inverse GLO(s) from the right-side of the table. Transitioning from any source technique to any target technique can be expressed as first transitioning to the null state and then transitioning to the target technique. (Though note that this may not be the most efficient transition.)

In Appendix A and B, I provide GLOv1 and GLOv2 specifications for each of the GLOv1 and GLOv2 seed techniques, respectively. Using the GLOv2 specifications, I calculated the usage of each of the 72 GLOv2 operations. Table 5 lists the 55

24 size edges by constant
21 size nodes by constant
18 color nodes by constant
17 display nodes as circles
16 show all edges
14 display edges as curved lines
13 color edges by constant
13 evenly distribute nodes on {*axis*}
12 color edges by {*attr*}
12 position nodes on {*axis*} by {*attr*}
11 align nodes {*dir*}
11 color nodes by {*attr*}
9 display edges as straight lines
9 display nodes as {*attr*} labels
9 size nodes by {*attr*}
8 clone nodes
8 set target generation {*num*}
7 show {*axis*} axis
6 Position edges by {*attr*},{*attr*}
6 display edges as squares
6 rotate nodes {*deg*}
5 partition canvas on {*axis*}
4 aggregate edges by {*attrs*} using {*method*}
4 aggregate nodes by {*attrs*} using {*method*}
4 apply force-directed algorithm to nodes
4 clone edges
4 display nodes as squares
4 hide edges
4 position nodes by constant on {*axis*}
4 select canvas {*num*}
4 size edges by {*attr*}
3 highlight neighbors
3 show faded and incident edges
2 color edges by {*attr*},{*attr*}
2 display nodes as bars
2 filter partition canvas on {*axis*} by {*attr*}
2 position nodes evenly stacked attr
2 select row {*num*}
2 show in-out edges
2 show incident edges
1 align edges attr
1 color convex hulls by attr
1 display edges as {*attr*} labels
1 display edges as bars
1 display edges as right angles
1 evenly distribute edges on {*axis*}
1 highlight in-out neighbors
1 select column {*num*}
1 select edge generation {*num*}
1 select node generation {*num*}
1 set edge waypoint generation {*num*}
1 set source generation {*num*}
1 show convex hulls
1 show edges as faded
1 show meta {*axis*} axis

Table 5: Number of GLOv2 seed technique specifications (out of 29) containing each GLOv2 operation.

operations that are used in the specifications, along with the number of specifications in which each occurs. Table 6 lists the remaining 17 operations that do not appear in any of the 29 seed technique specifications. Each of these is either an inverse GLO or a GLO added during the augmentation stage of the induction method.

position nodes stacked on {*axis*} by {*attr*}
color convex hulls by constant
set {*axis*} axis node generation {*num*}
unrotate nodes
remove all edge waypoints
hide convex hulls
stop highlight neighbors
deaggregate nodes
deaggregate edges
remove node generation {*num*}
remove edge generation {*num*}
remove all cloned nodes
remove all cloned edges
remove canvas {*num*}
remove all partitions
hide {*axis*} axis
hide meta {*axis*} axis

Table 6: GLOv2 operations that do not appear in any GLOv2 seed technique specifications.

CHAPTER V

UTILITY OF GRAPH-LEVEL OPERATIONS

In the prior chapters, I described how to induce a model of graph visualization from a set of graph visualization seed techniques, presented two models induced using the method (GLOv1 and GLOv2), and defined how to describe techniques using those models. In this chapter, I reflect on how these models can positively affect three different common visualization tasks: simplifying graph visualization engineering, understanding the design space of graph visualization techniques, and identifying novel graph visualization techniques.

5.1 Easing the Engineering Challenge

In today's environment, developers can take advantage of the power and breadth of graph visualization and incorporate its tools and methods into their applications. However, such development hinges on: non-portability; detailed knowledge of low-level graphics technologies such as SVG, WebGL, CoreGraphics, and Swing; and repetitive "boilerplate" code to get even simple visualization elements onto the screen. This is striking, since abstractions have always been at the core of much of computer science. While one can still write the machine or assembly code to write a file to a hard drive, such low-level programming is no longer necessary with the advent of high-level programming languages that abstract away the details of hardware access. Graph-level operations provide an equivalent abstraction layer within the visualization software stack (see Figure 53).

At the base of the stack is a host language and a graphics library. Recently, Javacript/SVG (or Javascript/Canvas or Javascript/WebGL) has been a common

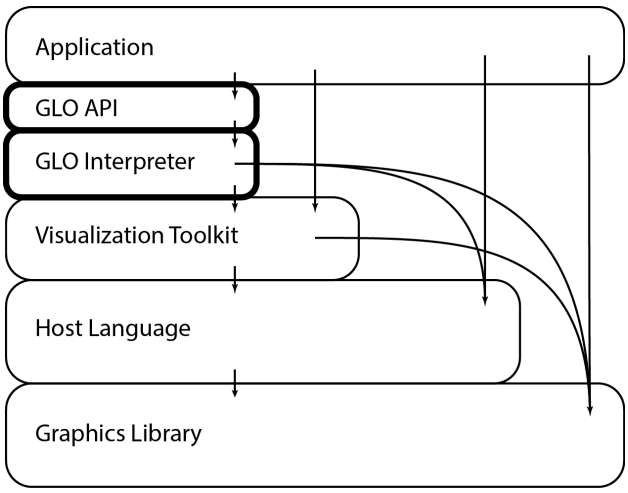


Figure 53: Visualization software stack

choice for these layers, replacing Java/Swing. The next layer of the stack is the visualization toolkit. These include the popular D3.js [41] toolkit for Javascript and prefuse [113] toolkit for Java. These toolkits are designed to abstract away some of the graphics code necessary to render visualizations while still providing maximum expressiveness. In exchange for the improved expressiveness, however, these toolkits often require a large amount of toolkit-specific "boilerplate" code to set up the environment. D3 requires far less than prefuse does, but it is still non-trivial. These toolkits also tend to be heavily tied to the host language and graphics system, requiring a developer to have a strong knowledge of both. For example, writing code using D3 requires a strong knowledge of Javascript, how the Document Object Model (DOM) is constructed, and about the underlying graphics system (such as SVG) and the system's elements.

At the top of the stack are user-facing applications. Examples of these might be a standalone interactive visualization of a network or a graph visualization design application such as Gephi [28]. As it stands, these applications must be built either directly on the host language or on a combination of the host language and the visualization toolkit. In order for developers to implement a given visualization technique within that application, they would need to either implement it from scratch or be

lucky enough to find an implementation using their chosen language and graphics model. They would also need to do this for each visualization technique that they wish to support.

A GLO model provides a buffer layer between the toolkit and the application. Since GLO specifications of techniques are portable (i.e. they are not tied to a specific software environment), one simply needs a way to convert a GLO specification into a visualization using the host language and graphics library. This conversion of GLO specifications into displays is performed by a **GLO interpreter**. A GLO interpreter takes as input a GLO specification of a technique and a graph dataset and outputs a visualization of the dataset using the technique.

In the previous chapter, I alluded to the fact that considering the operations set of a GLO model as an API is powerful. A GLO interpreter simply implements the API defined by the model's operations set. Once an interpreter for the appropriate language-graphics pairing exists, a developer can take advantage of any techniques already specified using GLOs rather than implementing each technique from scratch.

In the cases where an application only requires the use of specific visualization techniques, a developer can use an interpreter and GLO specifications to easily integrate pre-defined techniques into applications. In other cases, developers can pass along the power of GLOs to analysts (i.e. users of these applications) to allow them to explore their data or communicate their findings by customizing techniques (see Section 5.3). The GLO-STIX application described in Section 5.1.2 is an example of this latter case.

Finally, the GLO interpreter model provides developers with increased **superficial expressiveness**. More specifically, how each GLO interpreter implements a set of GLOs can be different. To start, each implementation must define the various constants that GLOs reference and an interpreter's developer could chose colors and typefaces that fit with a branding strategy. Another developer might build an

interpreter that renders all elements using "sketch-like" graphics [255] for showing uncertainty or encouraging others to feel comfortable critiquing visualizations. One could imagine an implementation where the elements wiggle in place like gelatin or look as if they are floating on water. In this sense, the specification-interpreter model functions like the HTML-CSS model of separating the look-and-feel from the underlying structure of the techniques. In all of these cases (assuming that the GLO API is implemented as it is defined) the visualizations that are produced by these custom implementations still retain the critical aspects of each of the input visualization techniques.

5.1.1 Implementations

In order to demonstrate the feasibility of implementing a GLO interpreter, I have built two GLO interpreters, one for GLOv1 and one for GLOv2. Both interpreters are written in Javascript and use Scalable Vector Graphics (SVG) [240] for graphics. Both interpreters utilize the D3.js library [41] for managing the SVG elements. The GLOv1 implementation covers the full operations set of GLOv1. However, the architecture of the implementation proved difficult to extend to include the larger GLOv2 visual element model and operations set. Therefore, the GLOv2 implementation is a complete rebuild, dubbed GLO.js.

To use the GLO.js implementation, a developer provides a node list, an edge list, a set of type descriptions (discrete or continuous) of the node and edge attributes, and an SVG element in the DOM where the resulting visualizations should appear. The developer can then apply pre-set techniques stored as functions to the data or apply GLOs one at a time using the GLO.js API (an implementation of the GLOv2 API). Information regarding node and edge glyph properties are stored on the backing nodes and edges, enabling the developer to easily access any SVG properties if he or she wishes to customize the glyphs beyond the scope of GLOs.

At the time of this dissertation, GLO.js supports 38 of the 72 GLOv2 GLOs, including the 22 GLOv2 operations necessary to fully support GLOv1 definitions. With its implemented operations, GLO.js can render 14 (and closely approximate 9 more, for a total of 23) of the 29 GLOv2 seed techniques (see figures in Appendix B). The remaining GLOs include inverse GLOs, GLOs with very similar functionality to implemented GLOs (e.g., displaying nodes as circles is implemented while displaying nodes as circles is not), and GLOs used only by one or two seed techniques (e.g. filter-partitioning canvases). GLO.js supports all of GLOv2's optional parameters: group-by, within, sort-by, invert, all-canvases, and all-generations. Furthermore, the GLO.js architecture is designed to support the full GLOv2 visual element model and operation set.

GLO.js currently consists of approximately 4000 lines of code (including line breaks and comments). This does not include the technique implementations, which each requires the number of lines as its specification in Appendix B. The code was written by a single developer (myself) over the course of approximately two months.

The GLO.js project is open-source and available on Github at `http://github.com/chadstolper/glo`. Open-sourcing the project enables both for collaborative future development as well as enables developers to easily access the GLO API in order to port GLOs to other popular software environments (such as R or python) in order to further increase their the utility of the GLO models.

5.1.2 GLO-STIX

GLO-STIX (**G**raph-**L**evel **O**perations for **S**pecifying **T**echniques and **I**nteractive e**X**ploration) is a prototype application for exploring graphs using GLOs. A team of undergraduate and graduate student researchers assisted me in designing and building the application atop my GLOv1 Javascript implementation.

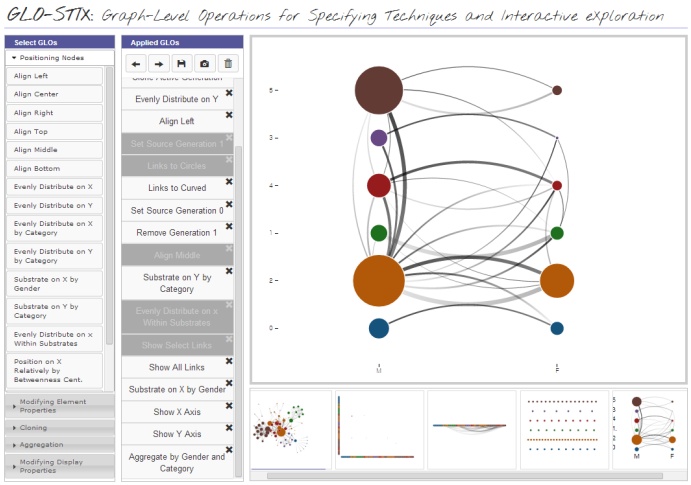


Figure 54: The GLO-STIX interface.

Our goal in designing the prototype focused on enabling a graph analyst to interactively explore a newly encountered graph dataset. We envisioned an analyst, upon first receiving a dataset, wishing to better understand the graph's features. The tool was designed to enable the analyst to apply individual GLOs with the possibility of identifying interesting views of the data to save for future use. In addition, developing the GLO-STIX prototype provided a testbed for evaluating the viability of GLOs and the GLOv1 model.

We generated a number of requirements for the prototype application that would become GLO-STIX:

- The prototype should implement the full set of GLOv1 operations.
- As our intent was to enable an analyst to explore their network data more effectively using GLOs, the prototype should enable an analyst to apply individual GLOs to a graph.
- The analyst should be able to experiment with applying various GLOs and therefore the prototype should enable an analyst to move backwards and forwards through the GLO history.

- If an analyst has identified an effective display of the network, he or she may wish to know the GLOs necessary to recreate the display, as opposed to the full path he or she took to reach the display. Therefore the prototype should suggest to the analyst which GLOs in the history might no longer be relevant to the current visualization due to more recently applied GLOs.

- An analyst should be able to easily recall techniques that he or she found interesting as well as be able to easily compare them side-by-side and switch between them seamlessly. Therefore the prototype should allow an analyst to save an image (snapshot) of the current visualization along with its GLO history to compare techniques.

A number of these requirements concern the analyst seeing both how he or she reached the current display and saving interesting displays for future analysis. These were influenced by the work on visualization provenance such as VisTrails [51] and Graphical Histories [109].

We began the development of the user interface by translating the requirements listed above into necessary software functions and user interface (UI) elements. We settled on four UI elements: a list of all available GLOs, a history view of applied GLOs, the visualization display, and a region for displaying the snapshotted techniques. The functions we identified included the GLOs themselves, support for unapplying and re-applying a GLO to a graph, and snapshotting the current configuration of GLOs.

Using these elements and functions we sketched a number of designs for the user interface. We discussed these drawings amongst the team, identifying potential advantages and disadvantages of each. We eventually settled on the interface in Figure 54. This interface features all of the basic elements (available GLOs, view of the history, visualization area, view of visualization states captured) and the functions envisioned. On the left, in the *Select GLOs* panel, are the GLOv1 operations grouped by category.

These can be dragged and dropped into the *Applied GLOs* panel to apply them to the GLO Display in the center. GLOs can be removed by pressing the X on each GLO in the *Applied GLOs* panel. The interface attempts to identify overwritten GLOs through tables of which GLOs manipulate the same node or edge glyph properties.[1] Finally, the analyst can use the Camera button to save the current technique to the snapshot list at the bottom of the interface. Clicking on one of these techniques restores it to the GLO Display.

We implemented GLO-STIX as a browser-based application in JavaScript using D3.js [41], jQuery[2], Bootstrap[3], and jQueryUI[4]. The code for the application is open-sourced as part of the GLO.js project (`http://github.com/chadstolper/glo`).

5.1.3 GLO-CLI

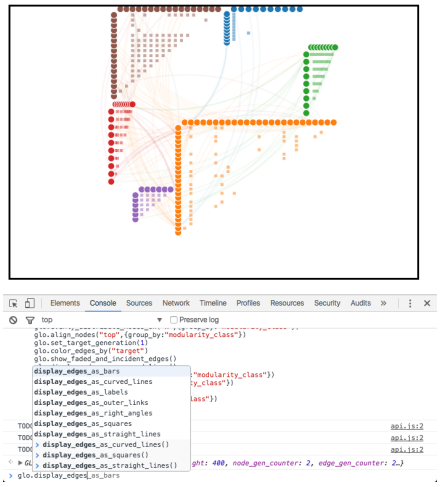


Figure 55: The GLO-CLI interface.

While GLO-STIX is a web-based application for manipulating a GLO Display using drag-and-drop elements, GLO-CLI is a web-based application for manipulating

[1]Since GLOv1 does not support the group-by flags supported by GLOv2, this method works reasonably well. The algorithm does has issues with cloned generations, since it considers the applied operations as a single, non-branching list.

[2]http://jquery.com

[3]http://getbootstrap.com

[4]http://jqueryui.com

a GLO Display using the browser's console (i.e., a command-line). The application loads a graph dataset, sets up a GLO Display, and populates the display in a null state. An analyst can then apply GLOs to the GLO Display using GLOv2 operations defined in the GLO.js API. (GLOv2 operations that have not yet been implemented in GLO.js fail gracefully by outputting a message that the operation has not yet been implemented or by outputting a message and applying a closely related operation.) As shown in Figure 55, the application makes use of modern browsers' advanced consoles to provide auto-complete and enable the analyst to inspect underlying GLO.js variables and resulting SVG elements.

The GLO-CLI tool has proved useful both for debugging the GLO.js interpreter as well as for identifying the novel visualization techniques described in Section 5.3.

5.2 Enabling A Deeper Understanding of Techniques

One of the most important aspects of visualization research is identifying techniques that are effective for various analytical tasks. One means to do this is to identify which techniques are similar to techniques that are known to be effective for certain tasks. However, determining the similarity between techniques is not trivial.

There is a body of visualization research on more formally understanding the design space of visualization techniques and then comparing techniques' distances within that space. Attempts at accomplishing this often fall into one of two strategies. One strategy defines a feature space over techniques then performs analysis on techniques using vector representations within that feature space. Another strategy defines a distance metric between techniques and performs analysis on the graph formed using the distance metric. In the following subsections, I show how graph-level operations provide an elegant means to identify both feature spaces and distance metrics and briefly demonstrate how these can be used to better understand the design space of graph visualization.

5.2.1 Feature Space Analysis

One of the primary challenges of feature space analysis is identifying a meaningful set of features to represent each technique instance. Because GLO specifications represent techniques as lists of operations, they suggest a number of elegant and useful feature sets that can be described as vectorization methods.

As a first vectorization method, one can consider graph-level operations as binary features. For a given technique, the value of each GLO's feature is a 1 if the technique's specification includes the operation and 0 otherwise. In this way, any technique can be easily represented as a vector in $\{0,1\}^{72}$. (These vectorization methods are why GLOv2's specific *apply force-directed algorithm* GLO is preferred over GLOv1's catch-all *apply algorithm* GLO.)

The optional parameters built into GLOv2 (sort-by, invert, group-by, within, all-canvases, all-generations) can be used to controllably and predictably increase this feature space. Representations resulting from calling operations with group-by and within parameters, especially, can significantly differ from representations resulting from the operation without the parameters. One could incorporate these optional parameters into the feature space using one of two methods.[5]

For the second vectorization method, rather than each operation being a feature, instead there are four features for each operation: the operation with no group-by or within attributes, the operation with a group-by attribute but no within attribute, the operation with a within attribute but no group-by attribute, and the operation with both group-by and within attributes. Once again, a technique's value for each feature is 1 if the technique's specification contains the relevant operation and parameter combination and 0 otherwise. In this space, a technique is represented as a vector in $\{0,1\}^{288}$. The first four elements of the vector correspond to the first operation, the

[5] I have focused on the group-by and within parameters, but these methods would work equivalently for the sort-by, invert, all-canvases and all-generations parameters as well.

next four to the second operation, and so on for each of the 72 operations.

For the third vectorization method, each operation remains a single feature ala the first vectorization method. However, two additional features are added to the vector, one for the group-by optional parameter and one for the within optional parameter. If the technique's specification includes any group-by parameters, then the first of these features has a value of 1 (otherwise a value of 0) and equivalently for the within parameter and the second extra feature. In this space, a technique is represented as a vector in $\{0, 1\}^{74}$.

Using the GLOv2 specifications in Appendix B, I have applied the three vectorization methods (which for brevity I will refer to as no-flags, flags, and flags-xtra, respectively) to generate three vector representations for each of the 29 GLOv2 seed techniques.

I then used the hierarchical clustering function included with the SciPy package [135] to cluster the techniques. The hierarchical clustering function takes two parameters: a distance metric and a cluster-comparison method.

I chose three distance metrics suitable for binary feature vectors: Hamming distance [107], Jaccard distance [151], and cosine distance [206]. The Hamming distance is the number of positions in which two vectors have differing values. The Jaccard distance takes the ratio of positions where the two vectors are 1 and agree over to the positions where the two vectors are 1 and either agree or disagree and then subtracts this value from 1. The cosine distance between two vectors is the cosine of the angle formed by the two vectors.

I chose four cluster-comparison methods included with SciPy suitable for these distance metrics: nearest-point, farthest-point, average, and weighted.[6] The nearest-point algorithm (or *single* method) considers the distance between two clusters to

[6] `http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html`

Method	Metric	Cophenetic Correlation
average	hamming	0.832799904046
weighted	hamming	0.821886497865
weighted	jaccard	0.755900447652
complete	hamming	0.726395383264
average	cosine	0.703449587146
average	jaccard	0.706271040351
complete	jaccard	0.68661661651
weighted	cosine	0.693924001138
complete	cosine	0.676424420188
single	hamming	0.592861499268
single	jaccard	0.468258988449
single	cosine	0.425621644926

Table 7: Results of hierarchically clustering technique vectors created by ignoring optional parameters (no-flags).

be the minimum distance between any point in the first cluster to any point in the second cluster. The farthest-point algorithm (or *complete* method) [239] considers the distance between two clusters to be the maximum distance between any point in the first cluster to any point in the second cluster. The *average* method [210] considers the distance between two clusters to be the mean over all distances between pairs of nodes where one node is in the first cluster and the second node is in the second cluster. Finally the *weighted* method [210] considers the distance between a cluster and a second cluster to be the average of the distance between the two clusters that make up the first cluster to the second using the average method. (In other words, the method weights larger contributing sub-clusters higher than lower-contributing sub-clusters).

I therefore generated 36 total hierarchical clusterings (3 vectorization functions x 3 distance metrics x 4 cluster comparison methods). For each clustering, I used SciPy to calculate the cophenetic correlation coefficient [211] which is a measure from 0-1 of how faithfully a hierarchical clustering preserves pair-wise distances between items. Tables 7, 8, 9 report the results for the no-flags, flags, and flags-xtra vectorization methods, respectively. For all three vectorization methods, Hamming distance with the average clustering method provided the highest cophenetic clustering correlation.

Using Matplotlib [129], I rendered the resulting dendrograms generated by the

Method	Metric	Cophenetic Correlation
average	hamming	0.832799904046
weighted	hamming	0.821886497865
weighted	jaccard	0.755900447652
complete	hamming	0.726395383264
average	cosine	0.703449587146
average	jaccard	0.706271040351
complete	jaccard	0.68661661651
weighted	cosine	0.693924001138
complete	cosine	0.676424420188
single	hamming	0.592861499268
single	jaccard	0.468258988449
single	cosine	0.425621644926

Table 8: Results of hierarchically clustering technique vectors with optional parameters (flags).

Method	Metric	Cophenetic Correlation
average	hamming	0.832799904046
weighted	hamming	0.821886497865
weighted	jaccard	0.755900447652
complete	hamming	0.726395383264
average	cosine	0.703449587146
average	jaccard	0.706271040351
complete	jaccard	0.68661661651
weighted	cosine	0.693924001138
complete	cosine	0.676424420188
single	hamming	0.592861499268
single	jaccard	0.468258988449
single	cosine	0.425621644926

Table 9: Results of hierarchically clustering technique vectors created by adding features for optional parameters (flags-xtra).

hierarchical clusters. Figure 56 consists of the three dendrograms created using the Hamming distance and average method.[7] There are a number of interesting features of these three clusterings.

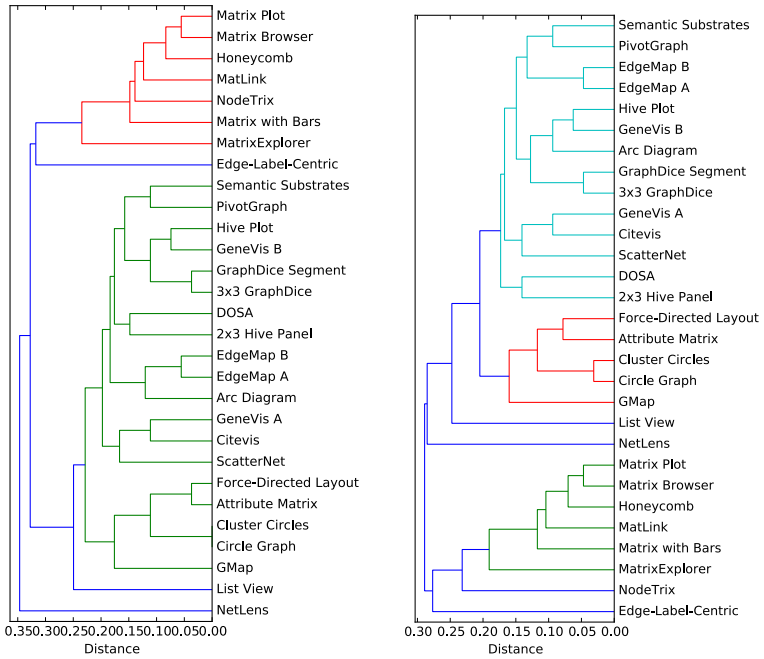
First, note that all three clusterings cluster the matrix-based seed techniques (Matrix Plot, Matrix Browser, Honeycomb, MatLink, NodeTrix, Matrix with Bars, and MatrixExplorer) together. (The cluster red cluster in the no-flags clustering, green cluster in the flags clustering, and red cluster in the flags-xtra clustering.) This aligns with the expectation that these techniques are similar to each other and different from the other techniques.

Second, within these matrix-technique clusters, note how the NodeTrix technique (Figure 57) moves between the three clustering techniques. As the only technique in the matrix-based cluster that utilizes the group-by parameter, it makes sense that the technique would be less closely aligned with the other techniques using the flags vectorization method. What is encouraging is that even though it is the last member of the cluster to be included in the flags case, it is still considered nearer to the matrix-based techniques than to any other technique.

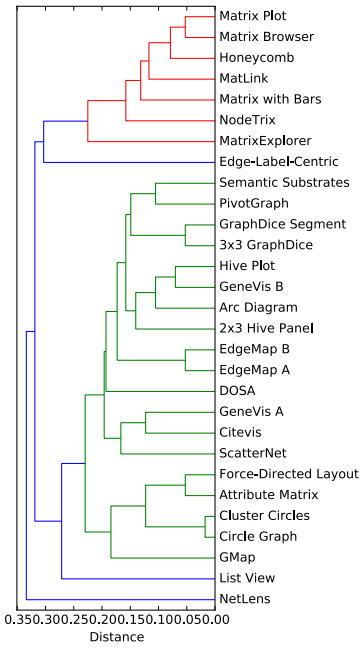
Third, consider the Cluster Circles and Circle Graph techniques. Specifications for these techniques consist of the same operations. The Cluster Circles technique includes duplicate operations, with the second set using the group-by parameter. When optional parameters are ignored (no-flags) the two techniques have identical vector representations. When optional parameters are taken into account (flags, flags-xtra) the two techniques are still considered very similar.

Fourth, the Semantic Substrates, PivotGraph, HivePlot, GeneVis A, GeneVis B, 3x3 GraphDice, GraphDice Segment, CiteVis, Scatternet, DOSA, and 2x3 Hive Panel seed techniques all concern positioning nodes based on attributes of those nodes. Thus, it is encouraging that they are all clustered together in all three clusterings.

[7]For completeness, I include all 36 dendrograms in Appendix F.



(a) no-flags (b) flags



(c) flags-xtra

Figure 56: Dendrogram results for three hierarchical clustering using three vectorization methods, Hamming distance, and average cluster comparison rendered using Matplotlib [129].

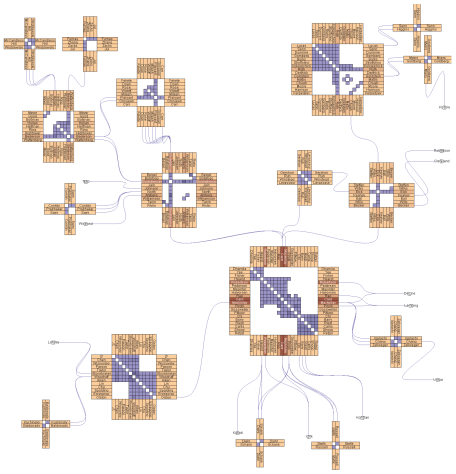


Figure 57: GLOv2 NodeTrix seed technique from [117].

Interestingly, the EdgeMap A, EdgeMap B, and Arc Diagram seed techniques also are clustered along with these.

Fifth, notice how Force Directed Layouts and Attribute Matrices (which consist of small multiples of force-directed layouts), and GMaps (force-directed layout with convex hulls) are clustered together in all three clusterings. In each of these seed techniques, there is no node interaction, no edge interaction, and all of the node and edge glyphs are colored and sized by a constant. Notably, the EdgeMap A seed technique is not clustered with these techniques. While the node glyphs in the EdgeMap A technique are also positioned using a force-directed layout, they are sized and colored by attributes, and have non-static interaction modes (as do its edges).

The EdgeMap A/Force-Directed Layout case might suggest weighting different features (such as operations that affect node glyph position) more than other features (such as operations that size node and edge glyphs). In addition, one could consider non-binary feature vectors in order to encode specifications that call the same operation more than once (such as the Circle Plot/ Circle Clusters case). While I have studied the unweighted, binary feature vector case, both weighted and non-binary feature vectors illustrate the breadth of interesting future research directions enabled by GLOs.

5.2.2 GLO Distance

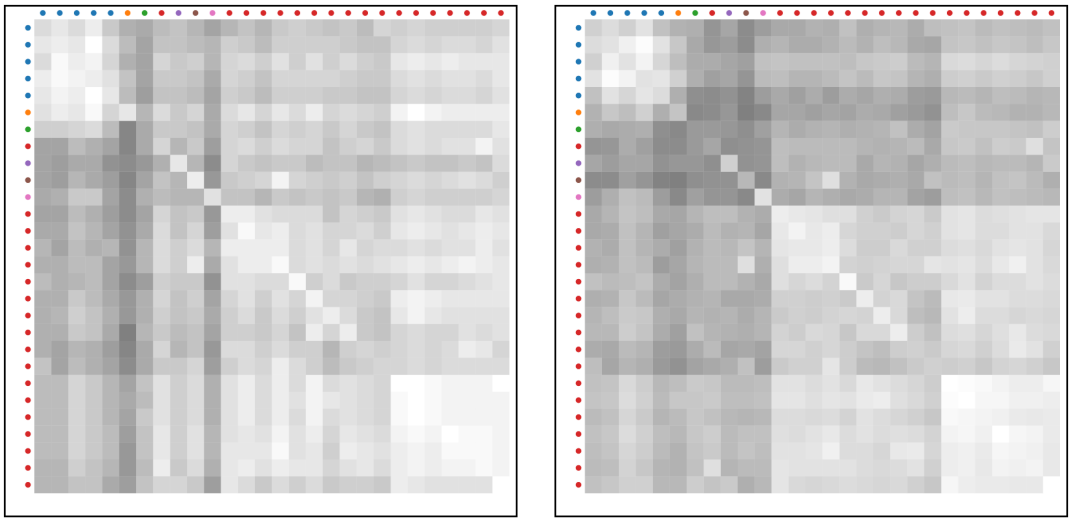
Beyond reducing techniques to vectors and comparing their distances in vector space, GLOs enable a novel, non-vector-based distance metric for comparing techniques. During the GLO identification process, a transition matrix was created where each of the cells of the matrix were the operations necessary to transition from one technique to another. For example, the seven operation transition-matrix entry for (Semantic Substrates, PivotGraph) described in Section 3.2.1.

These transitions can be used to define a **GLO Distance** between two techniques. In other words, how many operations are required to transition between two techniques? Notably, this is not a symmetric distance (see Figure 58a). This can make comparing two techniques more difficult. To simplify the comparison, one can simply sum the number of operations to transition back and forth between any two techniques to create a symmetric distance metric (see Figure 58b).[8]

Two clear clusters appear in the top-left and bottom-right corners of both matrices. The top corner cluster (Figure 59a) consists of the matrix-based plots. Note, the NodeTrix seed technique is the darker last row/column in this cluster. The bottom corner cluster (Figure 59b) consists of all techniques with a single generation of constantly-sized nodes (Force-Directed Layout, Arc Diagram, Circle Plot, Cluster Circles, GeneVis A, GeneVis B, and Hive Plots). Notably, these techniques are fairly easy for other techniques to transition to, since they are all instances of the null state.

In the symmetric distance matrix plot, a third cluster is visible in the middle of the display (Figure 59c). This cluster consists of seed techniques that position and/or color a single generation of unaggregated nodes based on attributes of the data (ScatterNet, CiteVis, EdgeMap A, EdgeMap B, and Semantic Substrates).

[8]In order to achieve a reasonable node ordering for the Matrix Plot displays in Figure 58, I clustered the GLOv2 seed techniques using the symmetric distance metric using scikit-learn's [179] agglomerative clustering (n=5 clusters, average linkage, and pre-computed distance parameters).



(a) Non-Symmetric (b) Symmetric

Figure 58: GLOv2 seed techniques clustered by symmetric transition distance rendered with GLO.js. On the left, edges are colored by the one-way transition distance. On the right, edges are colored by the symmetric sum of the transition distances. Rendered using GLO.js

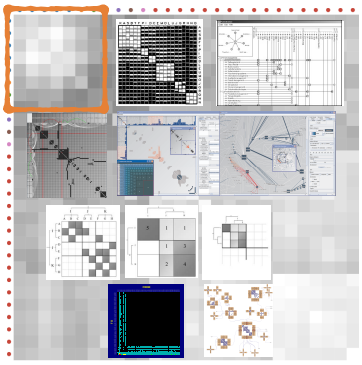
Identifying these clusters demonstrates the analysis potential of the GLO Distance metric.

5.3 Identifying New Techniques

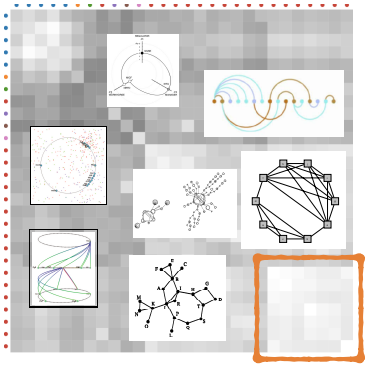
Most graph visualization techniques were developed with a specific task in mind. For example, sorted matrix layouts are effective for showing clusters in a graph; PivotGraphs are useful for showing how nodes with different properties interact; and the interaction of Semantic Substrates (showing only the edges adjacent to a specific node) is useful for reducing edge occlusion. But what if someone wanted a technique designed for the task of reducing occlusion while also seeing how groups of nodes interact with other groups? In this case, combining the layout of PivotGraphs with the interaction of semantic substrates would be highly effective.

We can describe such a technique using GLOv2:

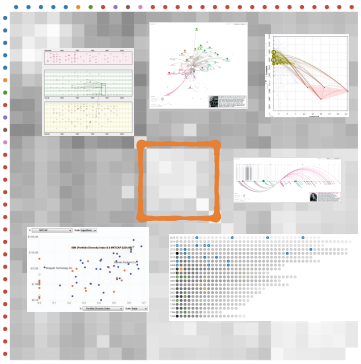
- display nodes as circles



(a) 'Matrix' Cluster



(b) 'Simple' Cluster



(c) Attribute-Driven Cluster

Figure 59: Clusters visible in the symmetric GLO Distance matrix.

- **show incident edges**
- aggregate nodes by {*[discrete1,discrete2]*} using {*agg-method*}
- size nodes by {*size-attr*}
- color nodes by {*node-color-attr*}
- aggregate edges by {*[source.discrete1, source.discrete2,target.discrete1,target.discrete2]*} using {*agg-method*}
- display edges as curved lines
- size edges by {*count*}
- color edges by {*count*}
- position nodes on {y} by {*discrete1*}
- position nodes on {x} by {*discrete2*}
- show axis(x)

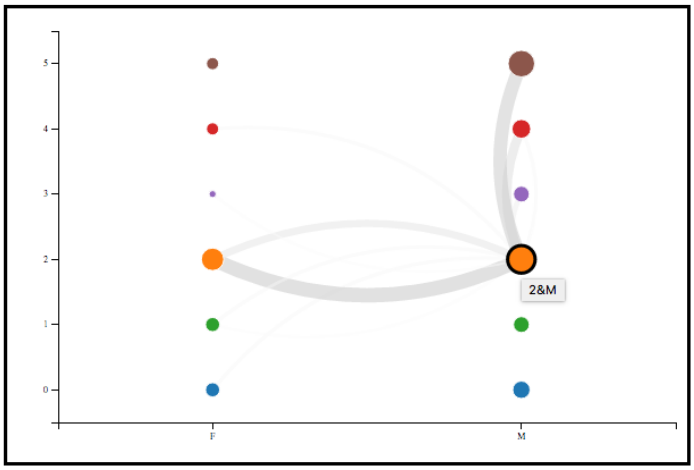


Figure 60: Novel graph visualization technique combining the layout of PivotGraphs [244] with the interaction of Semantic Substrates [96].

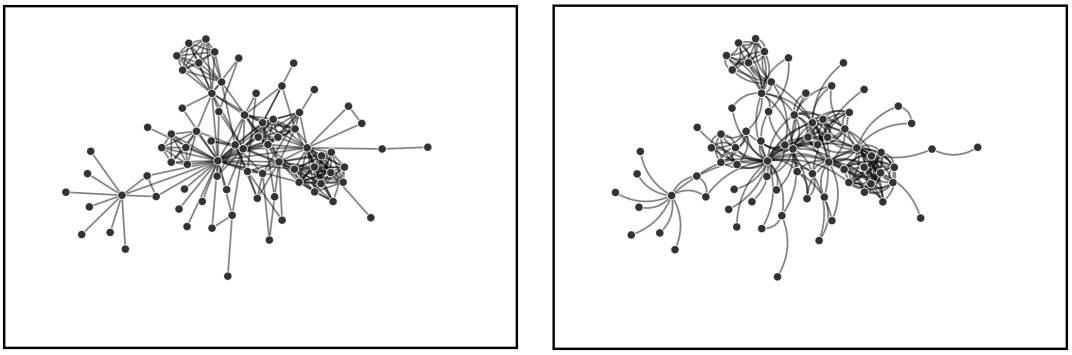
- show axis(y)

Figure 60 shows this novel technique. In this case, the only difference between this specification and the PivotGraph GLOv2 seed technique specification in Appendix B is replacing the *show all edges* GLO with the *show incident edges* GLO.

Predefined techniques are quite powerful and are certainly popular. We can see this in systems such as Excel [164], Spotfire [223], and Tableau [219] that enable analysts to visualize data sets using pre-defined, standard techniques. However, visualization researchers and analysts are always on the look out for new techniques for effectively communicating data. The PivotGraph/Semantic Substrates hybrid demonstrates that there exist interesting techniques to be found in the "space between" existing techniques. Simply changing a single GLO in the specification of a known effective technique can result in an interesting modified technique that is potentially effective at other tasks. In this way, GLOs provide a benefit to the visualization community by providing a novel means of identifying new techniques.

In some cases, like the modified PivotGraph above, seed techniques can be **tweaked** using GLOs. In the trivial case, one could display a force-directed layout with curved edges instead of the seed technique's straight edges (see Figure 61). While this hardly

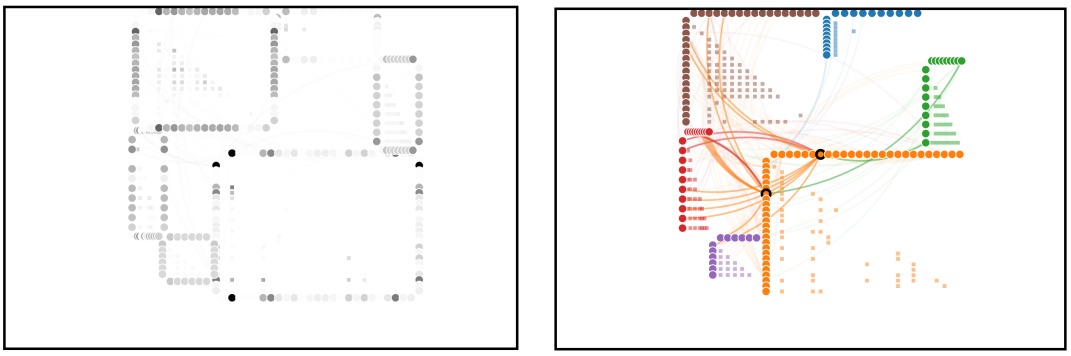
constitutes a radical new graph visualization technique, it might be useful for certain purposes. For example, the curved edges are always drawn counter-clockwise, and therefore they might make it easier to recognize features of the graph that are dependent on directionality than with straight lines.



(a) Seed Technique (b) Tweaked Technique

Figure 61: Example of a minor tweak to a seed technique by changing the Force-Directed Layout seed technique's edge display mode to curved lines.

On the other hand, one can also use GLOs to more substantially **fix** techniques. For example, consider the NodeTrix [117] GLOv2 seed technique in Figure 62a. The over-duplication of nodes adds little to the display and a few high edge weights and node degrees overwhelm the technique's color scales.



(a) Seed Technique (b) Modified Technique

Figure 62: NodeTrix [117] GLOv2 seed technique and modified NodeTrix display with nodes colored by cluster, intra-cluster edges colored by a constant, faded intra-cluster edges, and highlighting intra-cluster edges incident to a selected node created using GLO.js.

The specification in Table 10 (and rendered in Figure 62b) modifies the seed technique significantly. Rather than degree, nodes are colored by their clusters, edges are colored by their target nodes, intra-cluster edges are shown at full opacity, and the specification uses the *show faded and incident edges* operation with a group-by optional parameter to show inter-cluster edges as faded, but inter-cluster edges incident to a selected node (shown circled in black) with full opacity.

Is this modified NodeTrix an entirely novel technique? Not necessarily, yet it certainly provides much more utility than the tweaked Force-Directed Layout.

```
# Glyph properties
display nodes as {label-attr} labels
color nodes by {discrete}
size nodes by constant
size edges by constant

# Circle of nodes
position nodes by constant on {ρ}
evenly distribute nodes on {θ} (sort-by:discrete)

# Left columns
align nodes {left} (group-by:discrete)
evenly distribute nodes on {y} (group-by:discrete, invert:true)

# Top rows
clone nodes
rotate nodes {90}
evenly distribute nodes on {x} (group-by:discrete)
align nodes {top} (group-by:discrete)

# All edges (incl. inter-cluster edges to row)
set target generation {1}
color edges by {target.discrete}

# Inter-cluster edges
display edges as curved lines
show faded and incident edges

# Intra-cluster edges
display edges as squares (group-by:discrete)
show all edges (group-by:discrete)

# Additional inter-cluster edges to column
clone edges
hide edges (group-by:discrete)
set source generation {1}
set target generation {0}
```

Table 10: GLOv2 specification for modified NodeTrix display in Figure 62b.

GLOs can also enable the specification of completely new techniques beyond the initial set of seed techniques. Let me use a simple example to demonstrate this point.

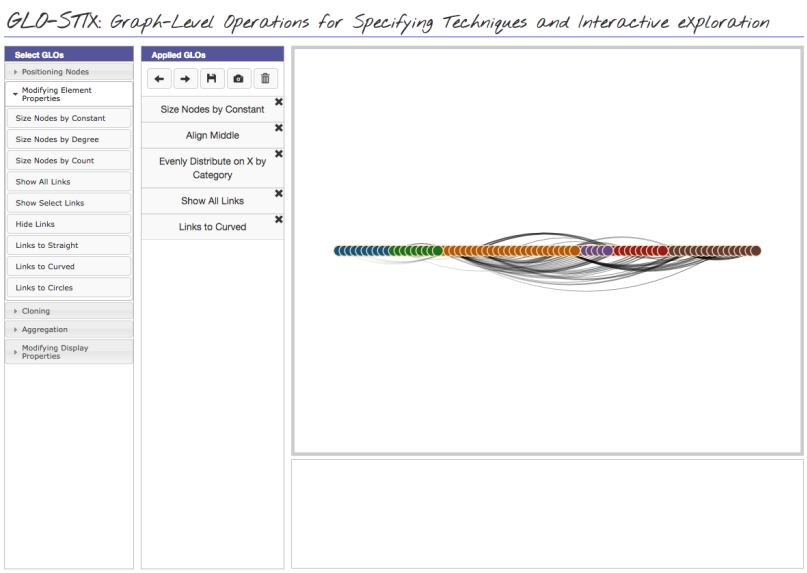


Figure 63: Arc Diagram [243] created using GLOv1 within GLO-STIX.

Consider GLOv1 and recall that the seed techniques for GLOv1 (Section A) explicitly did not include Arc Diagrams [243]. This was a conscious decision. Using GLOv1, we can still specify Arc Diagrams as:

- Size Nodes by Constant
- Align Nodes Middle
- Evenly Distribute Nodes on x
- Display All Links
- Display Links as Curved

Figure 63 shows an Arc Diagram created using this specification in GLO-STIX.

In the case of GLOv2, I did not explicitly choose to leave out a technique in this way. However, one can simulate this by identifying those seed techniques whose specifications only include operations found in other specifications. This simulated leave-one-out approach demonstrates that only 5 (GMap, Matrix with Bars, Matrix Browser, Edge-Label-Centric, and CiteVis) of the 29 GLOv2 seed techniques could not have been represented using only operations from other specifications. This represents

strong evidence of the reusability and generalizability of GLOs.[9]

In my experience, I have found the *partition canvas* and *filter partition canvas* operations particularly useful for creating novel visualizations. (Recall that the *partition canvas* GLO enables interactively linking any stand-alone techniques, while the *filter partition canvas* GLO enables creating small multiples of each cluster of a graph.) It is worth noting that with the expressiveness that graph-level operations provide, not every novel technique will be all that useful for graph analysis tasks. For example, Table 11 is the GLOv2 specification for the 'GLO' teaser image (Figure 1) from Chapter 1. It takes advantage of the *partition canvas* GLO to create three linked views of three distinct techniques.

```
# Initial glyph properties
display nodes as circles
display edges as straight lines
color nodes by {discrete}
size nodes by constant
color edges by constant
size edges by constant
show edges as faded

# Partition into 3 parts
partition on {x} (parts:3)

# G
select canvas {0}
evenly distribute nodes on {θ} (sort-by:discrete)
evenly distribute nodes on {ρ} (sort-by:discrete)

# L
select canvas {1}
display edges as squares
align nodes {left}
evenly distribute nodes on {y} (sort-by:discrete, invert:true)
clone nodes
set target generation {3}
align nodes {bottom}
evenly distribute nodes on {x} (sort-by:discrete)

# O
select canvas {2}
evenly distribute nodes on {θ} (sort-by:discrete)
position nodes by constant on {ρ}
display edges as curved lines
```

Table 11: GLOv2 specification for 'GLO' teaser technique in Figure 1.

[9]Furthermore, the only operation in the CiteVis specification that does not occur in another technique is the *highlight in-out neighbors* GL. While this operation could simply replaced by the similar *highlight neighbors* GLO, the resulting technique would be a slightly different technique than the seed technique.

While these three techniques were chosen for their similarity to the letters G, L, and O rather than for their utility for a specific task, one could imagine using appropriate techniques for a specific task instead.

5.3.1 Approximate Measures of GLO Expressiveness

When discussing specifying novel techniques, it it worth quantifying what can be expressed under a given GLO model and a given interpreter. To that end, here I quantify the ways in which a single generation of unaggregated node or edge glyphs can be represented using GLOs under the GLOv1 and GLOv2 models and the GLO.js interpreter. Note that this is a lower-bound on the expressiveness since cloning, aggregation, partitioning, and optional parameters can further increase the expressiveness beyond these counts.

GLOv1 Expressiveness For a given unaggregated node generation in GLOv1 there are:

- 15 positioning operations (4 of which support x or y axes)
- 1 display mode (circles)
- 1 display property (size)

For a given unaggregated edge generation in GLOv1 there are:

- 3 display modes (straight lines, curved lines, and circles)
- 3 interaction modes (show all edges, show incident edges, show no edges)

Node aggregation further increases the expressiveness of the model.

GLOv2 Expressiveness For a given unaggregated node generation in a given canvas (i.e., with no partitioning or cloning) in GLOv2 there are:

- 7 positioning operations (each with various axis and attribute combinations)

- 4 display modes (circles, squares, labels, and bars)
- 3 display properties (size, color, and rotation)
- 2 convex hull display options (show, hide)
- 3 interaction modes (no interaction, highlight neighbors, highlight in-out-neighbors)

For a given unaggregated edge generation in a given canvas (i.e., with no partitioning or cloning) in GLOv2 there are:

- 3 positioning operations (each with various axis and attribute combinations)
- 6 display modes (straight lines, curved lines, squares, labels, bars, and right angles)
- 2 display properties (size, color)
- 2 waypoint modes (on or off)
- 6 interaction/visibility modes (show none, show all, show faded, show incident, show in-out, and show faded and incident)

Aggregation and cloning of nodes and edges and partitioning of the GLO Display further increases the expressiveness of the model.

GLO.js Expressiveness For a given unaggregated node generation in a given canvas (i.e., with no partitioning or cloning), at the time of this dissertation, GLO.js supports:

- 6 positioning operations (each with various axis and attribute combinations)
- 1 display mode (circles)
- 2 display properties (size and color)
- 0 convex hull display options
- 2 interaction modes (no interaction or highlight neighbors)

and each of these can be modified using a group-by attribute.

For a given unaggregated edge generation in a given canvas (i.e., with no partitioning or cloning) GLO.js supports:

- 2 positioning operations (each with various axis and attribute combinations)
- 3 display modes (straight lines, curved lines, and squares)
- 2 display properties (size, color)
- 0 waypoint modes
- 5 interaction/visibility modes (show none, show all, show faded, show incident, and show faded and incident)

and each of these can be modified using a group-by attribute.

Aggregation and cloning of nodes and edges and partitioning of the GLO Display further increases the expressiveness of the implementation.

CHAPTER VI

CONCLUSION

6.1 Contributions and Impact

To recap the contribution and impact of this dissertation on five facets of graph visualization research and practice:

Models I presented a novel class of graph visualization model, the graph-level operations model (GLO model) including a method for inducing a model from a set of canonical seed techniques, two instances of GLO models (GLOv1 and GLOv2), and a means of defining techniques using the model.

Analysis Methods I introduced GLO-based methods for reducing techniques to vector representations as well as a novel GLO-based distance metric for techniques in Chapter 5 to demonstrate how the GLO model represents a giant leap forward in our ability to easily and effectively compare and cluster graph visualization techniques.

Open-Source Software I presented the GLO.js graph visualization toolkit to easily incorporate a large variety of graph visualization techniques into web-based graph analysis software and used the toolkit to build the GLO-STIX GUI application and GLO-CLI command-line application for using visualization to explore a network. All three packages are available as open-source software at `https://github.com/chadstolper/glo`.

Techniques Throughout this dissertation, I introduced a number of novel techniques identified using the GLO model and provide their definitions using GLOs.

Education Below, I describe the potential that graph-level operations have to revolutionize graph visualization education through demonstrating the variety and interconnectedness of graph visualization techniques and on discrete mathematics education through enabling demonstrations of graph theory properties and algorithms.

6.2 Limits of GLOs

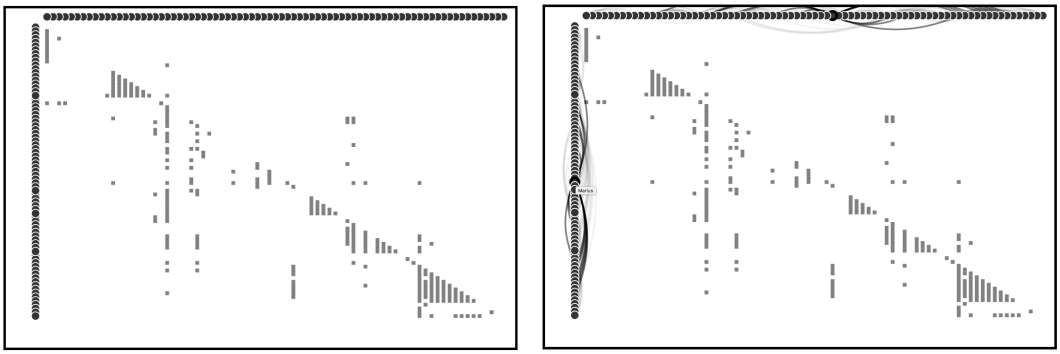
Graph-level operations models are a powerful class of models for graph visualization. At the same time, they do have limits. First of all, as shown by the differences between GLOv1 and GLOv2, there is inherent imprecision in determining the set of operations using the GLO identification method. This imprecision can be minimized through precise instructions for an identifier and through consistent adoption of baseline GLO models (such as GLOv1 or GLOv2). In addition, having mappings between baseline models (such as the mappings in Table 3 between GLOv1 and GLOv2) can help ensure this consistency.

As is common with models induced from training data, GLO models are dependent on the set of seed techniques used to induce them. The GLOv2 operations in Table 12 appear in the specification of a single GLOv2 seed technique. Had the associated techniques not been included in the set of seed techniques, those operations would not have been included in the resulting set of operations. Similarly, if every technique has a feature in common then that feature would not appear in the transition matrix. For example, Arc Diagrams and EdgeMap B both display node glyphs aligned in the middle of the y axis. Performing the identification process on only these two techniques would not identify any operations that adjust the y coordinate of node glyphs. Identifying seed techniques through extensive searches, such as the literature review I conducted for GLOv2, can minimize this effect.

In Chapter 5, I described metrics for determining similarity between techniques such as comparing vector representations of specifications and utilizing the novel GLO

GLO	GLOv2 Seed Technique
show edges as faded	GMap
show convex hulls	GMap
color convex hulls by {*attr*}	GMap
display edges as bars	Matrix with Bars
display edges as right angles	Matrix Browser
align edges {*dir*}	Edge-Label-Centric
display edges as {*attr*} labels	Edge-Label-Centric
evenly distribute edges on {*axis*} (by {*attr*})	Edge-Label-Centric
set edge waypoint edge generation {*num*}	Edge-Label-Centric
highlight in-out neighbors	Citevis

Table 12: GLOv2 operations unique to a single seed technique.



(a) Matrix Plot (b) MatLink

Figure 64: Matrix Plot and MatLink [118] techniques rendered in GLO.js. Transitioning between these two techniques is more efficient without transitioning through an intermediate null state.

Distance metric. Notably, this latter metric can be naively computed by reducing the first technique to the null state and then creating the new technique from its specification. While this naive method occasionally results in an optimal transition, more often there is a more efficient transition that a human identifier can find. For example, it is more efficient to transition between a Matrix Plot and a MatLink display (Figure 64) without reducing to a null state since both techniques utilize two generations of node label glyphs in the same positions. There is potential for algorithms that can identify more efficient transitions. For example, an algorithm might start by identifying similar generations in both techniques (e.g. the node generations in the matrix-based plots) and attempt to transition those elements first before utilizing inverse GLOs.

Using vectorization to compare techniques is quite useful for computing similarity

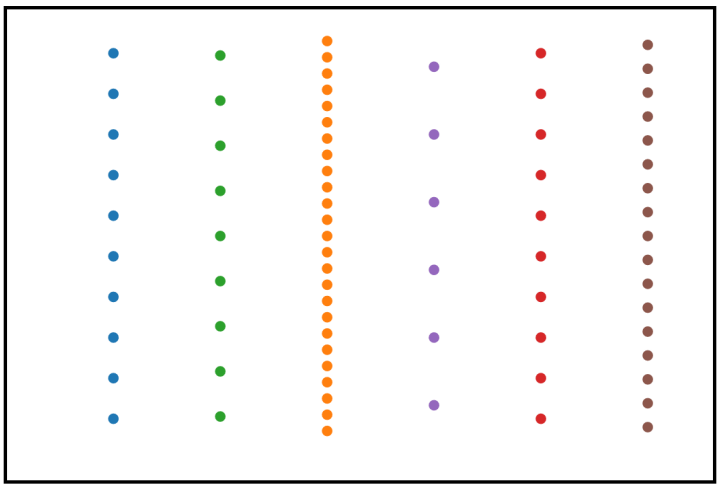


Figure 65: Single technique defined using two distinct specifications.

between techniques, as I have shown. However, using vectors to determine equivalence is much more challenging. Two techniques are equivalent if and only if their specifications result in the same glyphs with the same properties (including display mode, interaction mode, and visual properties). Note that two techniques with the same set of operations are not necessarily equivalent (recall the discussion of the Circle Plot and Cluster Circles seed techniques) and two techniques with different sets of operations could be equivalent.

For example, the following two techniques are equivalent (both generate the display in Figure 65) even though they have different operation sets:

- display nodes as circles
- size nodes by constant
- color nodes by {*discrete*}
- hide edges
- align nodes middle
- aggregate nodes by {*discrete*}
- evenly distribute nodes on x (sort-by {*discrete*})
- deaggregate nodes
- evenly distribute nodes on y (within {*discrete*})

and

- display nodes as circles
- size nodes by constant
- color nodes by $\{discrete\}$
- hide edges
- align nodes middle
- position nodes on x by $\{discrete\}$
- evenly distribute nodes on y (within $\{discrete\}$)

Of course, this equivalence challenge exists with many lossy encoding schemes. For example, the prevalent 'bag-of-words' schemes for text document vectorization similarly struggle at judging equivalence since the scheme ignores the order of the words in the document.

Finally, an important property of both the GLOv1 and GLOv2 GLO models is that operations affect every node glyph or every edge glyph in a generation equally. In contrast, most tree and DAG visualization techniques (and graph visualization techniques that reduce the graph to a tree or a DAG such as ego-centric graph visualization) depend on considering nodes differently based on their distance to the root of the tree. For this reason, I expect that a GLO-like model that covers the variety of tree visualization techniques will likely need to be considerably different than either of the two models I have presented here.

6.3 Future Research Directions

Even with their limitation, the GLO models remain powerful and useful. For example, the models open up numerous pathways for interesting and worthwhile research. I end this dissertation by briefly describing six research avenues graph-level operations afford.

First, fully constraining techniques is useful for rendering techniques, but when

comparing and understanding techniques, flexibility proves useful. For example, to transition from the Force-Directed Layout GLOv2 seed technique to the GMap GLOv2 seed technique requires the following operations:

- show edges as faded
- size nodes by {*attr*}
- display nodes as {*attr*} labels
- show convex hulls within {*discrete*}
- color convex hulls by {*attr*}

However, imagine a node-link diagram with solid edges and circular, constant-sized nodes with convex hulls displayed around each cluster. While not the GLOv2 seed technique GMap, it is still a GMap. In that sense, only the last two operations are really necessary for a transition between the Force-Directed Layout seed technique and *a* GMap (as opposed to *the* GMap seed technique). This suggests an alternate description for GMap using GLOv2:

- One of
  - show all edges
  - show edges as faded
  - show faded and incident edges
  - show in-out edges
  - show incident edges
- One of
  - display edges as curved lines
  - display edges as straight lines
- One of
  - display nodes as labels

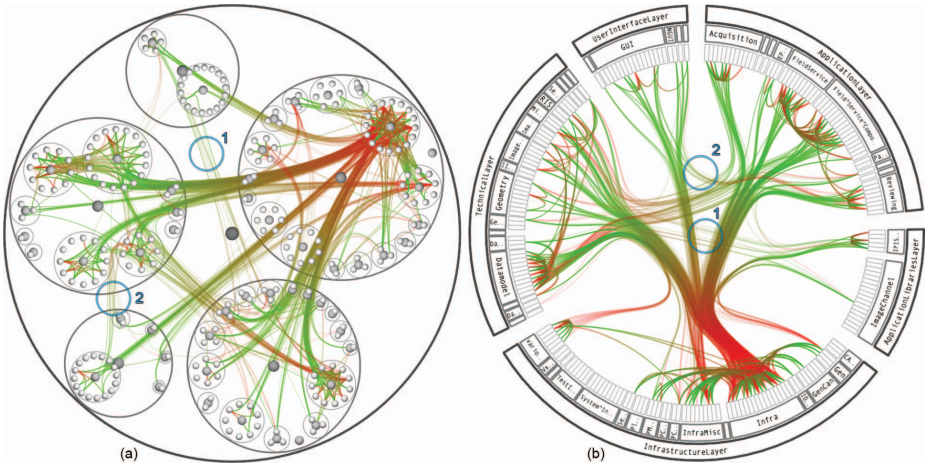


Figure 66: Examples of edge bundling from [120].

  - display nodes as circles
  - display nodes as squares

- show convex hulls within {*discrete*}
- color convex hulls by {*attr*}

I have not studied these more flexible technique specifications deeply, but they are certainly worth pursuing further.

The second research direction concerns **technique-independent graph-level operations (TI-GLOs)**. (I.e., the visual and interaction sub-techniques that I chose to exclude from the GLOv2 seed technique set.) The most obvious example of a visual technique-independent operation is edge-bundling [120] (Figure 66). Edge bundling is not critical to the definition of any technique. Rather, edge-bundling can simply be applied to any line-based edge generation. Other examples of technique-independent operations include subgraph motif glyphs [71] and alternate directed edge glyphs [124, 121, 170]. One could imagine a given GLO interpreter applying one or more of these visual TI-GLOs by default, but it would likely prove useful to allow specifications to include this class of operations. A potential solution would be to make TI-GLOs optional for interpreters. In other words, an interpreter can simply support some, all,

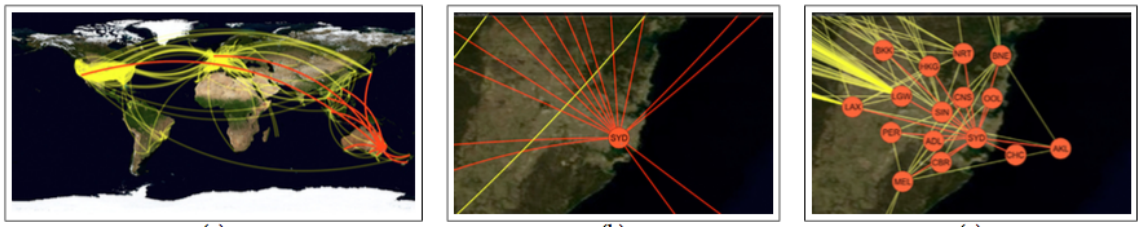


Figure 67: Bring-and-Go interaction from [166].

or no visual TI-GLOs.

Fitting interaction TI-GLOs into a GLO model might provide a greater challenge. I expect that some interactions would fit seamlessly. For example, the Bring-and-Go interaction from [166] (Figure 67) is quite similar in many respects to GLOv2's highlight neighbors interaction mode. On the other hand, as I described in Chapter 4, multi-selection is more challenging. Some aspects of the GLO models that are topology dependent (such as the highlight in-out-neighbors interaction mode) do not work well with multi-selection. Exploring the full extent of whether and how additional interactions can be integrated into GLO models merits further explorer.

Third, I defined the three technique vectorization methods in order to compare them to each other. Yet recall that the overarching goal is to automate mapping techniques to tasks. For example, consider a database of techniques that are known for their ability to aid in certain analytical tasks. Should a situation require a set of tasks, one might query this database for a technique that either can solve both tasks or a technique that is similar to techniques that can solve each of the tasks.

There is also potential for going further. Rather than simply mapping techniques to tasks, one could potentially map GLOs to tasks. In other words, do certain operations signal that a technique will be effective at a certain task? For example, I hypothesize that the *display incident edges* GLO signals that a technique is effective at analyzing dense graphs since it reduces occlusion. Conversely, this GLO likely signals that a technique is less effective for sparse graphs, where hiding most of the edges could lead to the analyst never noticing a critical edge.

Another potential result is that certain operations do not signal strength in a particular task, but rather certain *sequences* of operations do. A good example is that simply drawing convex hulls is less effective than drawing convex hulls and coloring those hulls by an attribute of the data. Performing frequent sequence analysis over a corpus of techniques known to be effective at a task could provide valuable insights.

Fourth, graph-level operations provide rich potential for teaching discrete mathematics and teaching graph visualization. For discrete math education, GLOs can be used to demonstrate graph theory concepts using appropriate techniques. For visualization education, GLOs make it easier to demonstrate the variety of graph visualization techniques and show how they relate to each other through transitions.

Fifth, there are more graph visualization techniques to be discovered! I described novel techniques in Chapter 5, but there are certainly more to be identified. Some may be found through matching tasks to GLOs, some through exploration inside the GLO-CLI terminal, and some may even be found through running a shuffle algorithm on the set of operations. Identifying (and confirming the effectiveness of) novel GLO-specified techniques hopefully will lead to a more robust collection of visualization techniques, and therefore more effective solutions for graph analysts.

Sixth and finally, I hope that GLOs are adopted as a standard. Much of the power of GLOs rests in their consistency, such as being a portable specification language between different host language and graphics implementations. I hope that developers who code for architectures such as python and R write GLO interpreters for them so that techniques specified using GLOs can be rendered easily no matter the underlying system.

APPENDIX A

GLOV1 SEED TECHNIQUES

Here I present the set of 6 hand-picked abstract seed techniques used to induce GLOv1 (See Chapter 3). For each technique I include a figure representing the technique rendered using the GLOv1 Javascript interpreter, a description of the technique, and a GLOv1 specification for the technique (i.e., a transition to the technique from a null state).

The graph being rendered in each figure is the Les Misérables character co-occurrence graph included with D3.js based on Donald Knuth's jean.dat file[1]. Nodes are characters, and an edge connects two characters if they co-occur in a chapter of the novel. For each operation that requires one or more parameters, I note the parameters used in a #comment the first time the parameter appears in the specification.

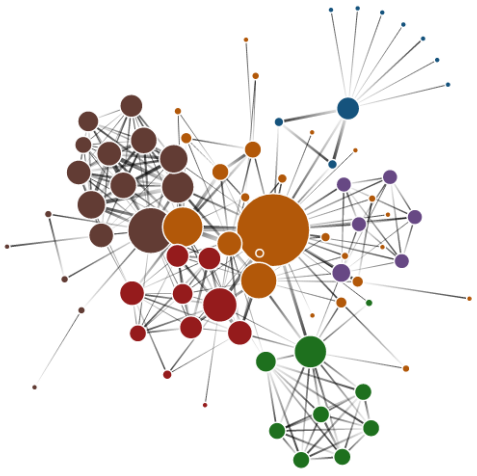


Figure 68: GLOv1 force-directed layout seed technique.

[1]http://www-cs-staff.stanford.edu/ uno/sgb.html

Force-Directed Layout The GLOv1 seed version of the Force-Directed Layout technique (Figure 68) has straight line edge glyphs, node glyphs sized by an attribute, and node glyphs colored by an attribute. Nodes are positioned according to a force-directed algorithm.

- Size Nodes Relatively by {*continuous attribute*} #degree
- Apply {*force-directed*} algorithm to the Nodes
- Display Links as Straight
- Display All Links

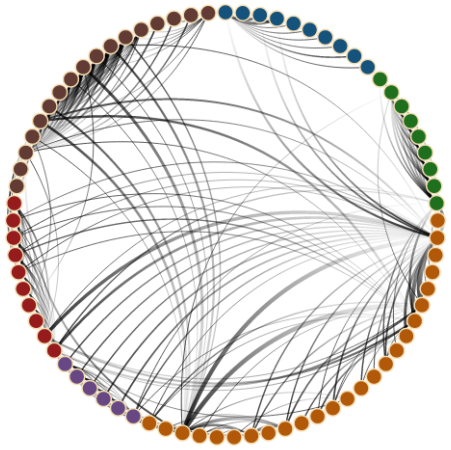


Figure 69: GLOv1 circle plot seed technique.

Circle Plot The GLOv1 seed version of the Circle Plot technique (Figure 69) has curved line edge glyphs, node glyphs sized by a constant, node glyphs colored by an attribute, and the node glyphs are sorted along θ by an attribute.

- Size Nodes by Constant
- Evenly Distribute Nodes Radially by {*attribute*} #cluster
- Position Nodes Along Plot Radius by Constant
- Display All Links
- Display Links as Curved

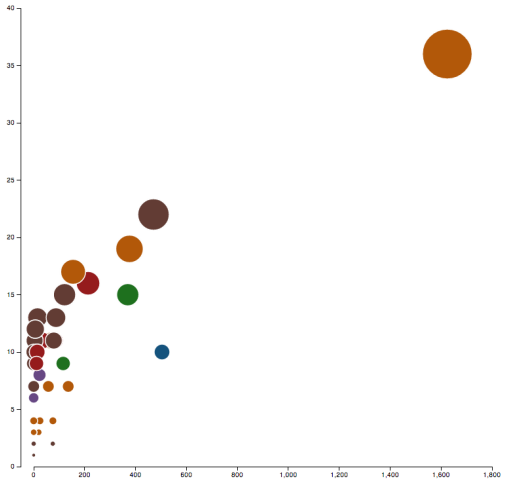


Figure 70: GLOv1 scatterplot seed technique.

Scatterplot The GLOv1 seed version of the Scatterplot technique (Figure 70) has no edge glyphs, node glyphs sized by an attribute, and node glyphs colored by an attribute. Nodes are positioned along the x and y axes by continuous attributes and axis labels are displayed on both axes.

- Hide Links
- Size Nodes Relatively by {*continuous attribute*} #degree
- Position Nodes on xRelatively by {*continuous attribute 1*} #degree
- Position Nodes on y Relatively by {*continuous attribute 2*} #betweenness centrality
- Show x axis
- Show y axis

Semantic Substrates The GLOv1 seed version of the semantic substrates technique [204] (Figure 71) has curved line edge glyphs that appear when the mouse hovers over an endpoint node, node glyphs sized by a constant, and node glyphs colored by an attribute. Nodes are positioned on the y axis by a discrete attribute and are distributed across the canvas within each row. Axis labels are shown for the y axis.

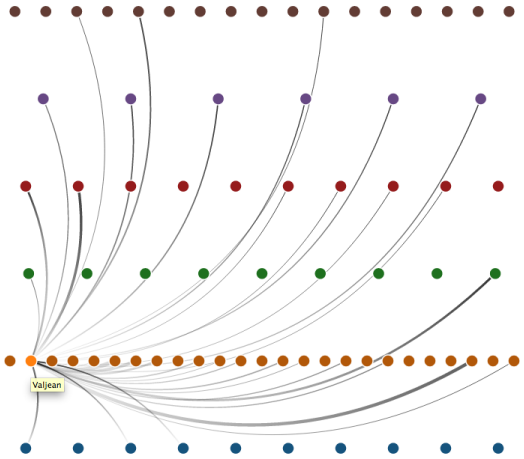


Figure 71: GLOv1 semantic substrates [204] seed technique.

- Size Nodes by Constant
- Substrate Nodes on y by {*categorical attribute*} #cluster
- Show y Axis
- Evenly Distribute Nodes within Substrates
- Display Links as Curved
- Display Selected Links

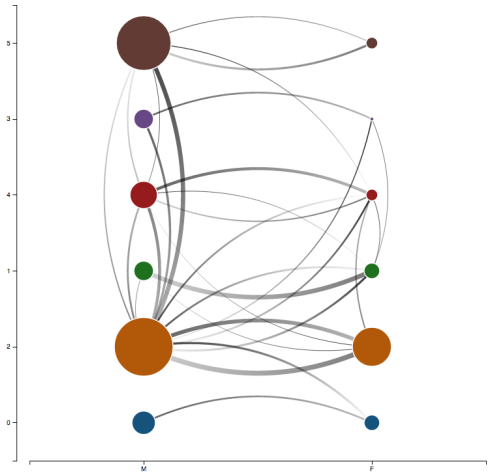


Figure 72: GLOv1 PivotGraph [244] seed technique.

PivotGraph The GLOv1 seed version of the PivotGraph technique [244] (Figure 72) has nodes and edges aggregated by two attributes, curved line edge glyphs,

node glyphs sized by the number of represented nodes, edge glyphs sized by the number of represented edges, and node glyphs colored by an attribute. Super-nodes are positioned along a grid based on the two aggregation attributes. Axis labels are shown along both the x and y axes.

- Display All Links
- Substrate Nodes on x by {*categorical attribute 1*} #cluster
- Substrate Nodes on y by {*categorical attribute 2*} #gender
- Show x Axis
- Show y Axis
- Aggregate by {*categorical attribute 1*} and {*categorical attribute 2*}
- Size Nodes Relatively by {*continuous attribute*} #count

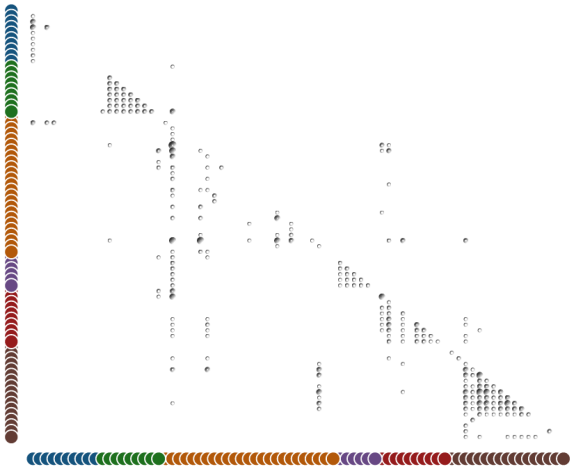


Figure 73: GLOv1 adjacency matrix seed technique.

Adjacency Matrix The GLOv1 seed version of the adjacency matrix technique (Figure 73) has two sets of node glyphs (one aligned on the left and one on the bottom) sorted by an attribute, node glyphs sized by a constant, and node glyphs colored by an attribute. Circle edge glyphs are positioned at the y value of the source node on the left and the x value of the target node on the bottom.

- #Bottom row of nodes
- Size Nodes by Constant
- Align Nodes {*Bottom*}
- Evenly Distribute Nodes on x by {*attribute 1*} #cluster
- #Left column of nodes
- Clone Active Generation
- Align Nodes {*Left*}
- Evenly Distribute Nodes on y by {*attribute 1*}
- #Links
- Display All Links
- Display Links as Circles #(includes positioning)

APPENDIX B

GLOV2 SEED TECHNIQUES

Here I present the 29 seed techniques used to induce GLOv2. For each technique, I include a figure from an early paper describing the technique, provide a description of the abstract form of the seed technique, and a GLO specification of the abstract technique (i.e. a transition from the null state). Any necessary flags or attributes are marked in the specifications. For those techniques that either can be rendered precisely or a close approximation can be rendered using the GLO.js implementation described in Chapter 5.1.1, I also include a rendering of the technique.

The graph being rendered in each figure is the Les Misérables character co-occurrence graph included with D3.js based on Donald Knuth's jean.dat file[1]. Nodes are characters, and an edge connects two characters if they co-occur in a chapter of the novel. For each technique that requires one or more parameters, I note the parameters used (or could be used if a rending was not included) in #comments at the beginning of the specification.

Force-Directed Layout [138]

The force-directed layout seed technique has constant-sized, constant-colored circular nodes connected by constant-sized, constant-colored straight-line edges. The nodes are positioned using a force-directed layout.

I have abstracted away the textual node labels.

- display nodes as circles
- display edges as straight lines

[1] http://www-cs-staff.stanford.edu/ uno/sgb.html

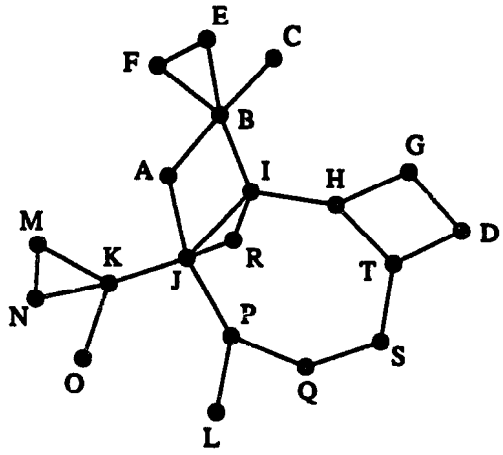


Figure 74: GLOv2 Force-Directed Layout seed technique from [138].

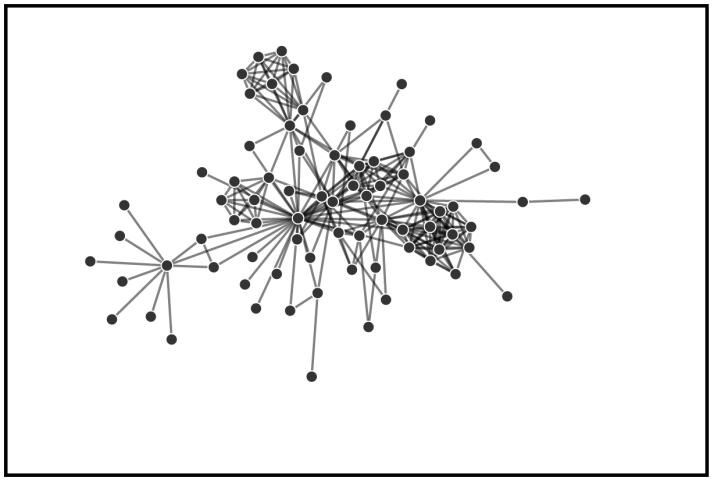


Figure 75: Force-directed layout [138] rendered in GLO.js.

- show all edges
- size nodes by constant
- size edges by constant
- color edges by constant
- color nodes by constant
- apply force-directed algorithm to nodes

Matrix Plot [34]

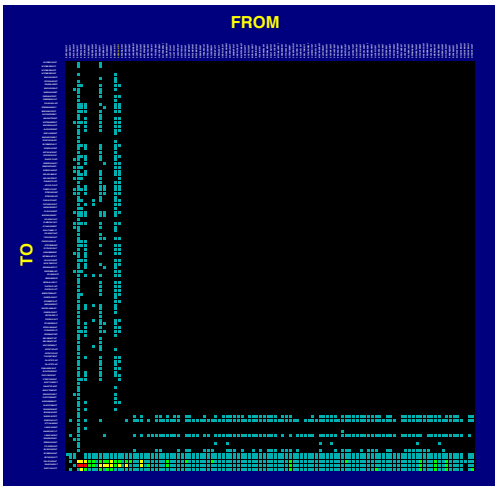


Figure 76: GLOv2 Matrix Plot seed technique from [34].

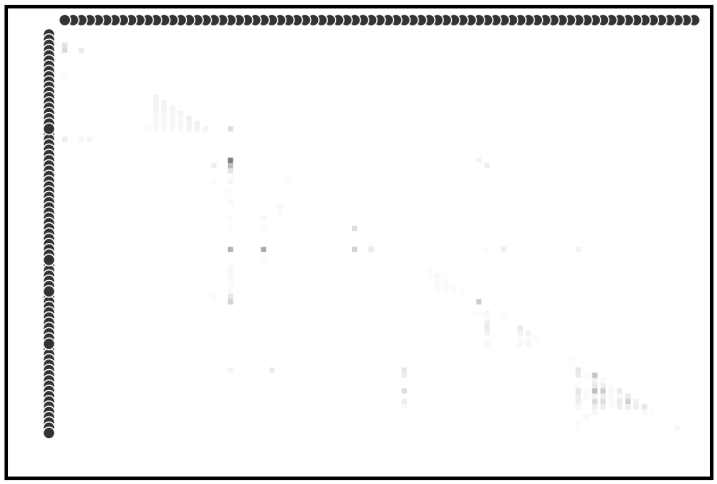


Figure 77: Approximate matrix plot [34] rendered in GLO.js.

The Matrix Plot seed technique has nodes displayed as constant-sized, constant-colored labels evenly spaced along the top and left, with edges displayed as constant-sized squares colored by an attribute positioned at the y value of the source node on the left and x value of the the target node on the top. On the left, the nodes are sorted top to bottom, while on top the nodes are sorted left to right.

I have abstracted away the specific color choices and the TO and FROM labels from the seed figure.

- #label_attr: name

- #sort_attr: cluster
- #edge_color_attr: weight
- #Left column
- display nodes as {*label_attr*} labels
- color nodes by constant
- size nodes by constant
- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true)
- align nodes {*left*}
- #Top row
- clone nodes
- rotate nodes {*90*}
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*})
- #Edges
- set target generation {*1*}
- display edges as squares
- position edges by {*target.x*},{*source.y*}
- show all edges
- size edges by constant
- color edges by {*edge_color_attr*}

Cluster Circles [69]

The Circle Clusters seed technique has constant-sized, constant-colored square nodes connected by constant-sized, constant-colored straight-line edges. The nodes are positioned such that the nodes each cluster are in evenly-distributed circles.

- #group_by_attr: cluster
- #internal_sort_attr: id

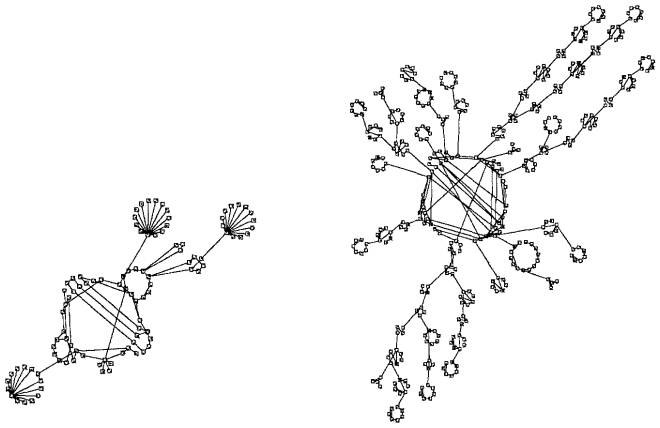


Figure 78: GLOv2 Cluster Circles seed technique from [69].

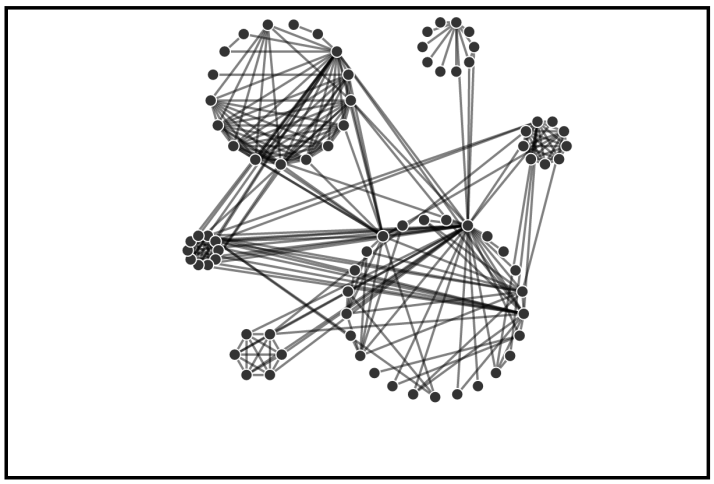


Figure 79: Cluster circles [69] rendered in GLO.js.

- display nodes as squares
- show all edges
- display edges as straight lines
- color edges by constant
- color nodes by constant
- size nodes by constant
- size edges by constant

  #Macro-circle positioning

- evenly distribute nodes on θ (sort-by:{*group_by_attr*})

- position nodes by constant on ρ

  #Micro-circles positioning

- evenly distribute nodes on θ (sort-by:{*internal_sort_attr*}, group-by:{*group_by_attr*})
- position nodes by constant on ρ (group-by:{*group_by_attr*})

Circle Graph [207, 93]

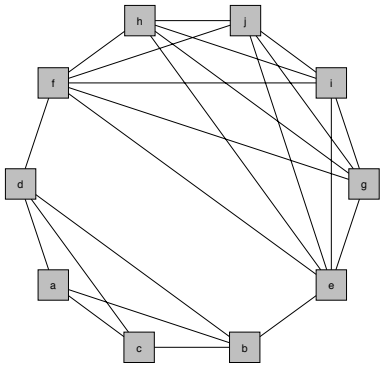


Figure 80: GLOv2 Circle Graph seed technique from [207].

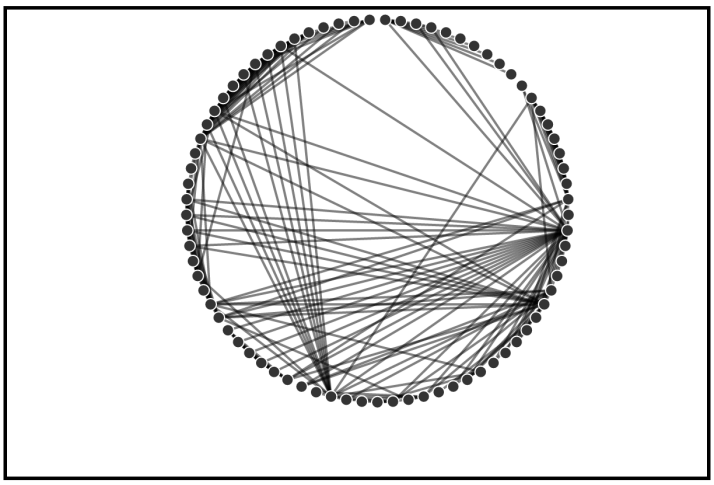


Figure 81: Circle graph [207] rendered in GLO.js.

The Circle Graph seed technique consists of constant-sized, constant-colored square nodes connected by constant-sized, constant-colored straight-line edges. The nodes

are positioned such that the nodes are in an evenly-distributed circle around the center of the canvas.

I have abstracted away the textual labels on the nodes.

- #sort_attr: cluster
- display nodes as squares
- show all edges
- display edges as straight lines
- color edges by constant
- color nodes by constant
- size edges by constant
- size nodes by constant
- evenly distribute nodes on θ (sort-by:{*sort_attr*})
- position nodes by constant on ρ

GeneVis A [23]

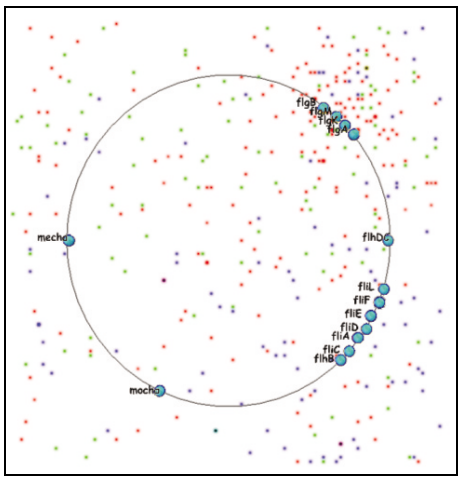


Figure 82: GLOv2 GeneVis A seed technique from [23].

The GeneVis A seed technique consists of constantly-sized, constantly-colored circle nodes positioned at a constant radius from the center of the plot at degrees relative to an attribute of the data. Edges are hidden.

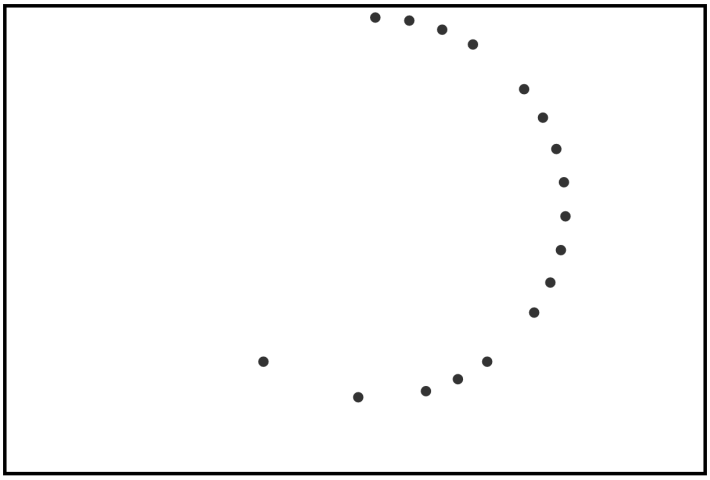


Figure 83: Genevis A [23] rendered in GLO.js.

I have abstracted away the specific color choices, the textual labels aside each node, and the overarching circle behind the plot.

- #position_attr: degree
- display nodes as circles
- hide edges
- position nodes on θ by $\{position\_attr\}$
- position nodes by constant on ρ
- size nodes by constant
- color nodes by constant

GeneVis B [23]

The GeneVis B seed technique consists of constant-size, constant-colored circle nodes positioned along the y axis by a discrete attribute and relatively along the x axis by an attribute. These nodes are connected by constant-sized curved edges colored by a gradient based on an attribute of the source and target nodes.

I have abstracted away the overarching ring glyphs, the textual labels accompanying each node glyph, and the specific color choices.

- #color_attr: cluster

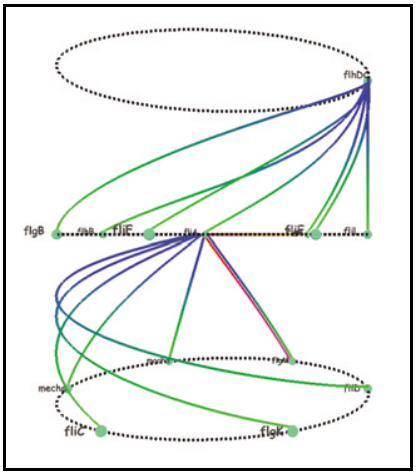


Figure 84: GLOv2 GeneVis B seed technique from [23].

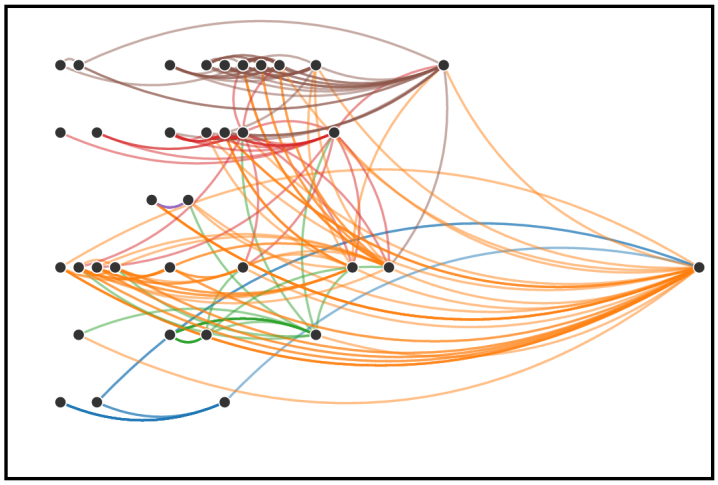


Figure 85: Approximate Genevis B [23] rendered in GLO.js.

- #discrete: cluster
- #attr: degree
- display nodes as circles
- size nodes by constant
- color nodes by constant
- show all edges
- display edges as curved lines
- size edges by constant
- color edges by $\{source.color\_attr\} \rightarrow \{target.color\_attr\}$

- position nodes on y by $\{discrete\}$
- position nodes on x by $\{attr\}$

Arc Diagram [243, 141]

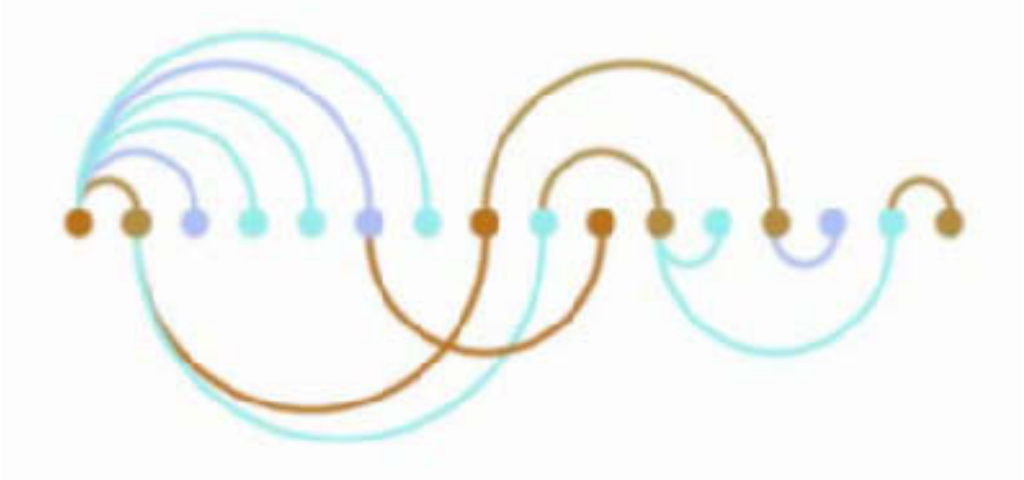


Figure 86: GLOv2 Arc Diagram seed technique from [141]. (Specifically the 'contributor coloring' subfigure.)

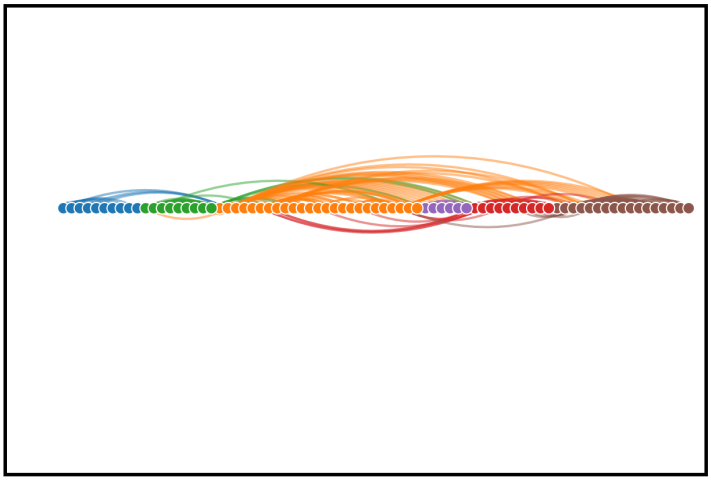


Figure 87: Arc diagram [141] rendered in GLO.js.

The Arc Diagram seed technique consists of constant-sized circle nodes colored by an attribute evenly distributed along the middle of the y axis. Edges are displayed as constant-sized curved edges colored the same as their target nodes.

I have abstracted away the specific color choices.

- #node_color_attr: cluster
- #sort_attr: cluster
- display nodes as circles
- size nodes by constant
- color nodes by {*node_color_attr*}
- align nodes {*middle*}
- display edges as curved lines
- size edges by constant
- color edges by {*target.node_color_attr*}
- show all edges
- evenly distribute nodes on x (sort-by:{*sort_attr*})

Matrix Browser [262]

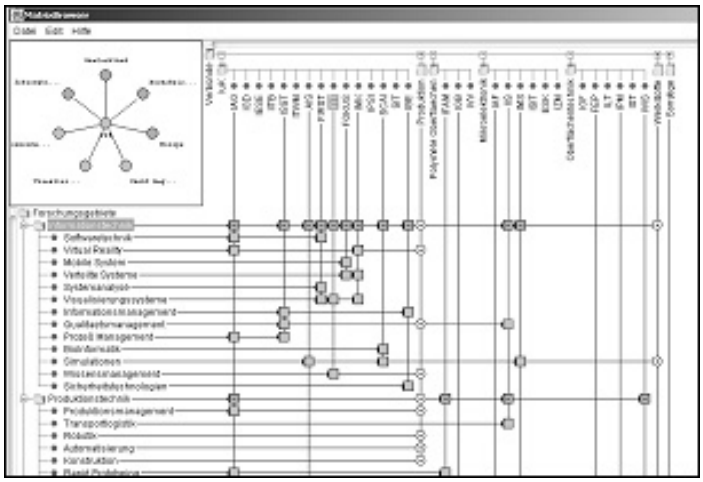


Figure 88: GLOv2 Matrix Browser seed technique from [262].

The Matrix Browser seed technique has nodes displayed as constant-sized, constant-colored labels evenly spaced along the top and left, with edges displayed as constant-sized squares colored by an attribute positioned at the y value of the source node on the left and x value of the the target node on the top. The display also has edges displayed as constant-sized, constant-colored right-angle lines from the source node

on the left to the target node on the top. On the left, the nodes are sorted top to bottom, while on top the nodes are sorted left to right.

I have abstracted away the interaction for collapsing hierarchies and the high-level node-link diagram.

- #label_attr: name
- #sort_attr: cluster
- #edge_color_attr: weight

  #Left column of nodes
- display nodes as {*label_attr*} labels
- color nodes by constant
- size nodes by constant
- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true)
- align nodes {*left*}

  #Top row of nodes
- clone nodes
- rotate nodes {*90*}
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*})

  #Square edges
- set target generation {*1*}
- display edges as squares
- position edges by {*target.x*},{*source.y*}
- show all edges
- size edges by constant
- color edges by {*edge_color_attr*}

  #Right angle edges
- clone edges

- display edges as right angles
- color edges by constant

Matrix with Bars [205]

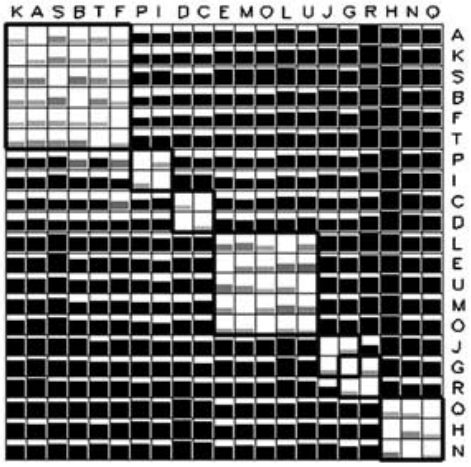


Figure 89: GLOv2 Matrix with Bars seed technique from [205].

The Matrix with Bars seed technique has nodes displayed as constant-sized, constant-colored labels evenly spaced along the top and right, with edges displayed as relatively-sized, relatively-colored rectangles positioned at the x value of the source node on the top and y value of the the target node on the right.

To align with the other matrix-based representations, the nodes are sorted top to bottom, while on top the nodes are sorted left to right and the labels are rotated. (Note that nodes are not sorted or rotated in the seed technique figure from [205].)

- #label_attr: name
- #sort_attr: cluster
- #edge_size_attr: weight
- #Right column of nodes
- display nodes as {*label_attr*} labels
- color nodes by constant
- size nodes by constant

- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true)
- align nodes {*right*}
- #Top row of nodes
- clone nodes
- rotate nodes {*90*}
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*})
- #Edges
- set target generation {*1*}
- display edges as bars
- position edges by {*source.x*},{*target.y*}
- show all edges
- size edges by {*edge_size_attr*}

MatrixExplorer [116]

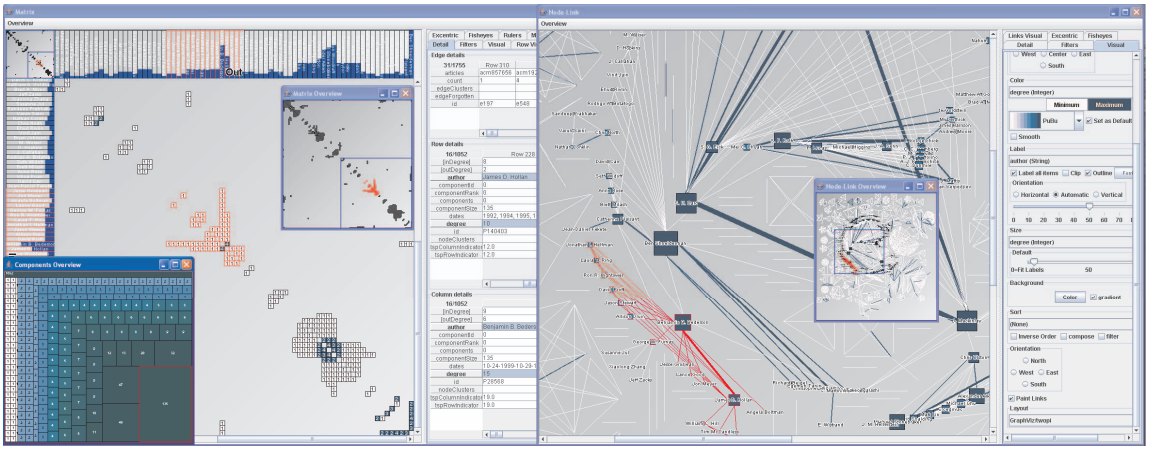


Figure 90: GLOv2 MatrixExplorer seed technique from [116].

The MatrixExplorer seed technique consists of two canvases. The first canvas has nodes displayed as constant-sized, constant-colored labels evenly spaced along the top and left, with edges displayed as constant-sized squares colored by an attribute

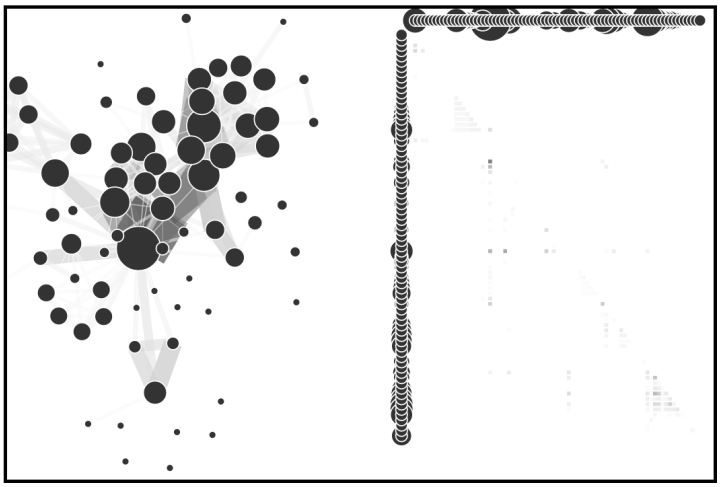


Figure 91: Approximate MatrixExplorer [116] rendered in GLO.js.

positioned at the y value of the source node on the left and x value of the the target node on the top. On the left, the nodes are sorted top to bottom, while on top the nodes are sorted left to right. Behind the node labels on the left, nodes are represented as bars sized by the out-degree (an attribute) of the nodes. Behind the node labels on the top, nodes are represented as bars sized by the in-degree (an attribute) of the nodes. The second canvas consists of relatively-sized, square nodes colored by an attribute connected by relatively sized, relatively colored straight line edges. In addition, a second set of nodes in the same positions are displayed as constant-sized, constant colored labels.

I have abstracted away the overview and details panes, the specific color scheme, and the dynamic query widgets.

- #sort_attr: cluster
- #node_size_attr: degree
- #label_attr: name
- #edge_attr: weight
- #Property constants
- color edges by {*edge_attr*}

- show all edges
- partition canvas on x
- #Matrix left column bars and labels
- color nodes by constant
- display nodes as bars
- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true)
- align nodes {*left*}
- size nodes by {*out_degree*}
- clone nodes
- size nodes by constant
- display nodes as {*label_attr*} labels
- #Matrix top row bars and labels
- select node generation {*1*} #bars
- clone nodes
- size nodes by {*in_degree*}
- rotate nodes {*90*}
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*})
- clone nodes
- size nodes by constant
- display nodes as {*label_attr*} labels
- #Matrix edges
- size edges by constant
- set target generation {*4*} #top row labels
- display edges as squares
- position edges by {*target.x*},{*source.y*}

  #F-D Diagram

- select canvas {*0*}
- display nodes as squares
- size nodes by {*node_size_attr*}
- display edges as straight lines
- size edges by {*edge_attr*}
- apply force-directed algorithm to nodes

  #F-D Diagram labels
- clone nodes
- display nodes as {*label_attr*} labels
- size nodes by constant

NetLens [140]

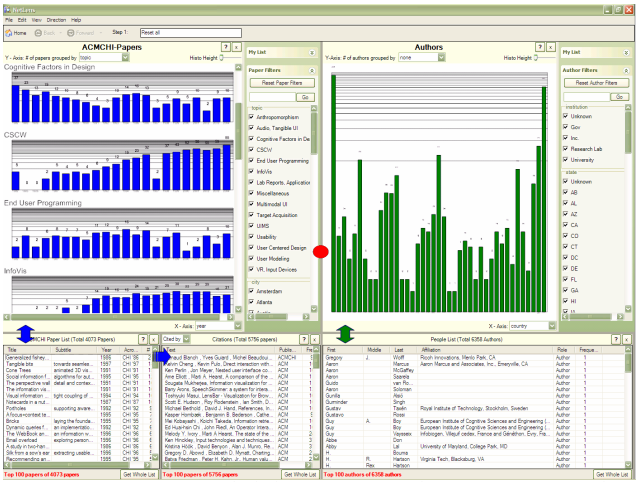


Figure 92: GLOv2 NetLens seed technique from [140].

In the NetLens seed technique, there are multiple canvases. The display is split into two sides. On one side the nodes are aggregated by a discrete attribute, displayed as constant-colored bars, aligned along the bottom, and sized by an attribute. On the other side, the nodes are filter-partitioned by a discrete attribute and then aggregated by a different discrete attribute and displayed as constant-colored, relatively-sized bars aligned along the bottoms of the canvases. A meta-axis is displayed on the y

axis. Edges are not displayed.

I have abstracted away the list panels, dynamic query widgets, and specific color scheme.

- #right_aggr_attr: cluster
- #right_sort_attr: cluster
- #partition_attr: gender
- #left_aggr_attr: cluster
- display nodes as bars
- color nodes by constant
- align nodes {*bottom*}
- hide edges
- #Single panel side
- partition canvas on x
- aggregate nodes by {*right_aggr_attr*} by {*method*}
- evenly distribute nodes on x (sort-by:{*right_sort_attr*})
- size nodes by {*count*}
- #Multi-panel side
- select canvas {*0*}
- filter partition canvas on y by {*partition_attr*}
- show meta y axis
- select column {*0*} #all multi-panel-side partitions
- aggregate nodes by {*left_aggr_attr*} by {*method*}
- size nodes by {*count*}

Semantic Substrates [204, 18]

The Semantic Substrates seed technique is simplified from the figure. In the

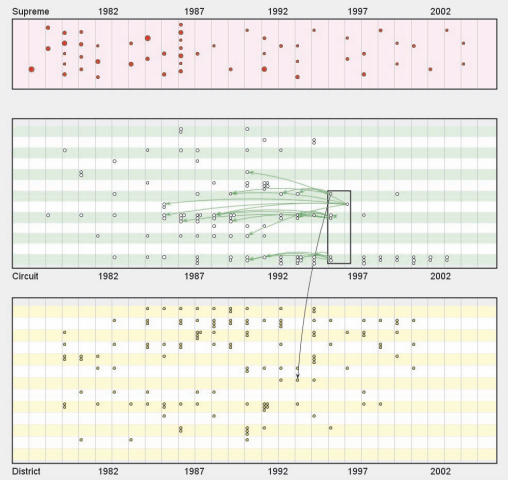


Figure 93: GLOv2 Semantic Substrates seed technique from [204].

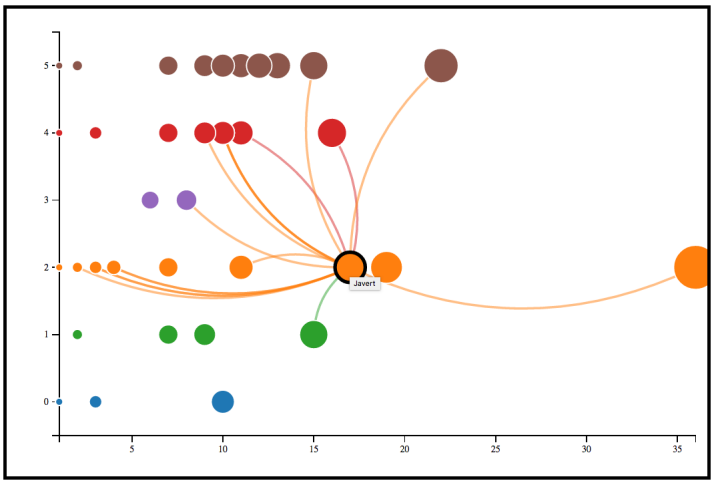


Figure 94: Semantic Substrates [204] rendered in GLO.js.

semantic substrates seed technique, nodes are relatively sized and colored by an attribute. Nodes are separated on the y axis by a discrete attribute and then positioned relatively on the x axis. Edges are only shown when the analyst mouses over an endpoint node, when they are displayed as constant-sized curved lines colored by an attribute of the target node.

I have abstracted away the color fills behind the nodes, the multi-node selection, and the specific color scheme.

- #node_color_attr: cluster

- #node_size_attr: degree
- #node_y_attr: cluster
- #node_x_attr: degree
- display nodes as circles
- color nodes by {*node_color_attr*}
- show incident edges
- size nodes by {*node_size_attr*}
- position nodes on y by {*node_y_attr*}
- show y axis
- position nodes on x by {*node_x_attr*}
- show x axis
- display edges as curved lines
- size edges by constant
- color edges by {*target.node_color_attr*}

PivotGraph [244]

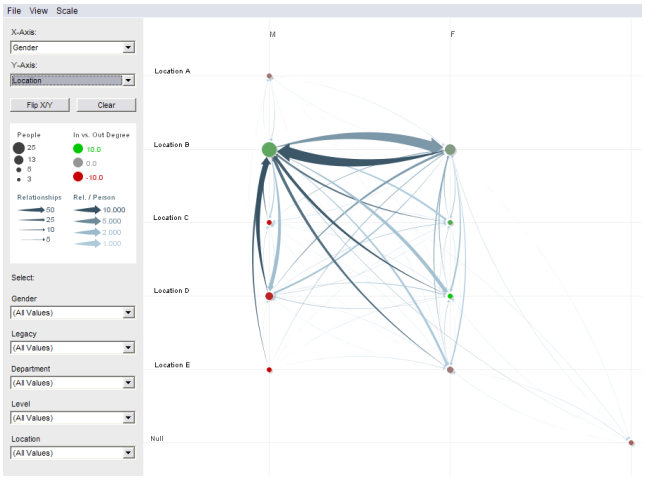


Figure 95: GLOv2 PivotGraph seed technique from [244].

The PivotGraph seed technique consists of aggregated super-nodes positioned on x and y by discrete attributes forming a grid. The super-nodes are sized by the

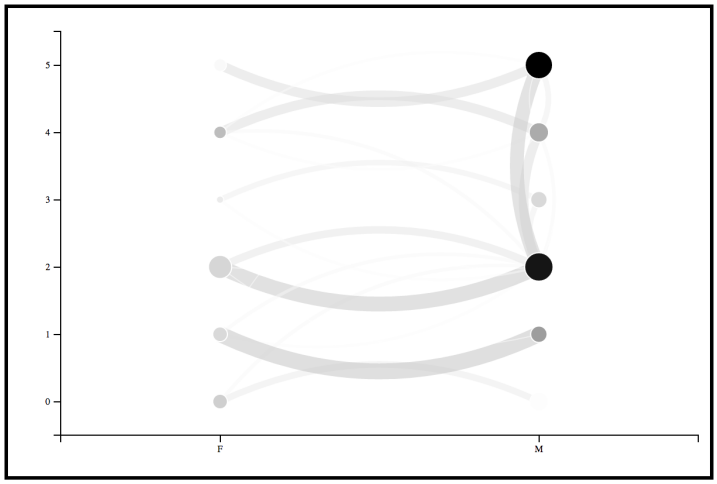


Figure 96: PivotGraph [244] rendered in GLO.js.

number of nodes represented by the super-nodes and colored by the degree of the super-node. Edges are displayed as curved lines, sized and colored by an attribute.

I have abstracted away the grid lines, dynamic query widgets, specific color scheme, and specific glyph rendering style.

- #disc1: gender
- #disc2: cluster
- #method: mean
- #node_color_attr: in_degree
- #edge_attr: count
- display nodes as circles
- show all edges
- aggregate nodes by [{*disc1*},{*disc2*}] by {*method*}
- size nodes by {*count*}
- color nodes by {*node_color_attr*}
- aggregate edges by [{*source.disc1*},{*source.disc2*},{*target.disc1*},{*target.disc2*}] by {*method*}
- display edges as curved lines

- size edges by {*count*}
- color edges by {*edge_attr*}
- position nodes on x by {*disc1*}
- position nodes on y by {*disc2*}
- show x axis
- show y axis

MatLink [118]

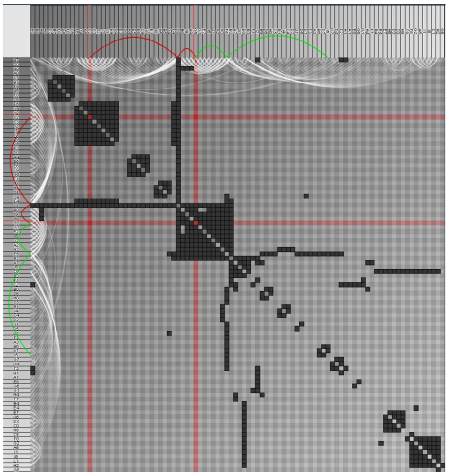


Figure 97: GLOv2 MatLink seed technique from [118].

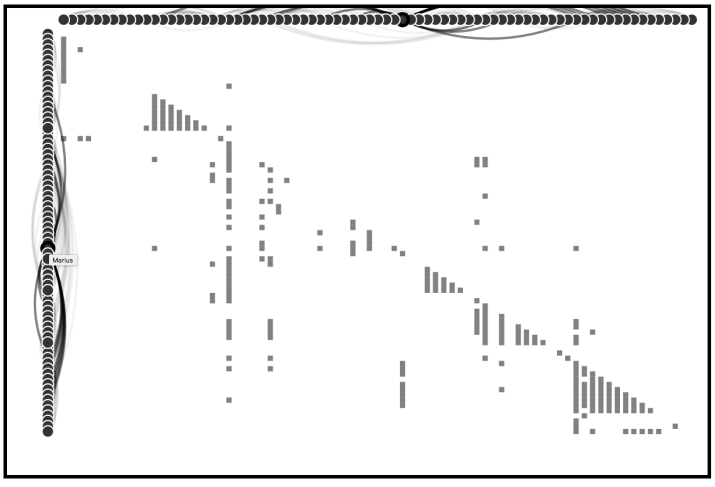


Figure 98: Approximate MatLink [118] rendered in GLO.js.

The MatLink seed technique has nodes displayed as constant-sized, constant-colored labels evenly spaced along the top and left, with edges displayed as constant-sized, constant-colored squares positioned at the y value of the source node on the left and x value of the the target node on the top. On the left, the nodes are sorted top to bottom, while on top the nodes are sorted left to right. Constant-sized and -colored curved edges are also shown faded between the nodes on each axis, and shown fully rendered when the analyst interacts with an endpoint.

I have abstracted away the specific color scheme, the background grid, and multi-selection.

- #label_attr: name
- #sort_attr: cluster

  #Constants amongst all glyphs
- color nodes by constant
- size nodes by constant
- size edges by constant
- color edges by constant

  #Left column nodes and curved edges
- display nodes as {*label_attr*} labels
- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true)
- align nodes {*left*}
- display edges as curved lines
- show faded and incident edges

  #Top row nodes and curved edges
- clone nodes
- clone edges
- set source generation {*1*} #eventual top row
- set target generation {*1*} #eventual top row

- rotate nodes {*90*}
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*})

  #Square edges
- clone edges
- set source generation {*0*}
- display edges as squares
- position edges by {*target.x*},{*source.y*}
- show all edges

List View [199, 214]

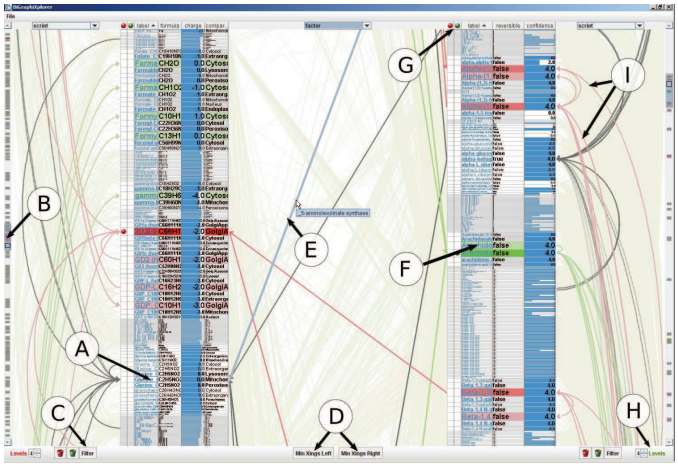


Figure 99: GLOv2 List View seed technique from [199].

The List View seed technique consists of nodes displayed as constantly-sized labels in vertically stacked lists by an attribute and aligned at the bottom of the display. Edges are constant-sized, colored by an attribute, and are displayed as faded unless the analyst interacts with an endpoint, when they are fully rendered. Edges are displayed as curved lines within a list and straight lines between lists.

I have abstracted away the specific color scheme, the fisheye display on the nodes,

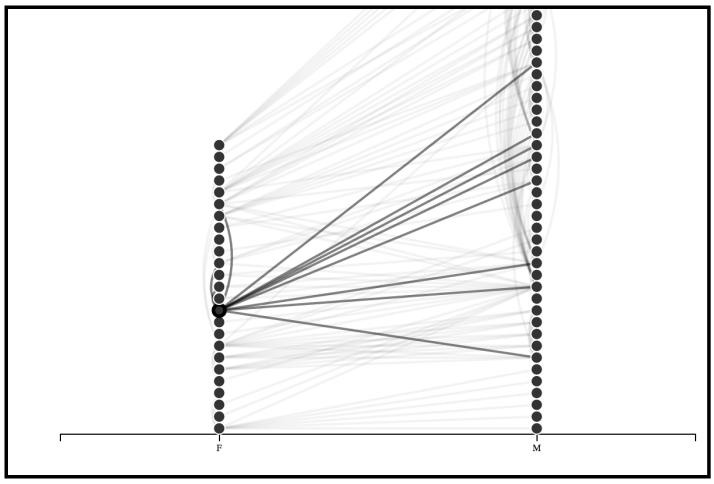


Figure 100: List view [199] rendered in GLO.js.

multiple-selection, the additional information columns, and the dynamic query widgets.

- #label_attr: name
- #discrete: gender
- #sort_attr: cluster
- display nodes as {*label_attr*} labels
- size nodes by constant
- color nodes by constant
- size edges by constant
- color edges by constant
- position nodes on x by {*discrete*}
- position nodes evenly stacked {*bottom*} (sort-by:{*sort_attr*}, within:{*discrete*})
- show x axis
- display edges as straight lines

  #Straight intra-cluster edges
- show faded and incident edges
- hide edges (group-by:{*discrete*})

#Curved inter-cluster edges

- clone edges
- display edges as curved lines
- hide edges
- show faded and incident edges (group-by:{*discrete*})

Edge-Label-Centric [182]

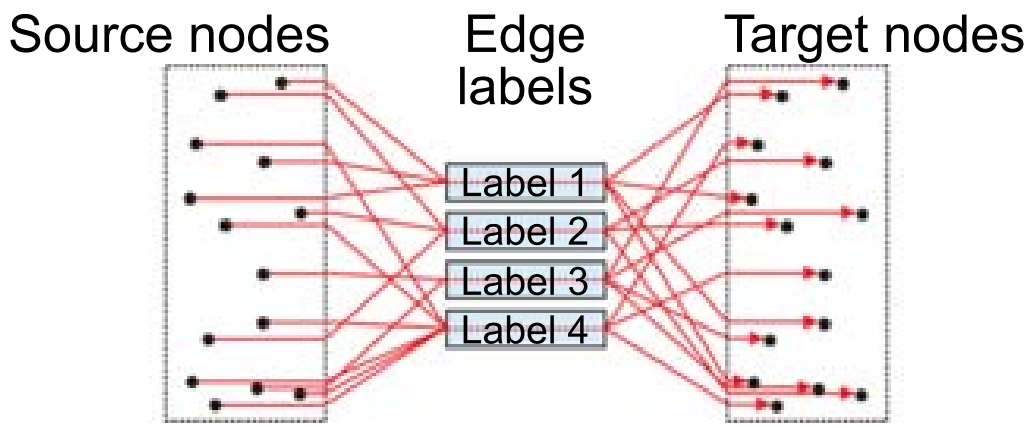


Figure 101: GLOv2 Edge-Label-Centric seed technique from [182].

In the Edge-Label-Centric seed technique, two sets of nodes are displayed as constant-sized, constant-colored circles evenly distributed along the y axis. One set is aligned on the left of the display and the other on the right. One set of edges are aggregated by a discrete property, displayed as constant-sized labels, aligned in the center of the display, and distributed along the y axis. A second set of edges are displayed as constant-sized, constant-colored straight lines from the source node in the left-hand set through the appropriate edge label in the center to the target node in the right-hand set.

I have abstracted away the specific color scheme.

- #node_sort_attr: cluster
- #edge_aggr_attr: target.gender

- #method: mean
- #edge_sort_attr: id

  #Left column of nodes
- display nodes as circles
- color nodes by constant
- size nodes by constant
- evenly distribute nodes on y (sort-by:{*node_sort_attr*})
- align nodes {*left*}

  #Right column of nodes
- clone nodes
- align nodes {*right*}

  #All edges
- color edges by constant
- size edges by constant
- clone edges

  #Center edge labels
- aggregate edges by {*edge_aggr_attr*} by {*method*}
- display edges as {*edge_label_attr*} labels
- align edges {*center*}
- evenly distribute edges on y (sort-by:{*edge_sort_attr*})

  #Line edges
- select edge generation {*0*} #unaggregated edges
- set edge waypoint generation {*1*} #edge label generation
- display edges as straight lines
- set target generation {*1*} #right column nodes

Honeycomb [106]

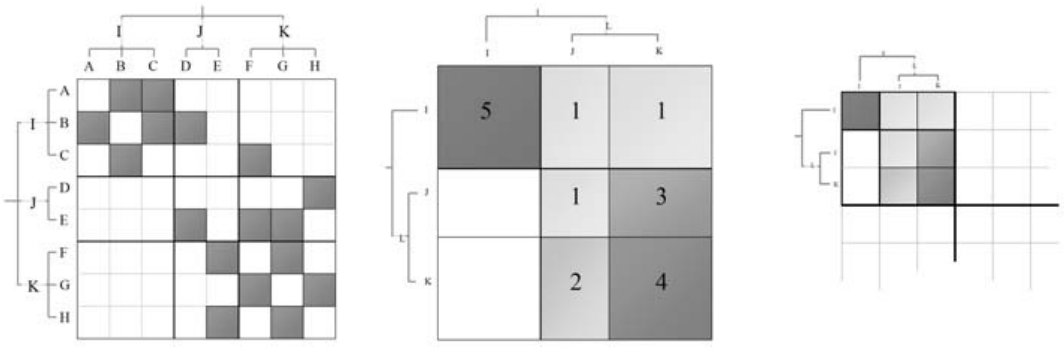


Figure 102: GLOv2 Honeycomb seed technique from [106].

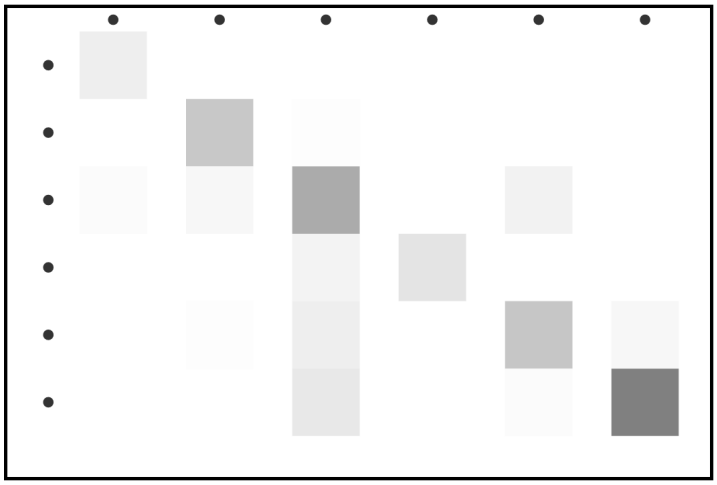


Figure 103: Approximate Honeycomb [106] rendered in GLO.js.

The Honeycomb seed technique consists of two sets of aggregated super-nodes displayed as constant-sized, constant-colored labels distributed along the left and top. The super-nodes along the left are sorted top to bottom, while the super-nodes on the top are sorted left to right. Aggregated super-edges are displayed as constant-sized squares colored by an attribute positioned at the y value of the source node on the left and x value of the the target node on the top.

I have abstracted away the hierarchical dendrograms and have rotated the top node labels to be consistent with the other matrix-based techniques.

- #disc: cluster
- #method: mean

- #sort_attr: cluster
- #edge_color_attr: count

  #Left column of nodes
- size nodes by constant
- aggregate nodes by {*disc*} by {*method*}
- display nodes as {*disc*} labels
- align nodes {*left*}
- evenly distribute nodes on y (sort-by:{*sort_attr*}, invert:true) #Top row of nodes
- clone nodes
- align nodes {*top*}
- evenly distribute nodes on x (sort-by:{*sort_attr*}) #Edges
- show all edges
- set target generation {*4*} #top super-node glyphs
- aggregate edges by [{*source.disc*},{*target.disc*}] by {*method*}
- display edges as squares
- position edges by {*target.x*},{*source.y*}
- color edges by {*edge_color_attr*}
- size edges by constant

GraphDice Segment [36]

The seed technique consists of circular nodes sized by an attribute and colored by a constant. The nodes are positioned relatively along the x and y axes by attributes. Scale axes are drawn for the x and y axes. Edges are drawn as constantly-sized, constantly-colored curved lines.

I have abstracted away the multi-selection interaction (and accompanying convex hull highlighting and non-constant colors) as well as the specific color scheme.

- #attr1: betweenness centrality

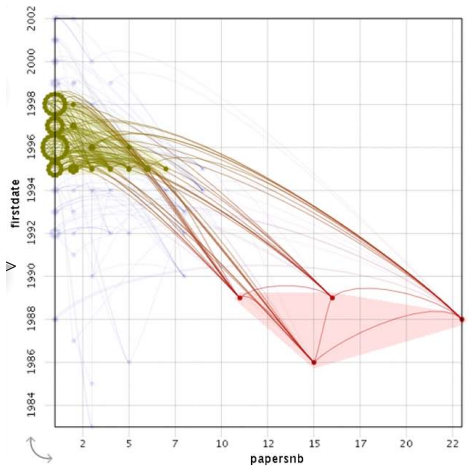


Figure 104: GLOv2 GraphDice Segment seed technique from [36].

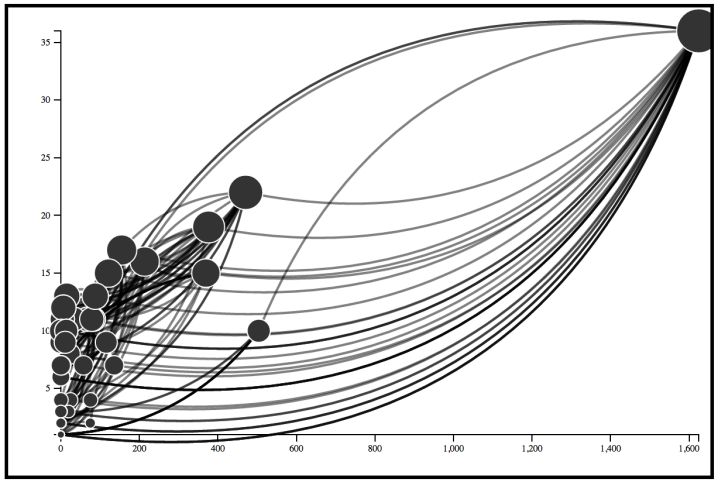


Figure 105: GraphDice segment [36] rendered in GLO.js.

- #attr2: degree
- #node_size_attr: degree
- display nodes as circles
- position nodes on x by {*attr1*}
- position nodes on y by {*attr2*}
- size nodes by {*node_size_attr*}
- color nodes by constant
- display edges as curved lines
- show all edges

- size edges by constant
- color edges by constant
- show x axis
- show y axis

3x3 GraphDice [36]

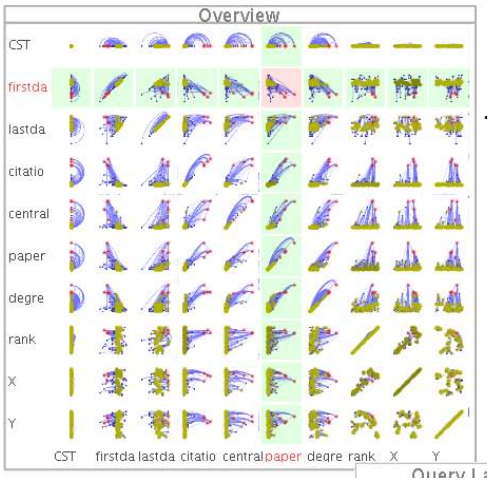


Figure 106: GLOv2 GraphDice seed technique from [36].

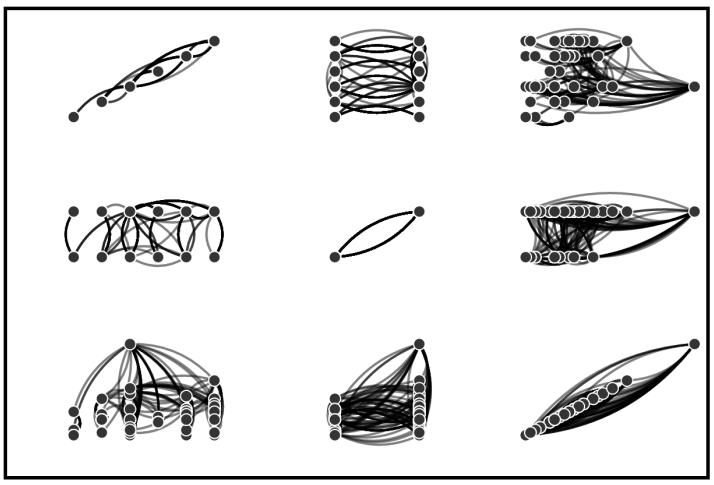


Figure 107: Approximate 3x3 GraphDice [36] rendered in GLO.js.

The 3x3 GraphDice seed technique consists of a 3x3 grid of canvases. In each canvas, circular nodes are sized by an attribute and colored by a constant. The nodes

in each canvas are positioned relatively along the x and y axes by attributes. Edges are drawn as constantly-sized, constantly-colored curved lines. The nodes in each column are positioned relatively along the x axis by the same attribute. The nodes in each row are positioned relatively along the y axis by the same attribute. The attributes are the same on both axes and ordered the same from top to bottom and from left to right.

I have abstracted away the multi-selection interaction (which resulted in the multiple colors). While they are a critical component of the GraphDice technique, I have abstracted away the meta-axis labels. Because the columns and rows represent attributes, rather than data elements, the choice of columns and rows cannot be done elegantly using the expected data model. This is also why I chose an explicitly 3x3 GraphDice grid.

- #attr1: cluster
- #attr2: gender
- #attr3: degree
- #Set up a single cell
- display nodes as circles
- show all edges
- color nodes by {*node_color_attr*}
- size nodes by {*node_size_attr*}
- size edges by constant
- color edges by constant
- display edges as curved lines
- position nodes on x by {*attr1*}
- position nodes on y by {*attr1*}

  #Set up first row (i.e. set each column's x attribute)
- partition canvas on x (parts:3)

- select canvas {*1*}
- position nodes on x by {*attr2*}
- select canvas {*2*}
- position nodes on x by {*attr3*}

  #Create grid and set each row's y attributes
- partition canvas on y (parts:3, all-canvases:true)
- select row {*1*}
- position nodes on y by {*attr2*}
- select row {*2*}
- position nodes on y by {*attr3*}

GMap [94]

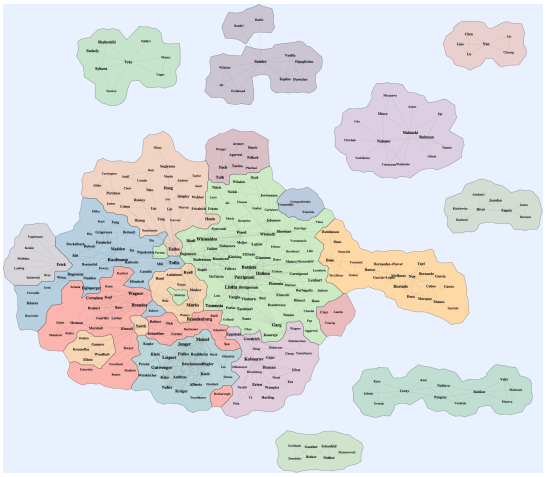


Figure 108: GLOv2 GMap seed technique from [94].

The GMap seed technique consists of nodes displayed as constant-colored, relatively-sized labels connected by faded, constantly-sized straight-line edges. Surrounding each cluster of nodes is a translucent convex hull colored by an attribute.

I abstracted away the blue background and the specific color scheme.

- #label_attr: name
- #node_size_attr: degree

- #group_by_attr: cluster
- display nodes as {*label_attr*} labels
- size nodes by {*node_size_attr*}
- color nodes by constant
- display edges as straight lines
- color edges by constant
- size edges by constant
- show edges as faded
- show convex hulls (group-by:{*group_by_attr*})
- color convex hulls by {*group_by_attr*}

Attribute Matrix [153]

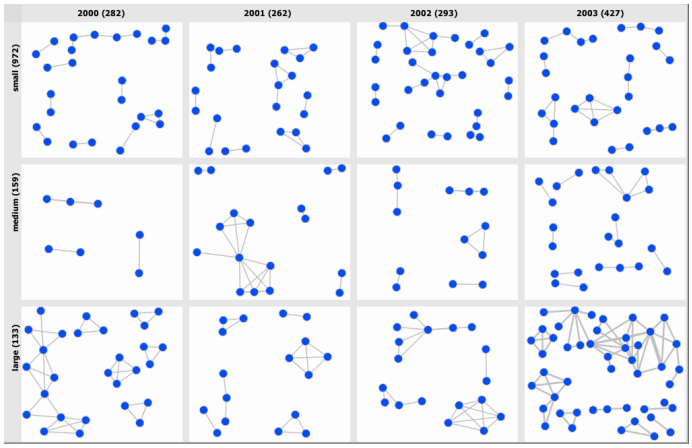


Figure 109: GLOv2 Attribute Matrix seed technique from [153].

The Attribute Matrix seed technique consists of a grid of canvases. The x and y meta-axes are of discrete attributes and the nodes in each cell are those that share those attributes and the edges are those in the subgraph induced by the nodes. Within each cell, the nodes are displayed as constant-sized, constant-colored circles connected by constant-sized, constant-colored straight line edges.

I have abstracted away the specific blue-node color scheme.

- #x_axis_disc: cluster
- #y_axis_disc: gender
- display nodes as circles
- size nodes by constant
- color nodes by constant
- display edges as straight lines
- size edges by constant
- color edges by constant
- filter partition canvas on x by $\{x\_axis\_disc\}$
- filter partition canvas on y by $\{y\_axis\_disc\}$ (all-canvases:true) #form the grid
- apply force-directed algorithm to nodes (all-canvases:true)

EdgeMap A [74]

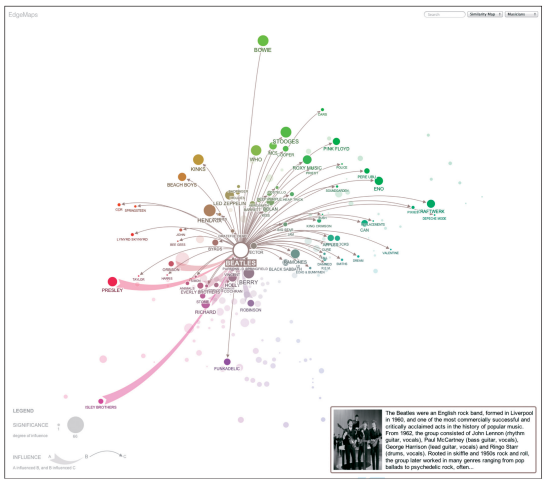


Figure 110: GLOv2 EdgeMap A seed technique from [74].

The EdgeMap A seed technique consists of circular nodes colored and sized relatively by attributes. The nodes are positioned using a force-directed algorithm. Edges are hidden, except when the analyst interacts with an endpoint, in which case the edge is drawn as a curved line colored the same as the source node. In-edges and out-edges of the interaction node are drawn differently. Neighbors of the interaction

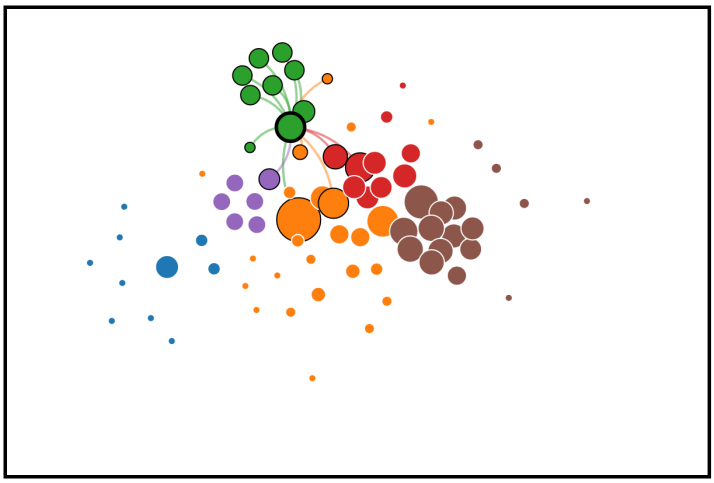


Figure 111: Approximate EdgeMap A [74] rendered in GLO.js.

node are highlighted.

I have abstracted away the specific rendering styles of the edges beyond curved lines, leaving it up to the implementation to choose how to differentiate the in- and out-edges. I have also abstracted away the specific color scheme, the details-on-demand panel, and the dynamic query widgets.

- #node_size_attr: degree
- #node_color_attr: cluster
- display nodes as circles
- size nodes by {*node_size_attr*}
- color nodes by {*node_color_attr*}
- display edges as curved lines
- size edges by constant
- color edges by {*source.node_color_attr*}
- show in-out edges
- highlight neighbors
- apply force-directed algorithm to nodes

EdgeMap B [74]

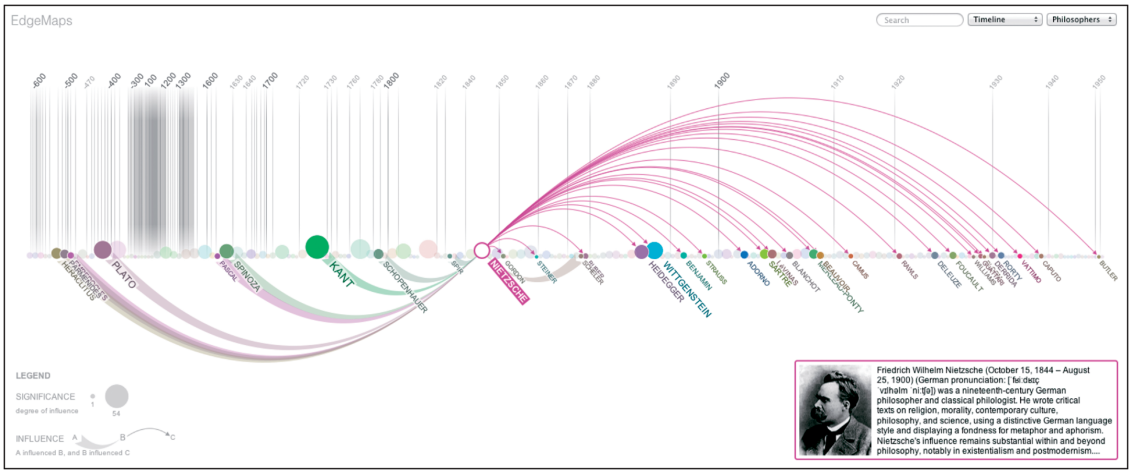


Figure 112: GLOv2 EdgeMap B seed technique from [74].

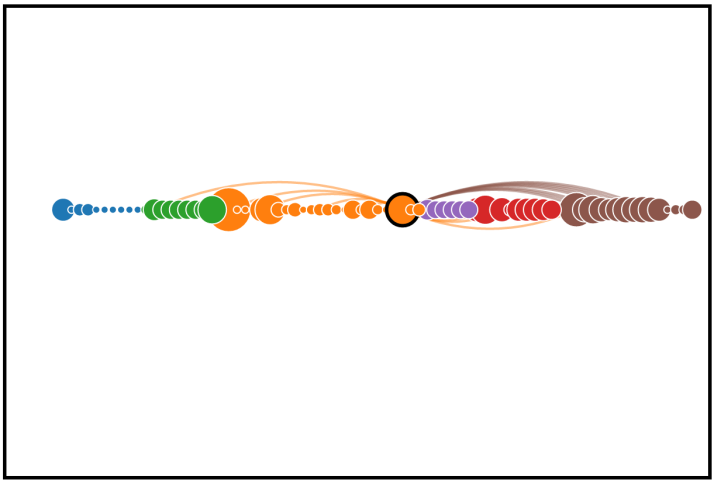


Figure 113: Approximate EdgeMap B [74] rendered in GLO.js.

The EdgeMap B seed technique consists of circular nodes colored and sized relatively by attributes. The nodes are evenly distributed along the middle of the y axis sorted by an attribute. Edges are hidden, except when the analyst interacts with an endpoint, in which case the edge is drawn as a curved line colored the same as the source node. In-edges and out-edges of the interaction node are drawn differently. Neighbors of the interaction node are highlighted.

I have abstracted away the specific rendering styles of the edges beyond curved lines, leaving it up to the implementation to choose how to differentiate the in- and

out-edges. I have also abstracted away the specific color scheme, the details-on-demand panel, and the dynamic query widgets.

- #node_size_attr: degree
- #node_color_attr: cluster
- #x_pos_attr: cluster
- display nodes as circles
- size nodes by {*node_size_attr*}
- color nodes by {*node_color_attr*}
- display edges as curved lines
- size edges by constant
- color edges by {*source.node_color_attr*}
- show in-out edges
- highlight neighbors
- align nodes {*middle*}
- evenly distribute nodes on x (sort-by:{*x_pos_attr*})

Hive Plot [144]

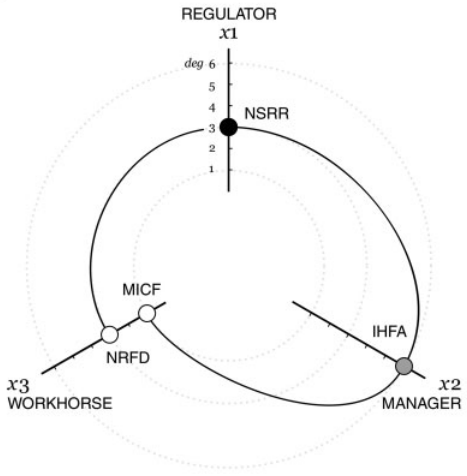


Figure 114: GLOv2 Hive Plot seed technique from [144].

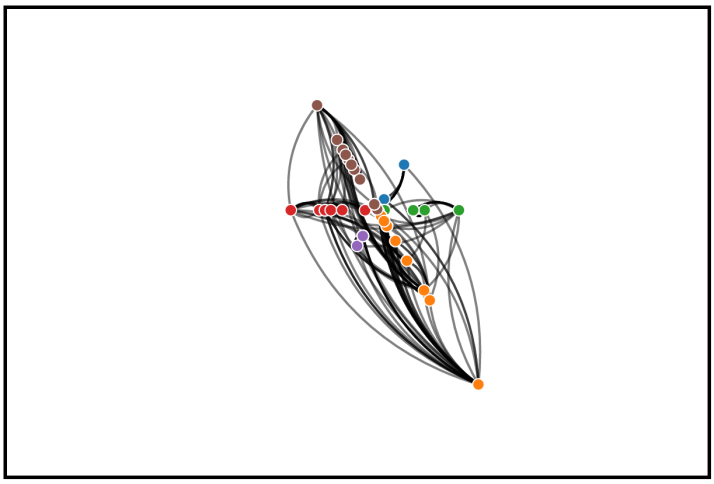


Figure 115: Hive Plot [144] rendered in GLO.js.

In the Hive Plot seed technique, constant-sized circular nodes colored by an attribute are distributed around the center of the display by a discrete attribute. The nodes are positioned at a distance from the center of the display relatively by an attribute. Edges are drawn as constant-colored, constant-sized curved lines.

I have abstracted away the radial axis labels and the specific (greyscale) color scheme.

- #discrete: cluster
- #attr: degree
- display nodes as circles
- size nodes by constant
- size edges by constant
- color nodes by $\{discrete\}$
- position nodes on θ by $\{discrete\}$
- position nodes on ρ by $\{attr\}$
- display edges as curved lines
- color edges by constant
- show all edges

2x3 Hive Panel [144]

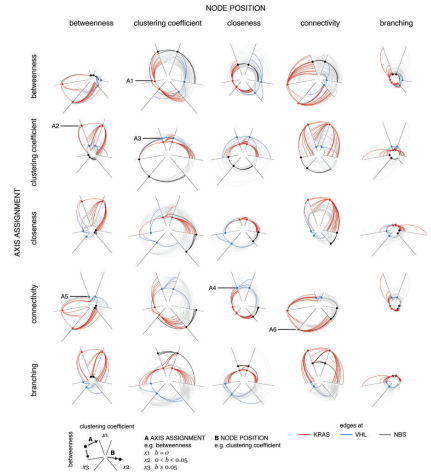


Figure 116: GLOv2 Hive Panel seed technique from [144].

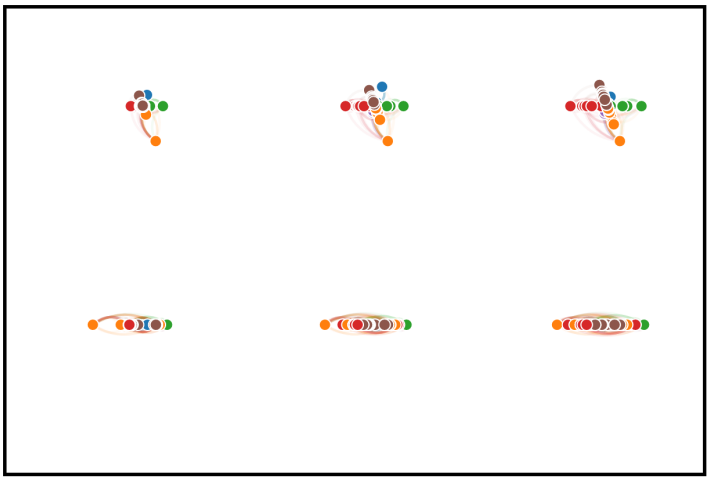


Figure 117: 2x3 Hive Panel [144] rendered in GLO.js.

The 2x3 Hive Panel seed technique consists of a 2x3 grid of canvases. In each canvas, constant-sized circular nodes colored by an attribute are distributed around the center of the display by a discrete attribute and positioned at a distance from the center of the display relatively by an attribute. The canvases in each row have the same attribute driving the angular position, while the canvases in each column have the same attribute driving the radial position. Edges are drawn as faded, constant-sized curved lines colored the same as their source nodes unless the analyst interacts

with an endpoint, then the edge is drawn fully rendered.

As with the 3x3 GraphDice seed technique, I have abstracted away the meta-axis labels and chose the specific 3x2 grid-size. I have also abstracted away the duplicate axes.

- #node_color_attr: cluster
- #discrete1: cluster
- #discrete2: gender
- #attr1: betweenness centrality
- #attr2: pagerank
- #attr3: degree
- display nodes as circles
- size nodes by constant
- size edges by constant
- color nodes by {*node_color_attr*}
- display edges as curved lines
- show faded and incident edges
- position nodes on θ by {*discrete1*}
- position nodes on ρ by {*attr1*}
- partition canvas on x (parts:3)
- select canvas {*1*}
- position nodes on ρ by {*attr2*}
- select canvas {*2*}
- position nodes on ρ by {*attr3*}
- partition canvas on y (parts:2, all-canvases:true)
- select row {*1*} #0-indexed
- position nodes on θ by {*discrete2*}
- color edges by {*source.node_color_attr*} (all-canvases:true)

ScatterNet [27]

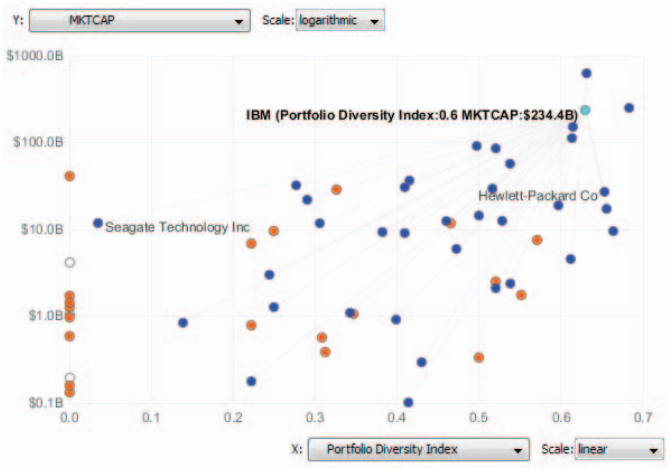


Figure 118: GLOv2 ScatterNet seed technique from [27].

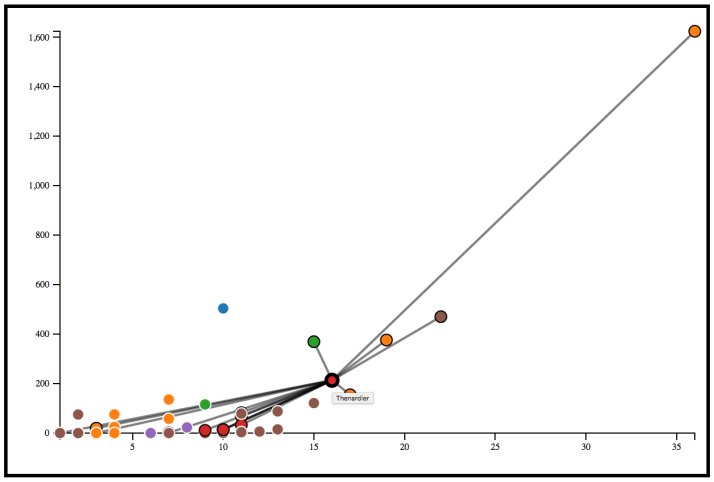


Figure 119: ScatterNet [27] rendered in GLO.js.

In the ScatterNet seed technique, nodes are displayed as constant-sized circles colored by an attribute. Nodes are positioned on the x and y axes relatively by attributes and scale axes for the attributes are displayed on the x and y axes. Edges are hidden, except when the analyst interacts with an endpoint, in which case the edge is drawn as a constant-colored, constant-sized straight line. Neighbors of the interaction node are highlighted.

I have abstracted away the alternative axis types (e.g. logarithmic), the few node

labels, and the specific color scheme.

- #node_color_attr: cluster
- #attr1: degree
- #attr2: betweenness centrality
- display nodes as circles
- size nodes by constant
- color nodes by {*node_color_attr*}
- display edges as straight lines
- size edges by constant
- color edges by constant
- position nodes on x by {*attr1*}
- position nodes on y by {*attr2*}
- show x axis
- show y axis
- show incident edges
- highlight neighbors

Citevis [213]

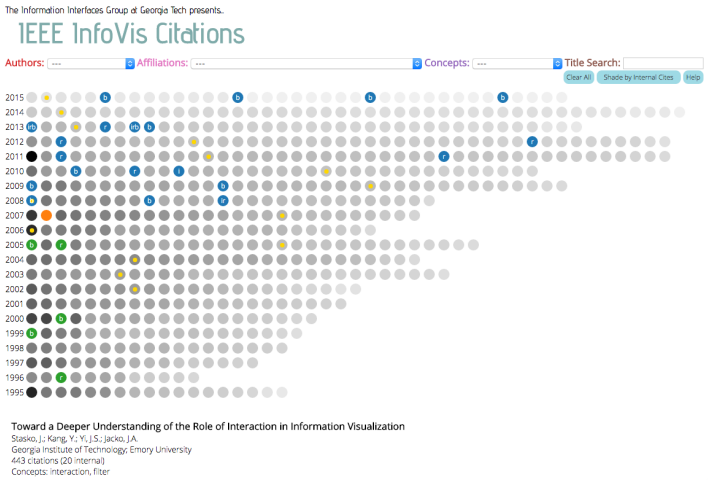


Figure 120: GLOv2 Citevis seed technique based on [213].

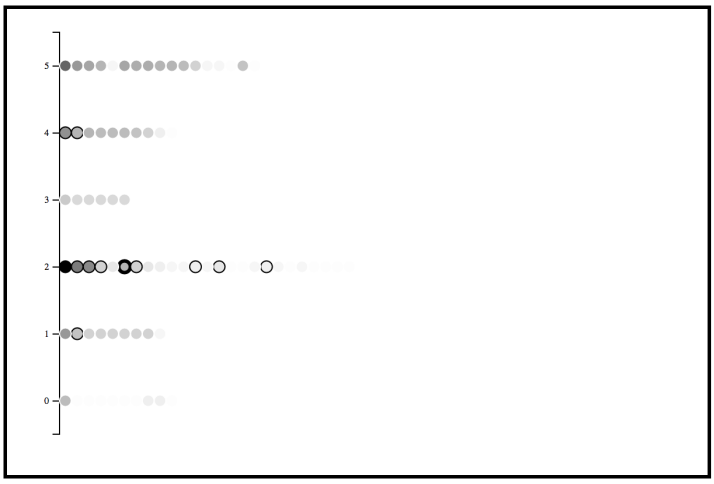


Figure 121: Citevis [213] rendered in GLO.js.

In the Citevis seed technique, nodes are distributed along the y axis by a discrete attribute. Within each row, the constant-sized, relatively-colored circular nodes are stacked to the left. When the analyst interacts with a node, the neighbors of that node highlight, with the in-neighbors and out-neighbors highlighting differently. Edges are hidden.

I have abstracted away the specific means by which the in- and out-neighbors are highlighted, the details-on-demand panel, and the dynamic query widgets.

- #discrete: cluster
- #color_attr: degree
- #sort_attr: betweenness centrality
- hide edges
- display nodes as circles
- highlight in-out neighbors
- size nodes by constant
- position nodes on y by $\{discrete\}$
- show y axis

- position nodes evenly stacked {*left*} (sort-by:{*sort_attr*}, within:{*discrete*}, invert:true)
- color nodes by {*color_attr*}

DOSA [232]

Figure 122: GLOv2 DOSA seed technique from [232].

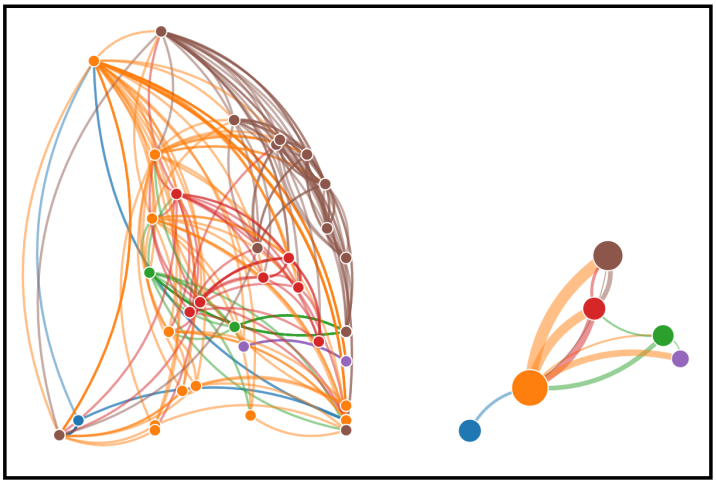


Figure 123: Approximate DOSA [232] rendered in GLO.js.

The DOSA seed technique consists of two canvases. In the first canvas, constant-sized circular nodes colored by an attribute are positioned relatively along the x and y axes by attributes. Edges are drawn as constant-sized curved lines colored by a gradient from the source node's color to the target node's color. In the second

canvas, the nodes in the first canvas are aggregated into super-nodes, displayed as constant-sized squares colored by the same attribute as the first canvas's nodes and positioned at the average position of the nodes aggregated. Super-edges between the super-nodes are displayed as relatively-sized curved lines colored by a gradient from the source super-node's color to the target super-node's color.

I have abstracted away the multi-selection interaction (instead, the effect is of every node being a member of a selection based on a discrete attribute). I have also abstracted away the embedded visualizations replacing the node glyphs in the aggregated display.

- #discrete: cluster
- #attr1: clustering coefficient
- #attr2: number of triangles
- #method: mean
- #Unaggregated canvas
- display nodes as circles
- color nodes by {*discrete*}
- size nodes by constant
- display edges as curved lines
- size edges by constant
- color edges by {*source.discrete*}→{*target.discrete*}
- position nodes on x by {*attr1*}
- position nodes on y by {*attr2*}
- #Aggregated canvas
- partition canvas on x
- aggregate nodes by {*discrete*} by {*method*}
- display nodes as squares
- aggregate edges by [{*source.discrete*},{*target.discrete*}] by {*method*}

- size nodes by {*count*}
- size edges by {*count*}

NodeTrix [117]

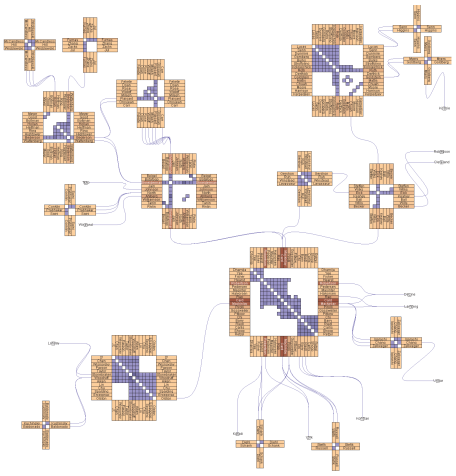


Figure 124: GLOv2 NodeTrix seed technique from [117].

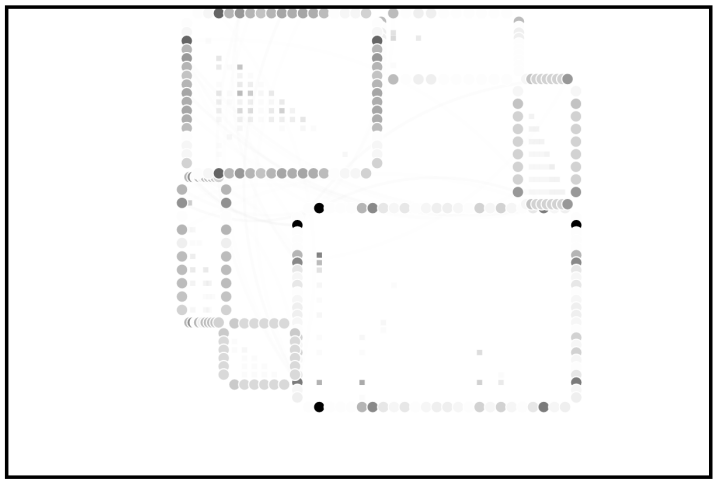


Figure 125: Approximate NodeTrix [117] rendered in GLO.js.

The NodeTrix seed technique consists of four sets of constant-sized, relatively-colored label nodes arranged in squares. Each square represents a cluster. Within each square, the first set of nodes is aligned to the left of the square and distributed along the y axis sorted from top to bottom. The second set is aligned to the right

right of the square and distributed along the y axis sorted from top to bottom. The third set is aligned along the top of the square, rotated 90 degrees, and ordered from left to right. The fourth set is aligned along the bottom of the square, rotated 90 degrees, and ordered from left to right. Edges within each square are displayed as constant-sized, relatively colored squares positioned at the y position of the source node in the first set of nodes and the x position of the target node in the third set of nodes. Edges between nodes in different squares are displayed as relatively-colored, constant-sized curved lines.

I have abstracted away the specific color scheme, as well as the fact that intra-cluster edges are drawn from the closest source glyph in one cluster to the closest target glyph in the second cluster.

- #label_attr: name
- #node_color_attr: degree
- #edge_color_attr: weight
- #discrete: cluster
- #Visual properties
- display nodes as {*label_attr*} labels
- color nodes by {*node_color_attr*}
- size nodes by constant
- color edges by {*edge_color_attr*}
- size edges by constant
- #Position nodes in a large circle
- position nodes by constant on ρ
- evenly distribute nodes on θ (sort-by:{*discrete*})
- #Left columns within clusters
- align nodes {*left*} (group-by:{*discrete*})
- evenly distribute nodes on y (group-by:{*discrete*}, invert:true)

- #Right columns within clusters
- clone nodes
- align nodes {*right*} (group-by:{*discrete*})
- #Top rows within clusters
- clone nodes
- rotate nodes {*90*}
- evenly distribute nodes on x (group-by:{*discrete*})
- align nodes {*top*} (group-by:{*discrete*})
- #Bottom rows within clusters
- clone nodes
- align nodes {*bottom*} (group-by:{*discrete*})
- #Edges
- set target generation {*2*} #Top row
- #Intra-cluster edges as curved lines
- show all edges
- display edges as curved lines
- #Inter-cluster edges as squares
- display edges as squares (group-by:{*discrete*})

APPENDIX C

GLOV1 OPERATIONS SET

Positioning Nodes The operations in this category each adjust the two-dimensional (x,y) coordinate positions of node glyphs.

- **Align Nodes {*Left, Center, Right, Top, Middle, Bottom*}**: adjusts the position of the nodes by changing the appropriate coordinate values of all nodes to a constant value.
- **Evenly Distribute Nodes on x or y by {*attribute*}**: disperses the nodes horizontally or vertically so that the nodes are evenly distributed on the appropriate axis, sorted by the provided attribute of the nodes.
- **Evenly Distribute Nodes on x or y**: disperses the nodes horizontally or vertically so that the nodes are evenly distributed on the appropriate axis, defaulting to the nodes' ordering in the data store.
- **Substrate Nodes on x or y by {*categorical attribute*}**: positions the nodes based on a categorical attribute value. Attribute values are assigned to locations evenly across the appropriate axis and each node is then positioned at its value's location.
- **Evenly Distribute Nodes within Substrates**: positions the nodes of the most recently applied substrate evenly along the opposite axis of the substrate axis.
- **Position Nodes on x or y Relatively by {*continuous attribute*}**: positions each node based on a continuous attribute. The left-most or bottom-most

position is assigned a zero value and the right-most or top-most position is assigned the maximum value amongst the nodes. Nodes are then positioned along the axis using a linear scale of their attribute values.

- **Evenly Distribute Nodes Radially by {*attribute*}**: position the nodes evenly around the center of the plot clockwise from the top, sorted by the attribute of the node.

- **Evenly Distribute Nodes Radially**: position the nodes evenly around the center of the plot clockwise from the top, defaulting to the nodes' ordering in the data store.

- **Position Nodes Radially by {*continuous attribute*}**: positions each node radially based on a continuous attribute. The top-most position is assigned a zero value and the position just left of the top value is assigned the maximum value amongst the nodes. Nodes are then positioned clockwise-radially using a linear scale of their attribute values.

- **Substrate Nodes Radially by {*categorical attribute*}**: positions the nodes based on a categorical attribute value. Attribute values are assigned to locations evenly around the center of the plot and each node is then positioned at its value's location.

- **Evenly Distribute Nodes Along Plot Radius by {*attribute*}**: disperses the nodes so that the nodes are evenly distributed in distance from the center of the plot to the edge of the plot, sorted from the center by the attribute of the node.

- **Evenly Distribute Nodes Along Plot Radius**: disperses the nodes so that the nodes are evenly distributed in distance from the center of the plot to the

edge of the plot, sorted from the center by the attribute of the node, defaulting to the nodes' ordering in the data store.

- **Position Nodes Along Plot Radius by {*continuous attribute*}**: positions each node based on a continuous attribute. The inner-most position is assigned a zero value and outer-most position is assigned the maximum value amongst the nodes. Nodes are then positioned from the inner-most position to the outer-most using a linear scale of their attribute values.
- **Substrate Nodes Along Plot Radius by {*categorical attribute*}**: positions the nodes based on a categorical attribute value. Attribute values are assigned to locations evenly along the radius of the plot and each node is then positioned at its value's location.
- **Position Nodes Along Plot Radius by Constant**: Positions the nodes a fixed distance from the center of the plot.
- **Apply {*algorithm*} to the Nodes**: positions the nodes using a physics-based algorithm, such as a force-directed algorithm.

Modifying Element Properties The GLOs in this category each modify the non-spacial visual properties of the node and edge glyphs.

- **Size Nodes by Constant**: adjusts the radius of each node to a constant value.
- **Size Nodes Relatively by {*continuous attribute*}**: adjusts the radius of each node using a linear scale between zero and the maximum value amongst the nodes.
- **Display All Links**: makes all edges visible.
- **Display Selected Links**: makes all edges invisible. When the user mouses over a node, makes the in- and out-edges of that node visible.

- **Hide Links**: makes edges invisible.
- **Display Links as Straight**: adjusts each edge to be drawn as a straight line from the center of the source node to the center of the target node.
- **Display Links as Curved**: draws each edge as a quadratic curve clockwise from the source node to the target node.
- **Display Links as Circles**: adjusts each edge to be drawn as a circle with y coordinate of its source node and x coordinate of its target node.

Cloning Nodes This category of GLOs allows for duplicating node glyphs and interacting with the various sets of duplicates. Each cloning operation creates a new *generation* of nodes, and each generation is assigned an identifying *generation number* so that the generation can be referenced by other operations. The initial set of nodes are assigned generation number 0. After that, the first clone (or aggregate) generation created is assigned generation number 1, the second 2, and so on. The *active generation* is the generation of nodes on which GLOs are applied. For example, if an *evenly distribute nodes on* x GLO is applied, only the nodes in the active generation are repositioned.

- **Clone Active Generation**: generates copies of all of the node glyphs of the current generation. The copies have the same visual properties of the cloned generation. The new generation is assigned a generation number for reference and becomes the active generation.
- **Select Generation** k: select a generation of nodes and makes it the active generation. Subsequent GLOs are applied to this generation.
- **Set Source Generation** k: adjust edges to be drawn from generation k.
- **Set Target Generation** k: adjust edges to be drawn to generation k.

- **Remove Generation** k: Removes the glyphs of generation k from the display. If edges were being drawn to or from this generation, they are instead drawn to or from generation 0 (the initial nodes). If generation k was the active generation, generation 0 becomes the active generation.

Aggregating Nodes and Edges This category of GLOs enable the creation of glyphs that represent more than a single node or edge. As with cloning GLOs, aggregation creates new *generations* of nodes and assigns them *generation numbers* for reference.

- **Aggregate by {*categorical attribute*}**: aggregates nodes with the same attribute into supernodes and aggregates edges into superedges between the supernodes. The original nodes and edges are discarded. The radius of the supernodes and width of the superedges are determined relatively by the number of nodes or edges the supernode or superedge represents. These supernodes and superedges are assigned a generation number in order to reference them and are set as the active generation as described in [244].

- **Aggregate by {*categorical attribute*} and {*categorical attribute*}**: as above, but aggregates nodes where the values of both attributes are the same.

- **Deaggregate Generation** k: deaggregates the supernodes and superedges of the kth generation back into the original nodes and edges. The original nodes retain their original sizes, but are positioned at their respective supernodes' most recent positions.

Modifying Display Properties The operations in this category do not modify the elements of the graph (the node and edge glyphs) but instead modify the display itself.

- **Show x or y Axis**: displays labels on the appropriate axis based on the currently applied positioning GLO. These labels are updated as new positioning GLOs are applied.

- **Hide x or y Axis**: hides the labels on the appropriate axis.

APPENDIX D

GLOV2 OPERATIONS SET

Please see Section 4.4 for properties of the GLO domain-specific language and for descriptions of the optional parameters of each operation: group-by, within, sort-by, invert, all-generations, and all-canvases.

Node and Edge Positioning The operations in this category reposition either the node or edge glyphs within a canvas.

- **Align nodes {*dir*}**: Position node glyphs at a constant value along an axis. Valid directions are up, down, left, right, center (x), and middle (y).
- **Evenly distribute nodes on {*axis*}**: Evenly disperse the node glyphs along the provided axis. If a sort-by parameter is provided, the nodes are sorted by the sort-by attribute. If an invert parameter is provided, the nodes are sorted in reverse order.
- **Position nodes on {*axis*} by constant**: Position node glyphs at a pre-defined constant value along the provide axis.
- **Position nodes on {*axis*} by {*attr*}**: Position node glyphs relatively along the axis by the provided attribute. If the attribute is a discrete attribute, distribute the values along the axis. If an invert parameter is provided, the axis is flipped.
- **Position nodes evenly stacked on {*dir*}**: Position the node glyphs at $axis_0 + i * default_stack_distance$, where i is the index in the sorted list. (A suggested default stack distance is the default node size.) Valid directions are

up, down, left, and right. The direction determines the value of $axis_0$. If a sort-by parameter is provided, the nodes are sorted by the sort-by attribute. If an invert parameter is provided, the nodes are sorted in reverse order.

- **Position nodes stacked on {*axis*} by {*attr*}**: If the provided attribute is continuous, given the sum of each values of the attribute ($s = \Sigma v_n$) and the total length of the axis (l), the position of the glyph of the largest valued node is $p_0 = axis_0$. Position each subsequent node's glyph at $p_{n-1} + (v_{n-1}/s) * l$. If the provided attribute is discrete, equivalent to *Position nodes evenly stacked on* {dir}. If an invert parameter is provided, the axis is flipped.

- **Apply force-directed algorithm to nodes**: Use a force-directed layout algorithm to position the node glyphs on the canvas. If the nodes are super-nodes, edges are assumed to be the super-edges induced by the nodes.

- **Align edges {*dir*}**: Position edge glyphs at a constant value along an axis. Valid directions are up, down, left, right, center (x), and middle (y). Operation has no effect unless edges are displayed as text, squares, or bars.

- **Evenly distribute edges on {*axis*}**: Evenly disperses the edge glyphs along the provided axis, sorted by the optional by attribute. Operation has no effect unless edges are displayed as text, squares, or bars.

- **Position edges by {*attr*},{*attr*}**: Position the edge glyphs such that the x coordinate is based on the first attribute and the y coordinate based on the second attribute. Operation has no effect unless edges are displayed as text, squares, or bars.

Node and Edge Visual Properties The operations in this category adjust how the individual glyphs are displayed. A given node or edge glyph has a single display

mode, single color, and single size. Node glyphs have a single rotation. Edge glyphs have a single waypoint or no waypoints.

- **Display nodes as circles**: Display node glyphs as circles.
- **Display nodes as squares**: Display node glyphs as squares.
- **Display nodes as {*attr*} labels**: Display each node glyph as text with the string value of the node's attribute value.
- **Display nodes as bars**: Display node glyphs as rectangles.
- **Display edges as straight lines**: Display edges as straight lines between the source and target nodes' glyphs in the edge generation's source and target generations, respectively.
- **Display edges as curved lines**: Display edges as curved lines between the source and target nodes' glyphs in the edge generation's source and target generations, respectively. Edges curve counter-clockwise, unless an invert flag is passed and then the edges should curve clockwise.
- **Display edges as squares**: Display edges as squares.
- **Display edges as {*attr*} labels**: Display edges as text of the edges' attribute values.
- **Display edges as bars**: Display edges as rectangles.
- **Display edges as right angles**: Display edges as right angles between the source and target nodes' glyphs in the edge generation's source and target generations, respectively. Edges are drawn counter-clockwise, unless an invert flag is passed and then the edges should curve clockwise.

- **Size nodes by constant**: Size node glyphs by a implementation-determined constant.
- **Size nodes by {*attr*}**: Size node glyphs relatively by the provided attribute.
- **Size edges by constant**: Size edge glyphs by an implementation-determined constant.
- **Size edges by {*attr*}**: Size edge glyphs relatively by the provided attribute.
- **Color nodes by constant**: Color node glyphs the implementation-default node color.
- **Color nodes by {*attr*}**: Color node glyphs relatively by the provided color attribute.
- **Color edges by constant**: Color edge glyphs by the implementation-default edge color.
- **Color edges by {*attr*}**: Color edges relatively by the provided color attribute.
- **Color edges by {*attr*}→{*attr*}**: Color edges using a gradient between the provided color attributes.
- **Rotate nodes {*num*} degrees**: Rotate the node glyphs from their current rotation the specified number of degrees counter-clockwise. This operation is additive.
- **Unrotate nodes**: Return the node glyphs to their default rotation.
- **Set edge waypoint edge generation {*num*}**: Edges route through the glyph in the provided edge generation associated with the same backing edge. (Operation has no visible effect when edges are displayed as text, squares, or bars.)

- **Remove all edge waypoints**: Edges no longer route through a waypoint generation, if they did.

Convex Hulls The operations in this category manipulate translucent convex hulls around node glyphs.

- **Show convex hulls**: Draw a translucent convex hull around node glyphs in the default convex hull color.
- **Hide convex hulls**: Hide convex hulls.
- **Color convex hulls by {*attr*}**: Color convex hulls relatively by the provided attribute.
- **Color convex hulls by constant**: Color convex hulls the default convex hull color.

Interaction The operations in this category manipulate if and/or how to render glyphs differently when the analyst interacts with a node. (It is left to the implementation what constitutes an interaction.) Only a single interaction mode (including no interaction) will be active for any given edge or node glyph.

- **Show all edges**: Fully render edge glyphs, regardless of interaction.
- **Hide edges**: Do not render edge glyphs, regardless of interaction.
- **Show edges as faded**: Render edge glyphs with a lower color saturation, regardless of interaction.
- **Show incident edges**: Do not render edge glyphs, except to fully render edge glyphs when the analyst interacts with the backing edge's end-point's node glyph. When rendering edge glyphs, fully render the edge glyph.

- **Show in-out edges**: Do not render edge glyphs, except to fully render edge glyphs when the analyst interacts with the backing edge's end-point's node glyph. When rendering edge glyphs, fully render the edge glyph. Glyphs representing in-edges and out-edges are rendered differently.

- **Show faded and incident edges**: Render edge glyphs with a lower color saturation, except to fully render edge glyphs when the analyst interacts with the backing edge's end-point's node glyph.

- **Highlight neighbors**: Render node glyphs differently if the backing node is a neighbor of the backing node of the glyph which the analyst is interacting.

- **Highlight in-out neighbors**: Render node glyphs differently if the backing node is a neighbor of the backing node of the glyph which the analyst is interacting. Further, render glyphs representing in-nodes and out-nodes differently.

- **Stop highlight neighbors**: Render all node glyphs the same, regardless of interaction.

Aggregation The operations in this category are for aggregating and deaggregating node and edge generations. The {*discrete*} attribute in the two aggregation GLOs can be an array of discrete attributes. When an array is used, nodes or edges are aggregated if they have the same values for all of the attributes in the array.

- **Aggregate nodes by {*discrete*} using {*method*}**: Create a new generation of node glyphs, one node glyph for each value of the discrete attribute with the display and interaction modes of the generation being aggregated. The new node glyphs are backed by super-nodes, which have attributes based upon the aggregated nodes: discrete attribute values are chosen from the most common values of the aggregated nodes, while continuous attributes are determined using

the provided method. (Allowed methods include sum, mean, mode, minimum, and maximum.) The new nodes are positioned at the mean x and y values of the aggregated nodes. Hides the original node generation. Sets the new node generation as the active generation.

- **Aggregate edges by {*discrete*} using {*method*}**: Create a new generation of edge glyphs, one edge glyph for each value of the discrete attribute with display and interaction modes and source and target generations of the generation being aggregated. The new node glyphs are backed by super-edges, which have attributes based upon the aggregated edges: discrete attribute values are chosen from the most common values of the aggregated nodes, while continuous attributes are determined using the provided method. (Allowed methods include sum, mean, mode, minimum, and maximum.) Hides the original edge generation. Sets the new edge generation as the active generation.

- **Deaggregate nodes**: If the node generation is backed by super-nodes, remove the node generation and unhides the original node generation from which the aggregate generation was created. The glyphs in the original generation have the display and interaction modes and positions of the removed aggregate generation.

- **Deaggregate edges**: If the node generation is backed by super-edges, remove the edge generation and unhides the original edge generation from which the aggregate generation was created. The glyphs in the original generation have the display and interaction modes and source and target generations of the removed aggregate generation.

Cloning The operations in this category relate to creating and manipulating generations of node and edge glyphs. Group-by and within optional parameters are

ignored if passed to any operations in this category. A node generation counter and edge generation counter are used to keep track of the generations.

- **Clone nodes**: Create a new generation of node glyphs with the same display mode, interaction mode, convex hull(s), and color and size properties as the generation to which the operation is applied. The generation is given the next index based on the generation counter.

- **Clone edges**: Create a new generation of edge glyphs with the same display mode, interaction mode, waypoints, and color and size properties as the generation to which the operation is applied. The generation is given the next index based on the generation counter.

- **Select node generation {*num*}**: Set the given node generation as the active node generation.

- **Select edge generation {*num*}**: Set the given node generation as the active edge generation.

- **Set source generation {*num*}**: Edge glyphs will be drawn with the node glyphs in the given node generation as their source.

- **Set target generation {*num*}**: Edge glyphs will be drawn with the node glyphs in the given node generation as their target.

- **Remove node generation {*num*}**: Removes the specified node generation's glyphs. If the node generation was the only node generation in its canvas, a null state node generation is created and assigned an index of 0 if there is only a single canvas or the next index if there are multiple canvases. If the node generation was a source (or target) generation for any edge generations, that edge generation is assigned the lowest-indexed node generation it the same canvas as the source (or target) generation.

- **Remove edge generation {*num*}**: Removes the specified edge generation's glyphs. If the edge generation was the only edge generation in its canvas, a null state edge generation is created and assigned an index of 0 if there is only a single canvas or the next index if there are multiple canvases.

- **Remove all cloned nodes**: Removes all but the lowest-indexed node generation in the canvas. Remaining node generation is assigned an index of 0 if there is only a single canvas or the next index if there are multiple canvases. All edge generations are assigned the remaining node generation as their source and target generation.

- **Remove all cloned edges**: Removes all but the lowest-indexed edge generation in the canvas. Remaining edge generation is assigned an index of 0 if there is only a single canvas or the next index if there are multiple canvases.

Partitioning The operations in this category relate to creating and manipulating canvases within the GLO Display. A canvas counter is used to keep track of the canvases.

- **Partition canvas on {*axis*} (by {*num*})**: Divide the current (original) canvas into $\{num\}$ evenly-sized canvases along the provided axis (x or y). Clones all node and edge generations in the original canvas into the new canvases. Sets the clones of the former active node and edge generations in the last canvas as the active node and edge generations and the last canvas as the active canvas.

- **Filter partition canvas on {*axis*} by {*discrete*}**: Divide the current (original) canvas into k evenly-sized canvases along the provided axis (x or y), where k is the number of different values of the provided discrete attribute. Clone all node and edge generations in the original canvas into the new canvases, however the cloned node generation in each canvas only includes the node glyphs for the

nodes with a single value for the discrete attribute and the edge generation only includes edge glyphs for the edges in the subgraph induced by the nodes with that value. Sets the clones of the former active node and edge generations in the last canvas as the active node and edge generations and the last canvas as the active canvas.

- **Select canvas {*num*}**: Set the specified canvas as the active canvas.
- **Select row {*num*}**: Set the canvases in the specified row (0-indexed from top to bottom) as active canvases.
- **Select column {*num*}**: Set the canvases in the specified column (0-indexed from left to right) as active canvases.
- **Remove canvas {*num*}**: Remove the specified canvas and merges it (if possible) with the canvas to the left (if there is no canvas to the left, then canvases to the right, up, and down are tried in order). All node and edge generations in the removed canvas are moved (rather than cloned) into the merged canvas. If the removed canvas is the only active canvas, the merged canvas is set as the active canvas. The active node and edge generations of the merged canvas remain the prior active node and edge generations of that canvas.
- **Remove all partitions**: Merge all canvases into the top-left canvas. All node and edge generations in the removed canvases are moved (rather than cloned) into the merged canvas. The merged canvas is set as the active canvas. The active node and edge generations of the merged canvas remain the prior active node and edge generations of that canvas. If the removed canvases were created through filter-partitioning, the node generations are merged and the edge generations are merged to create single node and edge generations.

Axes The operations in this category are for showing, hiding, and manipulating canvases' axis labels and the GLO Display's meta-axis labels.

- **Show {*axis*} axis**: If the provided axis's labels are not currently visible, display the axis labels for the provided axis on the canvas.
- **Hide {*axis*} axis**: If the provided axis's labels are currently visible, hide the axis labels for the provided axis on the canvas.
- **Set {*axis*} axis node generation {*num*}** The specified node generation will be used to determine the values for the specified axis's labels.
- **Show meta {*axis*} axis**: If the provided meta-axis labels are not currently visible, display the meta-axis labels for the provided axis on the GLO Display.
- **Hide meta {*axis*} axis**: If the provided meta-axis's labels are currently visible, hide the meta-axis labels for the provided axis on the GLO Display.

APPENDIX E

GLOV2 LITERATURE REVIEW RESULTS

In this appendix, I list the visualization techniques that were not chosen as GLOv2 seed techniques during the literature review described in Section 3.1.2. I have grouped the techniques by their categories described in that section.

E.1 Tree Visualization Techniques

- Reingold and Tilford [185]
- H-Trees [75, 230]
- Radial tree layout [75]
- Cone Tree [187]
- Balloon Tree [55]
- RINGS [222]
- 3D Balloon Trees [24]
- Treemaps [134]
- Squarified Treemaps [254]
- Cushion Treemaps [236]
- Voronoi Treemaps [172]
- Spiral Treemaps [227]
- Balloon Focus [228]
- Spatially-Ordered Treemaps [254]
- Quantum Treemaps [35]
- Bubblemaps [35]
- Information Slices [12]

- Sunbursts [212]
- Fan Chart [70]
- Hyperbolic [169, 148]
- CHEOPS [33]
- Multitrees [92]
- 3D Cluster Tree [76]
- Information Pyramids [13]
- Collapsible Cylindrical Trees [66]
- Botanical Trees [143]
- Disk Tree [59]
- SpaceTree [181]
- Beam Trees [234]
- TreeWiz [188]
- MatrixZoom [4]
- PolyPlane [126]
- Expand-Ahead [162]
- Zoomology [125]
- Circle-packing [241]
- Elastic Hierarchies [260]
- Bubble tree [102]
- CandidTree [150]
- Visual tree comparison [122]
- Bar trees [183]
- Dendogram-matrix [58]
- rectangle packing [132]
- TreeVersity [97]
- Treeversity2 [103]

- Cascaded Treemap [157]
- Flow Map Layout [257]
- WebFan [258]
- AdaptivTree [220]

E.2 DAG Visualization Techniques

- Sugiyama [218]
- Needle grid [3, 2]
- Star map [3, 2]
- Multi-comb [3, 2]
- Multi-wedge [3, 2]
- Sketches [3, 2]
- Orthogonal Bars Sketch [3, 2]
- DAGView [100]
- Quilts [21]
- GeneaQuilts [37]
- TimeNet [142]
- DagTreemaps [226]
- pygmybrowser [25]

E.3 Graph Visualization Display Customizations

- Motifs [71]
- Alternate node glyphs [108]
- Clusters as Convex Hulls [108]
- Hierarchical Aggregation [77]
- Edge Labels as Edge Glyphs [252]
- Edge Bundling [257, 120, 65, 123, 261, 221, 79, 147, 200, 95, 146, 156]
- Alternate Directed Edge Glyphs [124, 121, 170]

- In-out Directed Edge Glyphs [74]
- Partial Directed Edge Glyphs [50]
- Node Duplication [115]
- Alternate Node Glyph Techniques [202]
- Interactive Link Curvature [186]

E.4 Graph Visualization Interaction Customizations

- Fisheye [194, 54]
- Multiple Fisheye [175]
- Hyperbolic Fisheye [180]
- Fog Fisheye [86]
- DOI-Trees [53, 112]
- EdgeLens [249]
- Space-Folding [78]
- Bring-and-Go [166]
- Selective Highlighting [108, 166]
- Link-Fanning [186]
- Link-Sliding [166]
- Structure-based Brushing [91]
- Edge-Plucking [250]
- Pan [32]
- Geometric Zoom [32]
- Semantic Zoom [251]
- Dynamic Queries [145]
- Prune+Grow (Gardening Ops) [187]
- Prune nodes/labels, but links remain [60]
- Drag Expand-Collapse [161]

- Treemap Zooming Interactions [38]
- Subgraph Selection [248, 163]
- MoleView [130]
- Complex Node Glyph Lens [136]
- Visual Queries [201]
- TreemapBar [127]

E.5 Dynamic Graph Visualization Techniques

- Animating [46, 67]
- 3D Columns [43]
- Animating Colors [193]
- Static Dynamic-showing Glyphs [193]
- GestaltMatrix [44]
- Parellel-Coordinate-esque [48]
- TimeRadarTrees [47]
- Timeline Trees [49]
- TimeArcTrees [101]
- Manynets [89]
- InSitu Dynamic Graphs [104]
- Dynamic Differences [16]
- 1.5D Network [203]
- Matrix Cube [20]
- DiffAni [191]
- Alluvial diagram of cluster changes [189, 184]
- stability/consistency slider [81]
- degree heatmap [99]
- Weighted Comparison [11]

- Dynamic Hypergraph Vis [133]
- Pixel-Oriented Matrices [215]
- Tree-Ring Social Networks [80]
- TimeMatrix [259]

E.6 General Graph Visualization Non-Seed Techniques

E.6.1 Reduce to Tree

- SPF [14]
- MO-Tree [42]
- Space-Filling Curves [167]
- Treemap of hierarchy [168]
- Treemaps with Links [82]
- Similarity trees [177]
- ArcTree [171]
- TreePlus [149]
- Grouse [17]
- GrouseFlocks [15]
- TreeNetViz [98]

E.6.2 Three-Dimensional

- Ask-Graphview [5]
- 3D node-link using stereoscope [242]
- Graph Surfaces [6]
- Cityscape [57]
- State-Transition Graphs [233]
- Landscape [45]
- WilmaScope [9]

E.6.3 Topology-Dependent

- C-Group [139, 27]
- B-Matrix [22]
- Compressed Adjacency Matrix [68]
- Edge-Compression [72]
- SegmentView [27]
- JauntyNet [137]

APPENDIX F

HIERARCHICAL CLUSTERINGS

Here I present the results of the various hierarchical clusterings using the three vectorization methods (no-flags, flags, and flags-xtra), three distance metrics (Hamming distance [107], Jaccard distance [151], and cosine distance [206]), and four cluster comparison methods (single, complete [239], average [210], and weighted [210]) described in Section 5.2.1. Within each vectorization method, the clustering results are sorted by their cophenetic correlation coefficient [211], which is a measure of how closely a hierarchical clustering maintains pair-wise distances between data points using the chosen distance metric.

F.1 Binary GLO-Vectors without Optional Parameters (no-flags)

Average Method and Hamming Distance Cophenetic Correlation: 0.832799904046

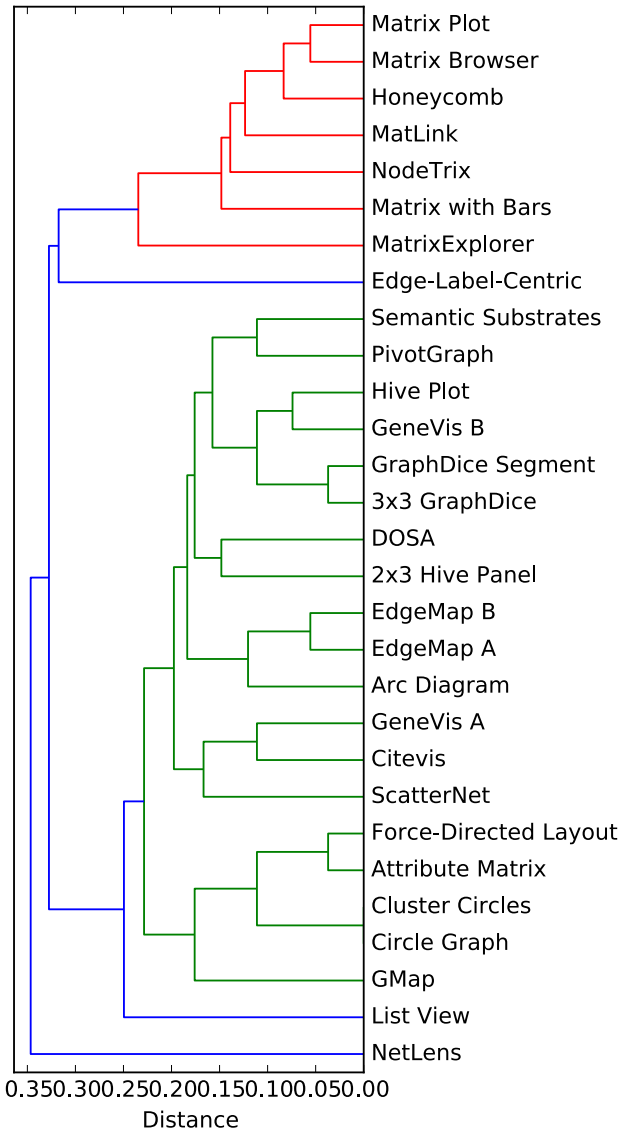


Figure 126: SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and Hamming distance.

Weighted Method and Hamming Distance Cophenetic Correlation: 0.821886497865

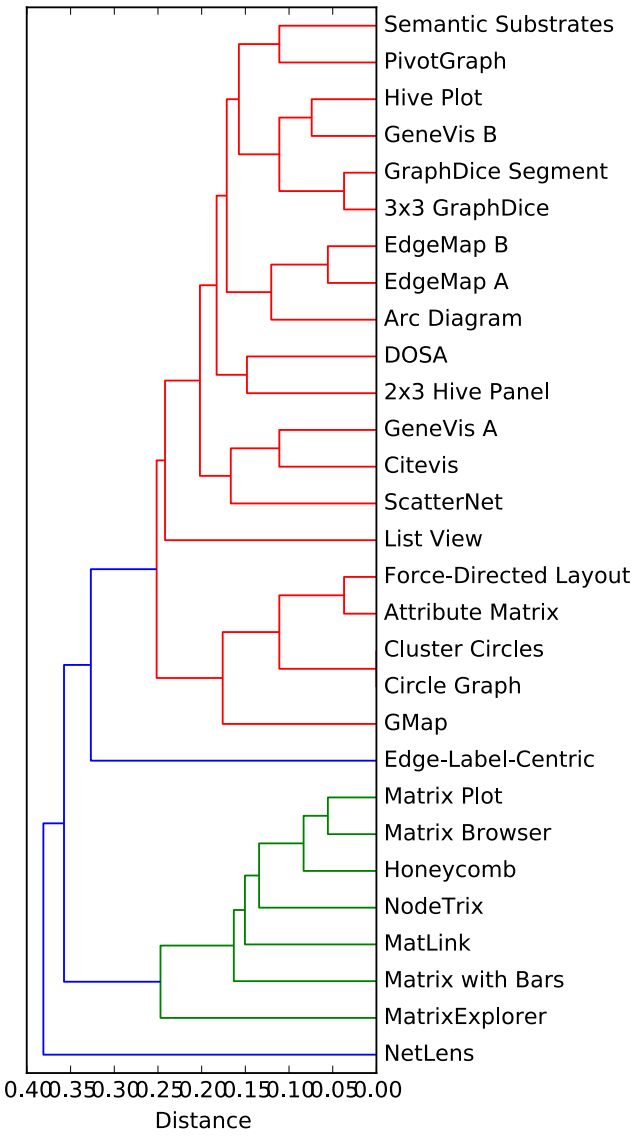


Figure 127: SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and Hamming distance.

Weighted Method and Jaccard Distance Cophenetic Correlation: 0.755900447652

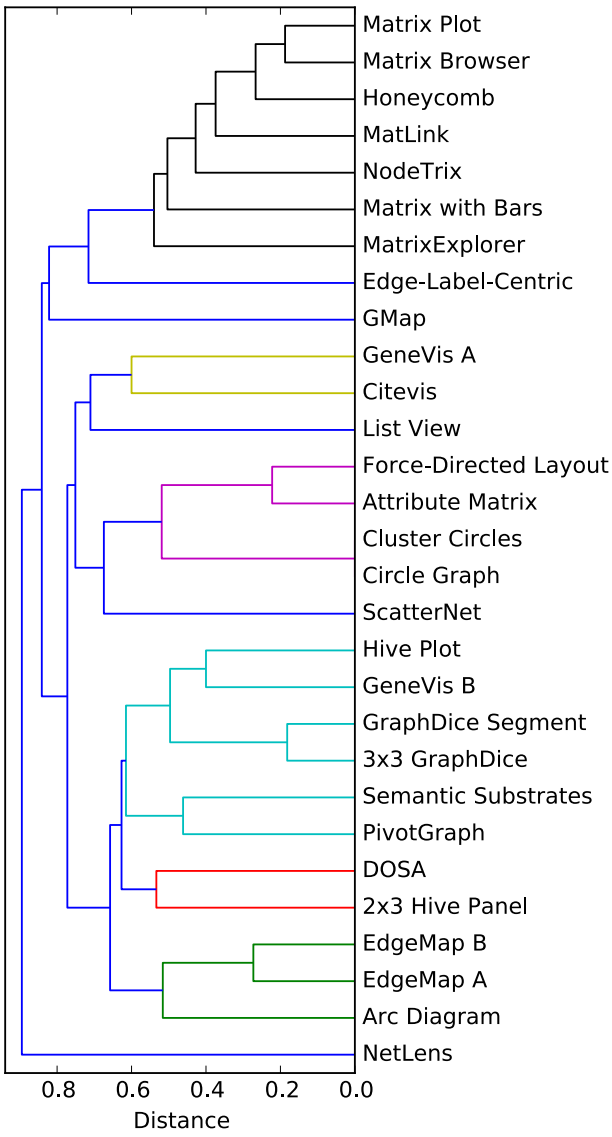


Figure 128: SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and Jaccard distance.

Complete Method and Hamming Distance Cophenetic Correlation: 0.726395383264

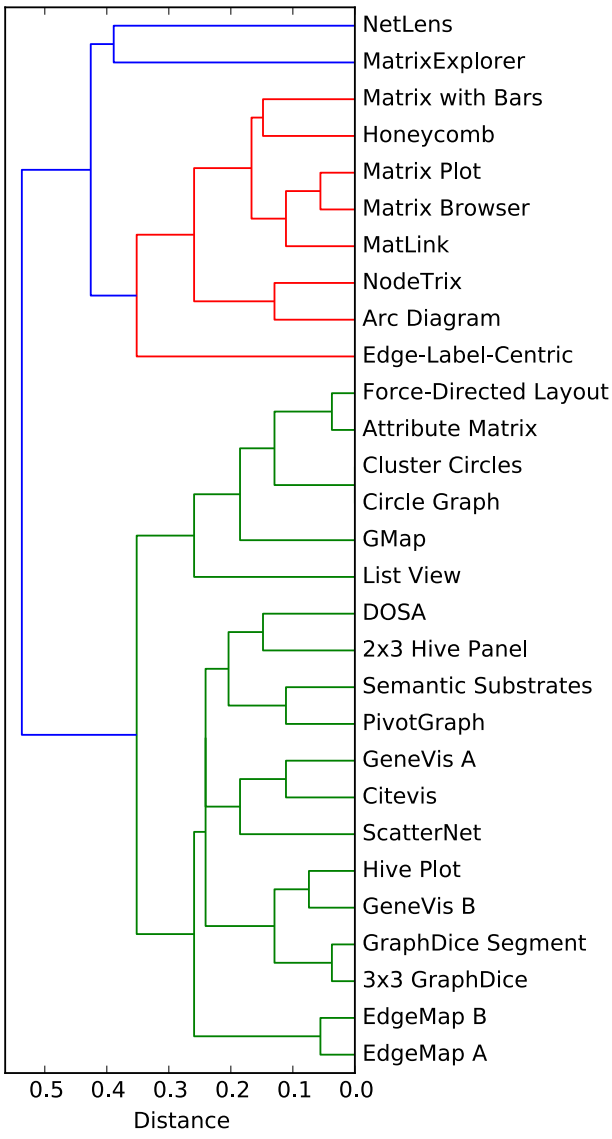


Figure 129: SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and Hamming distance.

Average Method and Cosine Distance Cophenetic Correlation: 0.703449587146

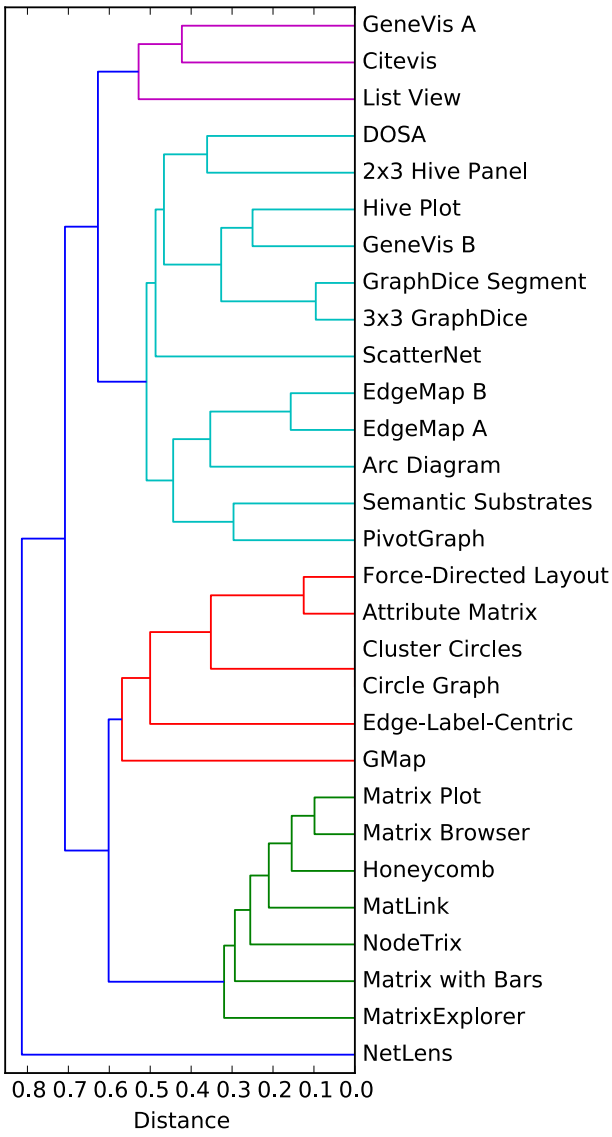


Figure 130: SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and cosine distance.

Average Method and Jaccard Distance Cophenetic Correlation: 0.706271040351

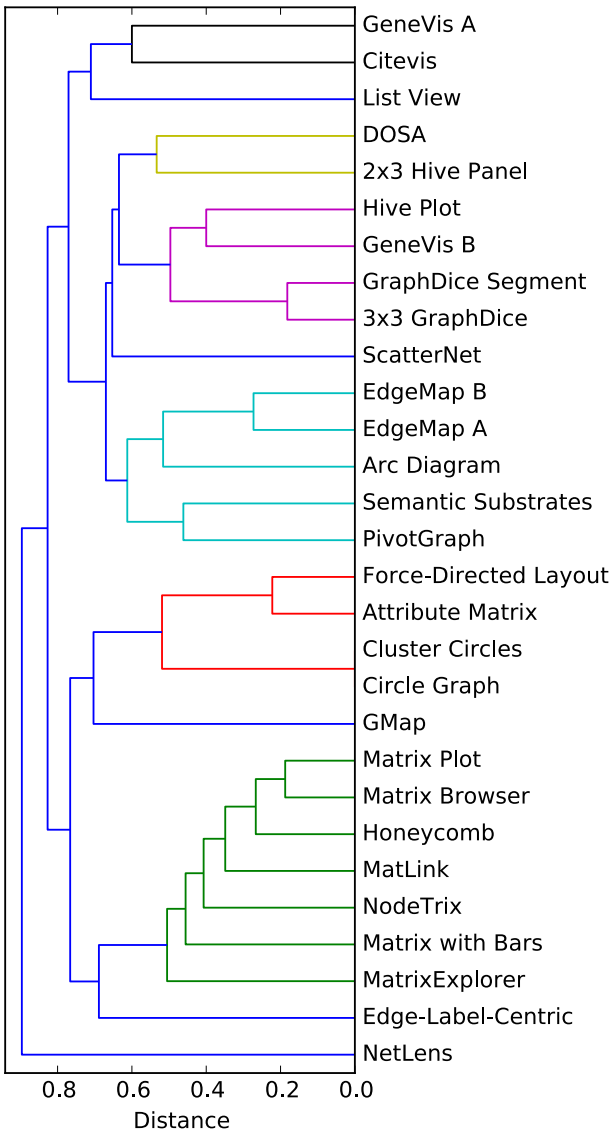


Figure 131: SciPy hierarchical clustering of GLO-Vectors without optional parameters using average method and Jaccard distance.

Complete Method and Jaccard Distance Cophenetic Correlation: 0.68661661651

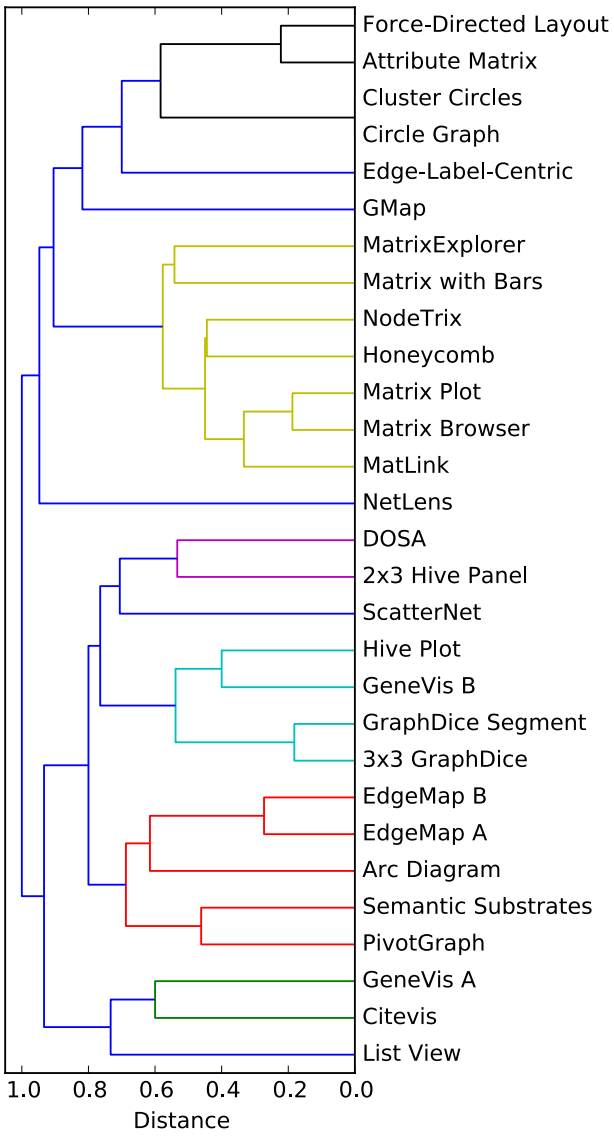


Figure 132: SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and Jaccard distance.

Weighted Method and Cosine Distance Cophenetic Correlation: 0.693924001138

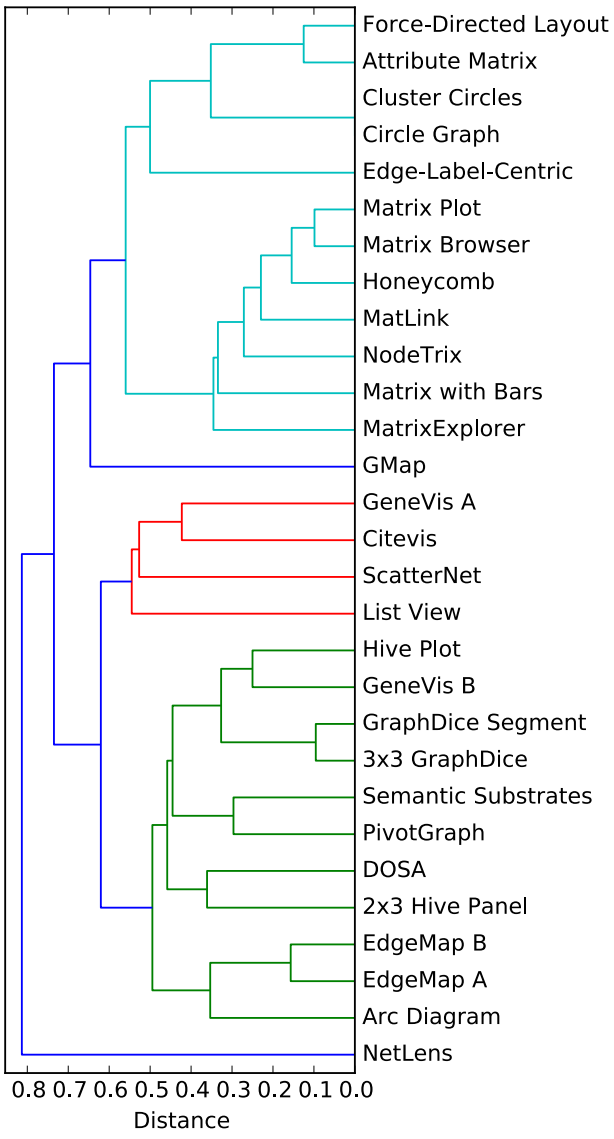


Figure 133: SciPy hierarchical clustering of GLO-Vectors without optional parameters using weighted method and cosine distance.

Complete Method and Cosine Distance Cophenetic Correlation: 0.676424420188

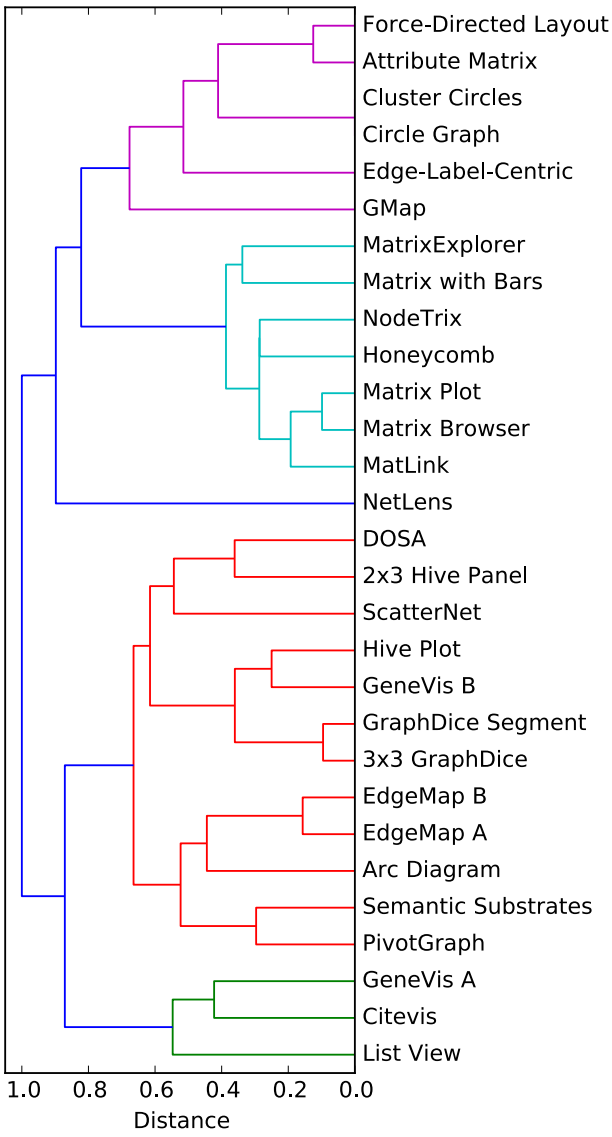


Figure 134: SciPy hierarchical clustering of GLO-Vectors without optional parameters using complete method and cosine distance.

Single Method and Hamming Distance Cophenetic Correlation: 0.592861499268

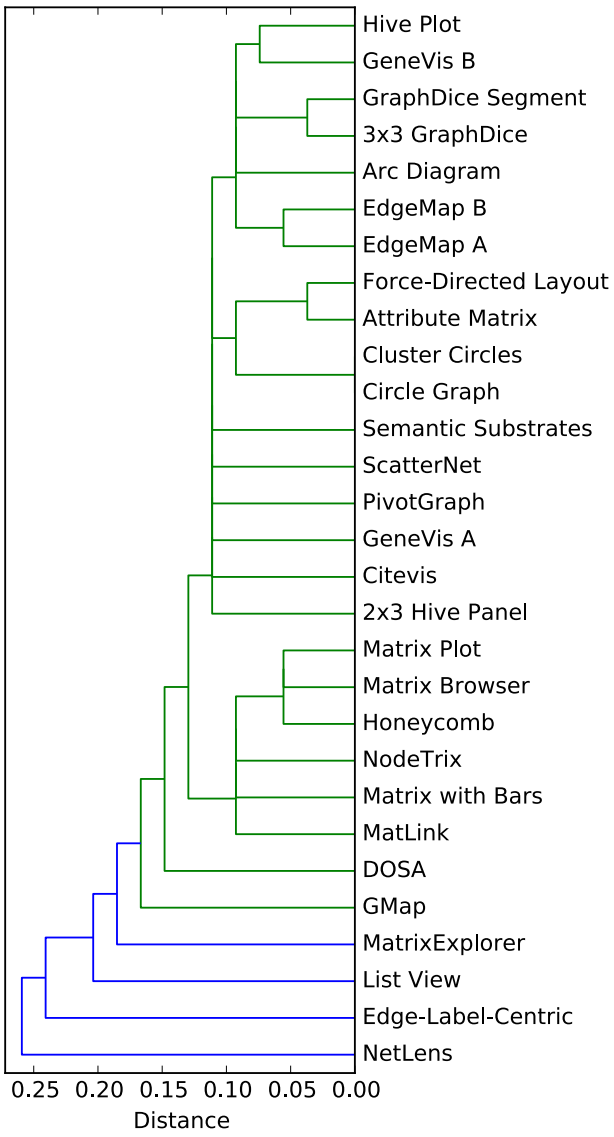


Figure 135: SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and Hamming distance.

Single Method and Jaccard Distance Cophenetic Correlation: 0.468258988449

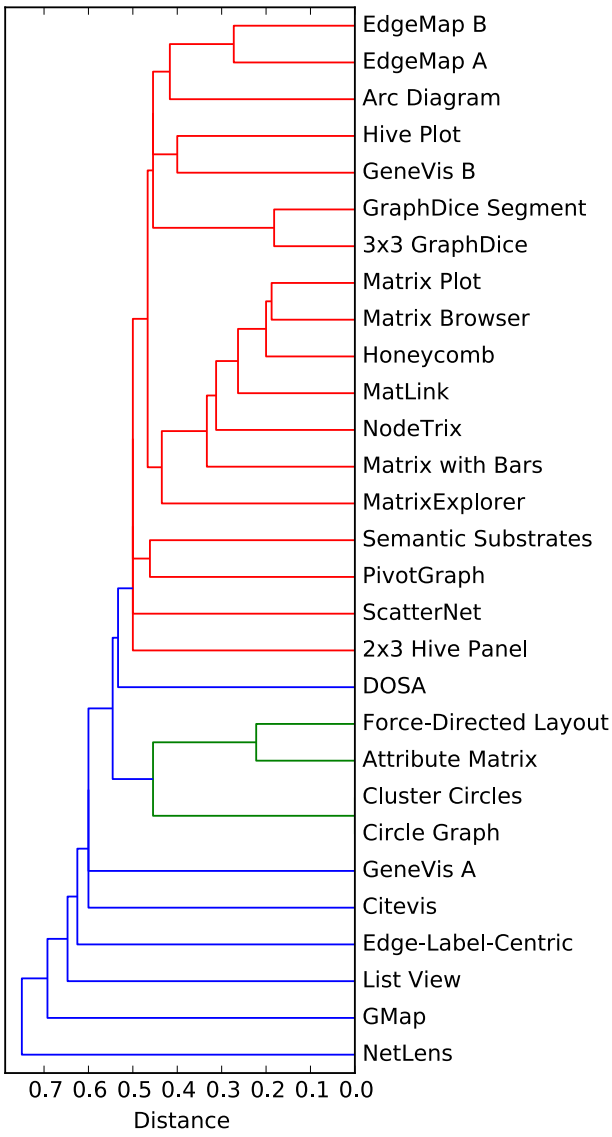


Figure 136: SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and Jaccard distance.

Single Method and Cosine Distance Cophenetic Correlation: 0.425621644926

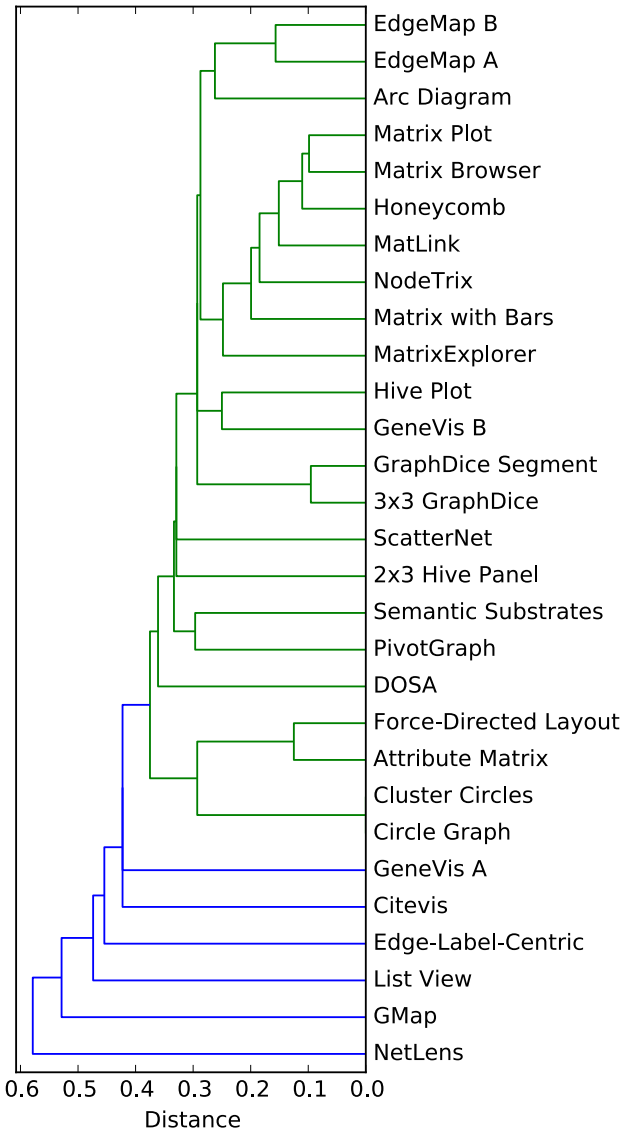


Figure 137: SciPy hierarchical clustering of GLO-Vectors without optional parameters using single method and cosine distance.

F.2 Binary GLO-Vectors with Optional Parameters (flags)

Average Method and Hamming Distance Cophenetic Correlation: 0.82737073354

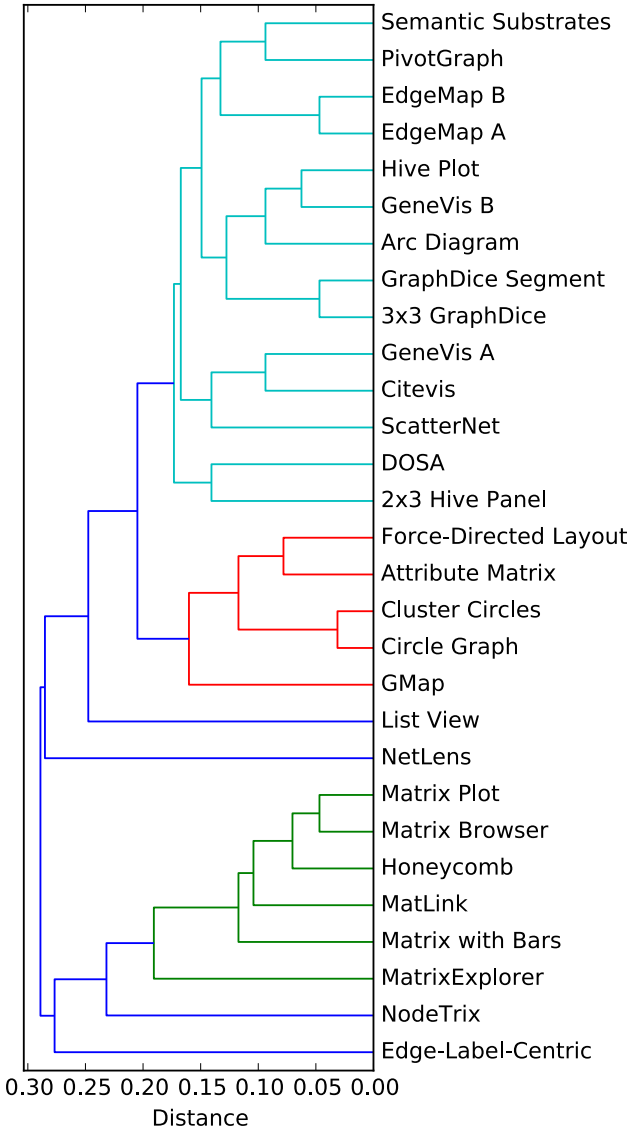


Figure 138: SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and Hamming distance.

Weighted Method and Hamming Distance Cophenetic Correlation: 0.818087307422

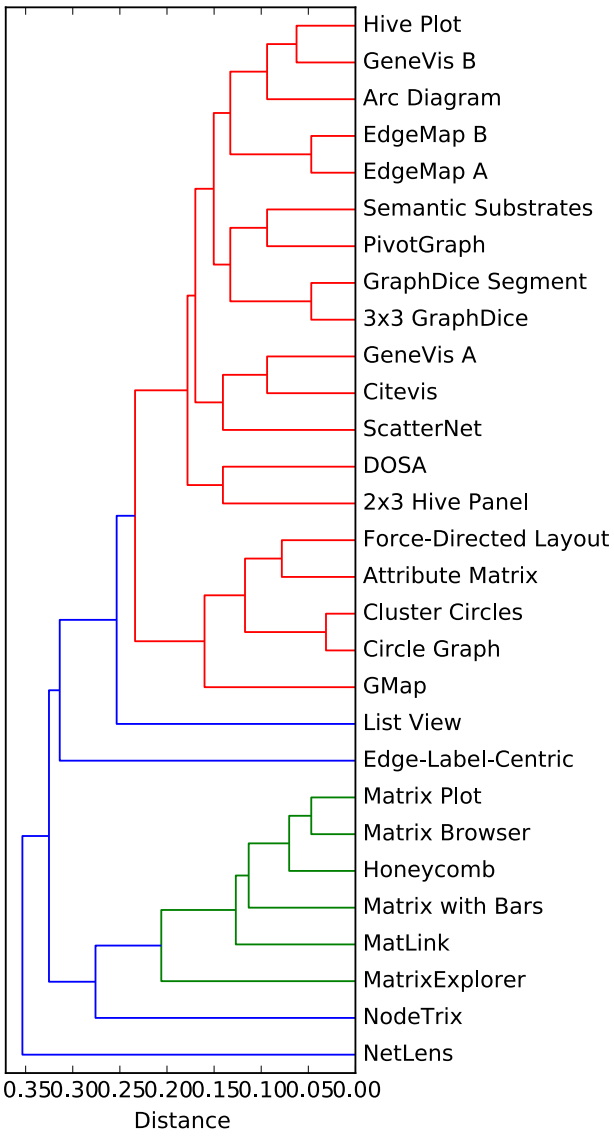


Figure 139: SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and Hamming distance.

Complete Method and Hamming Distance Cophenetic Correlation: 0.783238551964

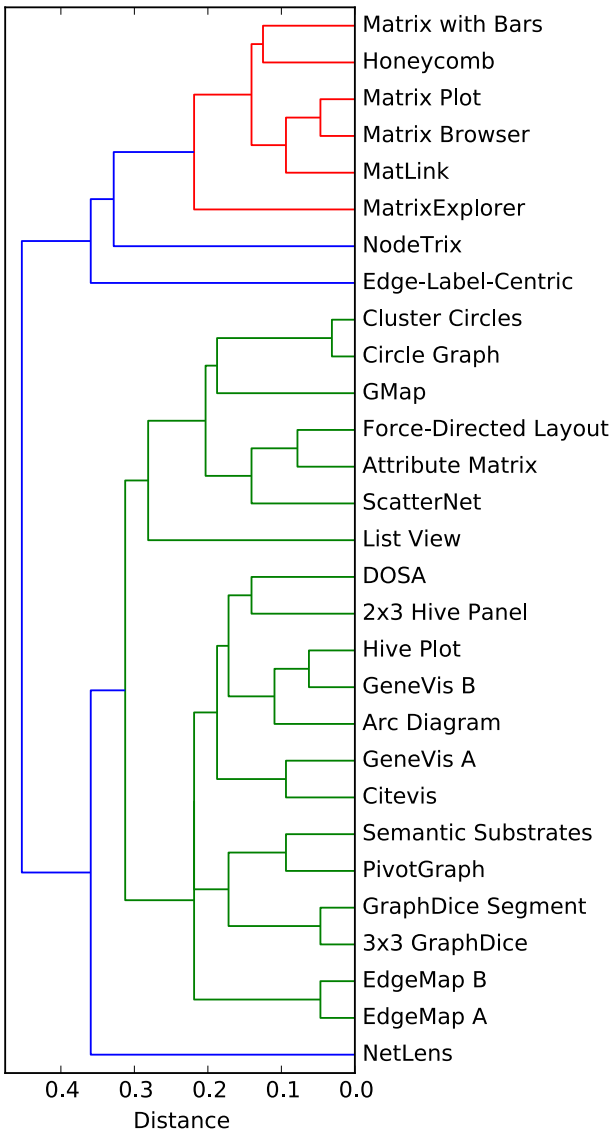


Figure 140: SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and Hamming distance.

Single Method and Hamming Distance Cophenetic Correlation: 0.712729004514

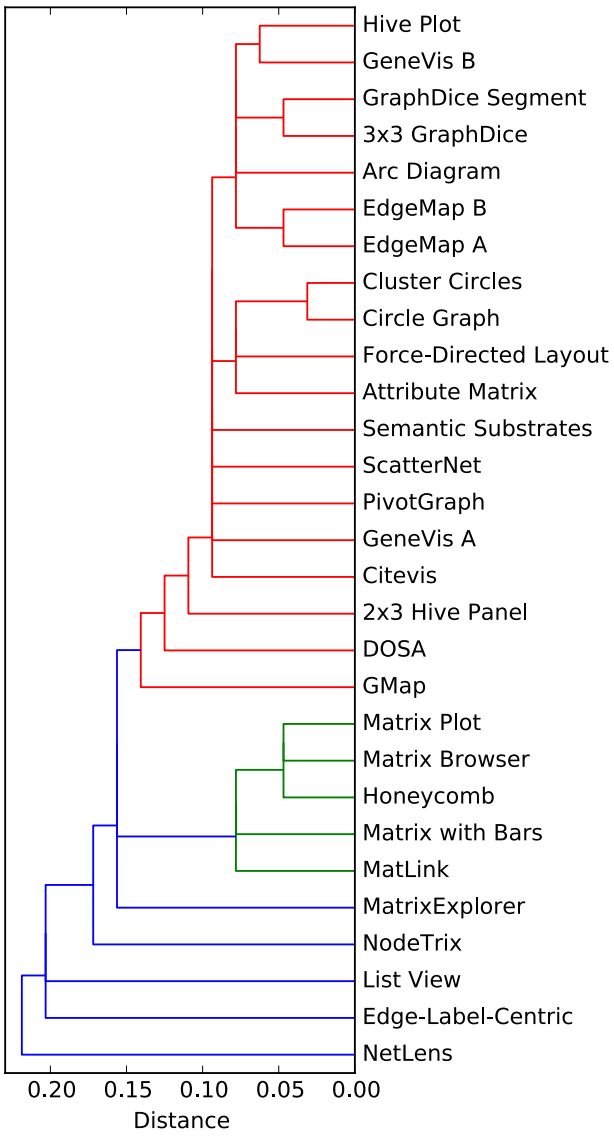


Figure 141: SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and Hamming distance.

Average Method and Cosine Distance Cophenetic Correlation: 0.683772174974

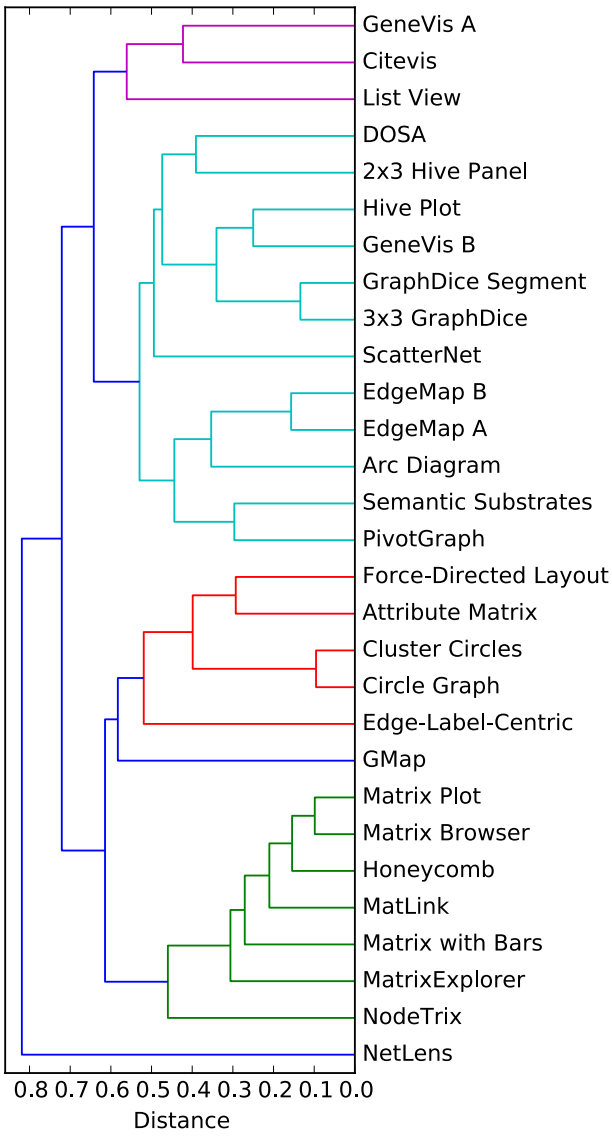


Figure 142: SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and cosine distance.

Weighted Method and Cosine Distance Cophenetic Correlation: 0.693246619992

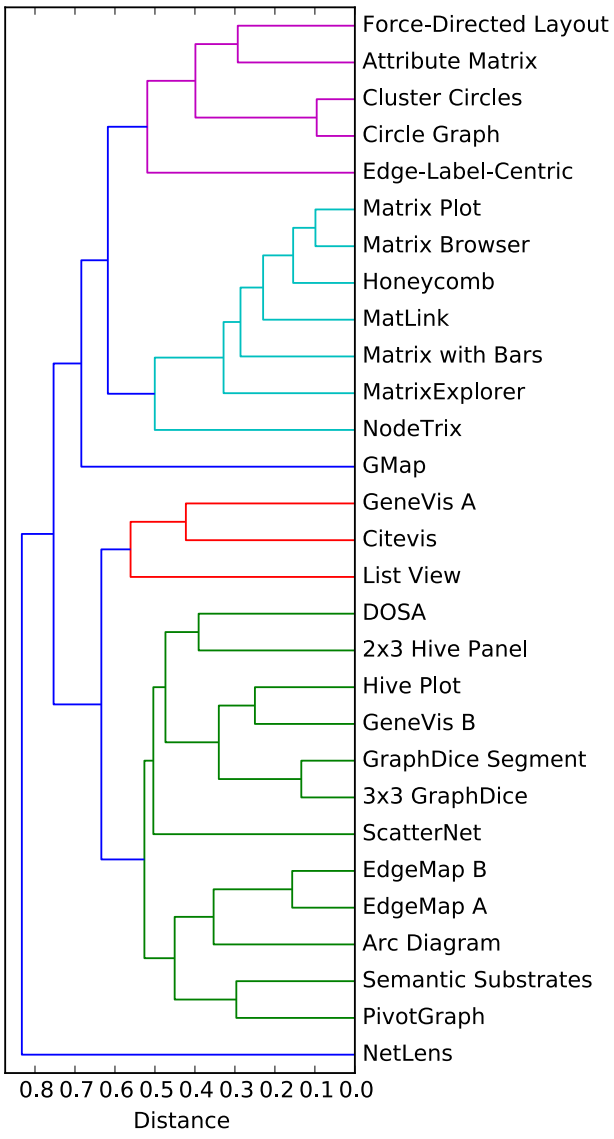


Figure 143: SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and cosine distance.

Weighted Method and Jaccard Distance Cophenetic Correlation: 0.694940609581

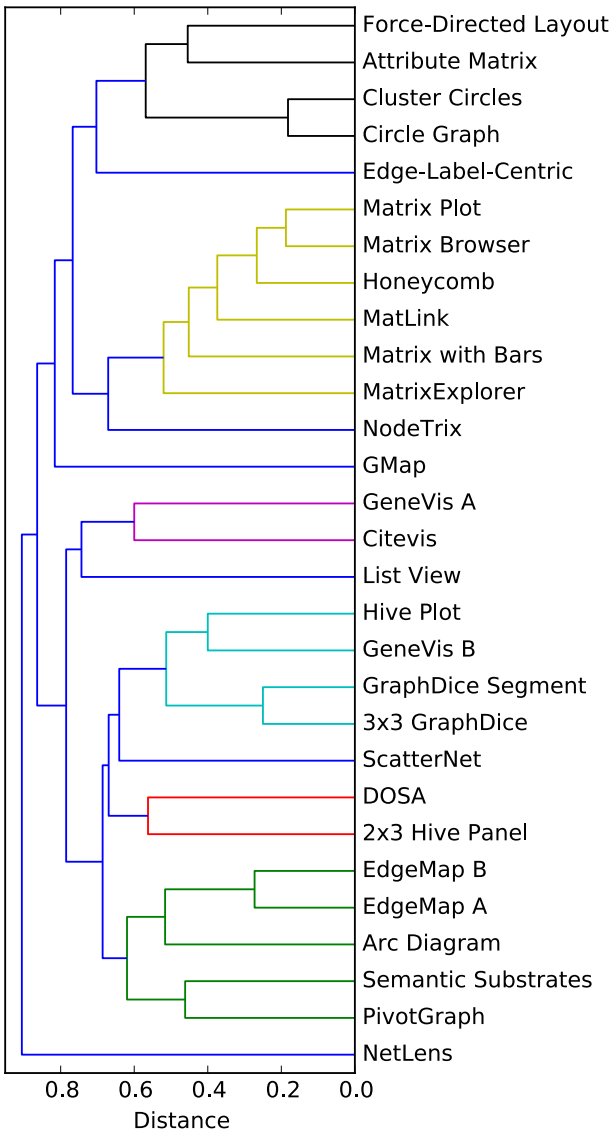


Figure 144: SciPy hierarchical clustering of GLO-Vectors with optional parameters using weighted method and Jaccard distance.

Average Method and Jaccard Distance Cophenetic Correlation: 0.681375526271

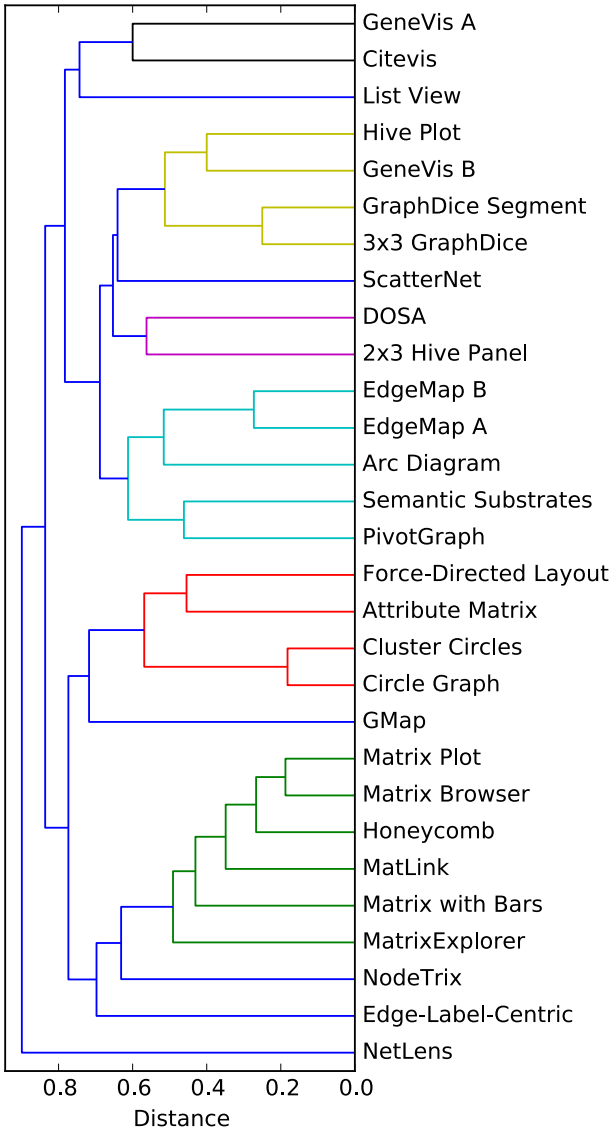


Figure 145: SciPy hierarchical clustering of GLO-Vectors with optional parameters using average method and Jaccard distance.

Single Method and Jaccard Distance Cophenetic Correlation: 0.659515346766

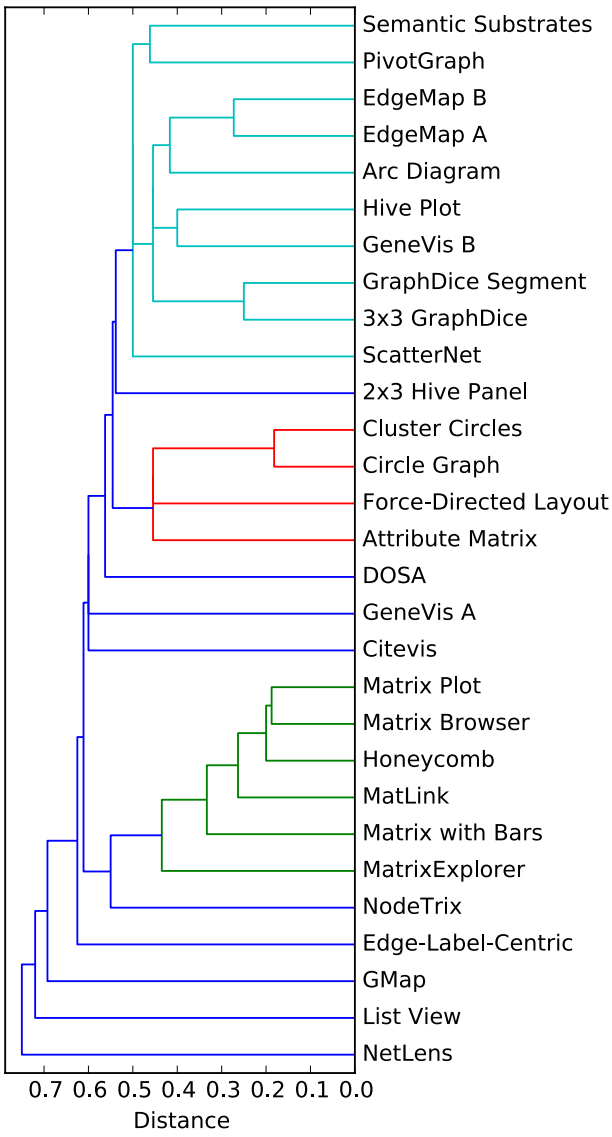


Figure 146: SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and Jaccard distance.

Single Method and Cosine Distance Cophenetic Correlation: 0.607775788637

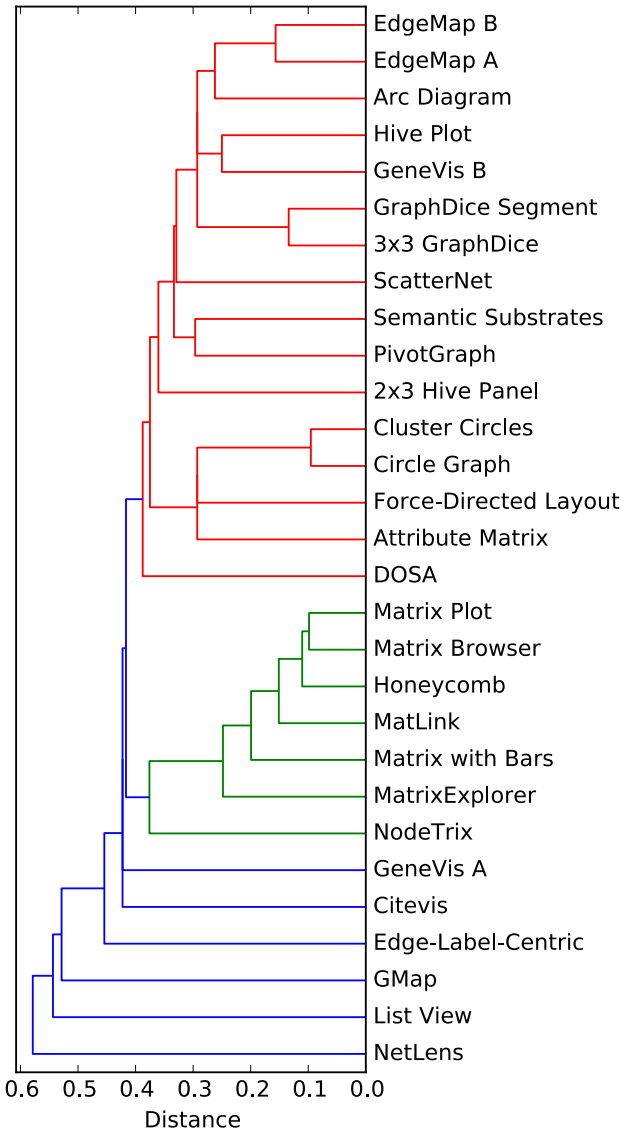


Figure 147: SciPy hierarchical clustering of GLO-Vectors with optional parameters using single method and cosine distance.

Complete Method and Jaccard Distance Cophenetic Correlation: 0.581438654173

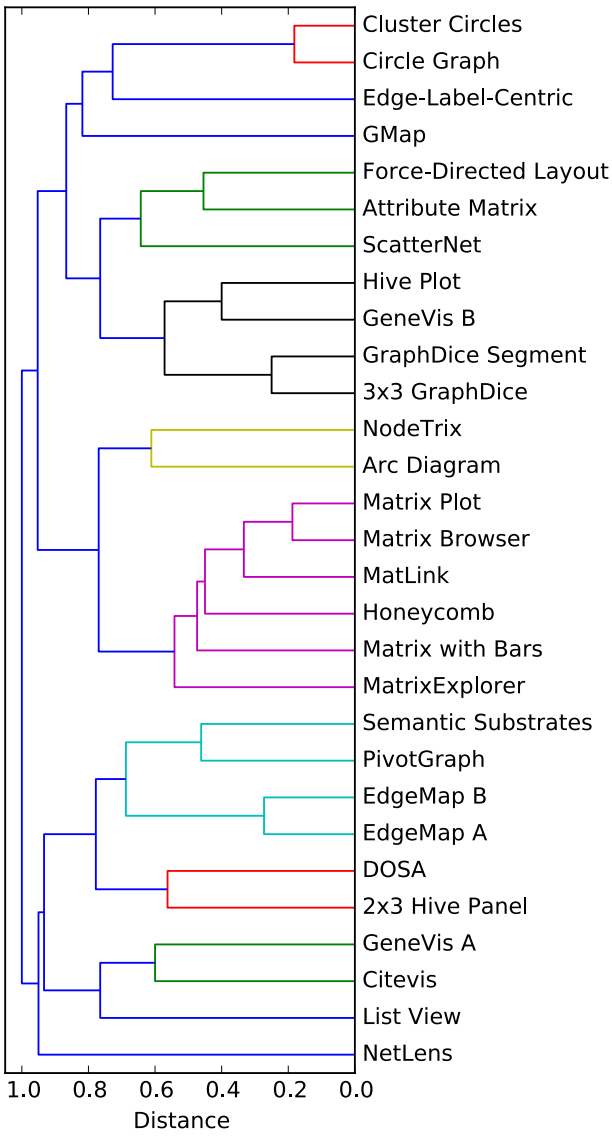


Figure 148: SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and Jaccard distance.

Complete Method and Cosine Distance Cophenetic Correlation: 0.567172438706

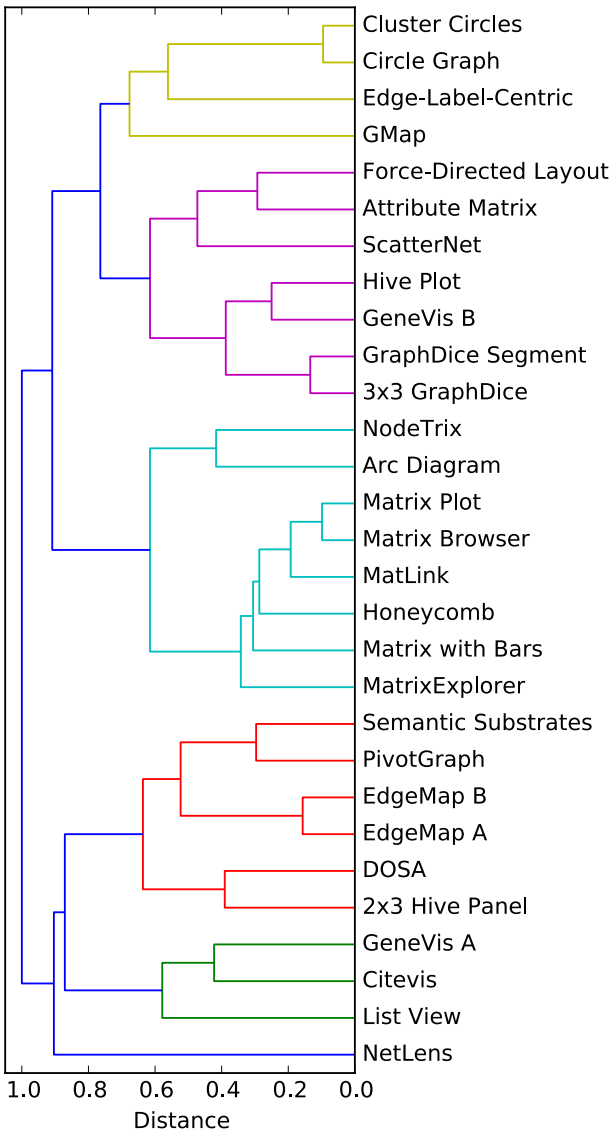


Figure 149: SciPy hierarchical clustering of GLO-Vectors with optional parameters using complete method and cosine distance.

F.3 Binary GLO-Vectors with Optional Parameters as Features (flags-xtra)

Average Method and Hamming Distance Cophenetic Correlation: 0.824649599232

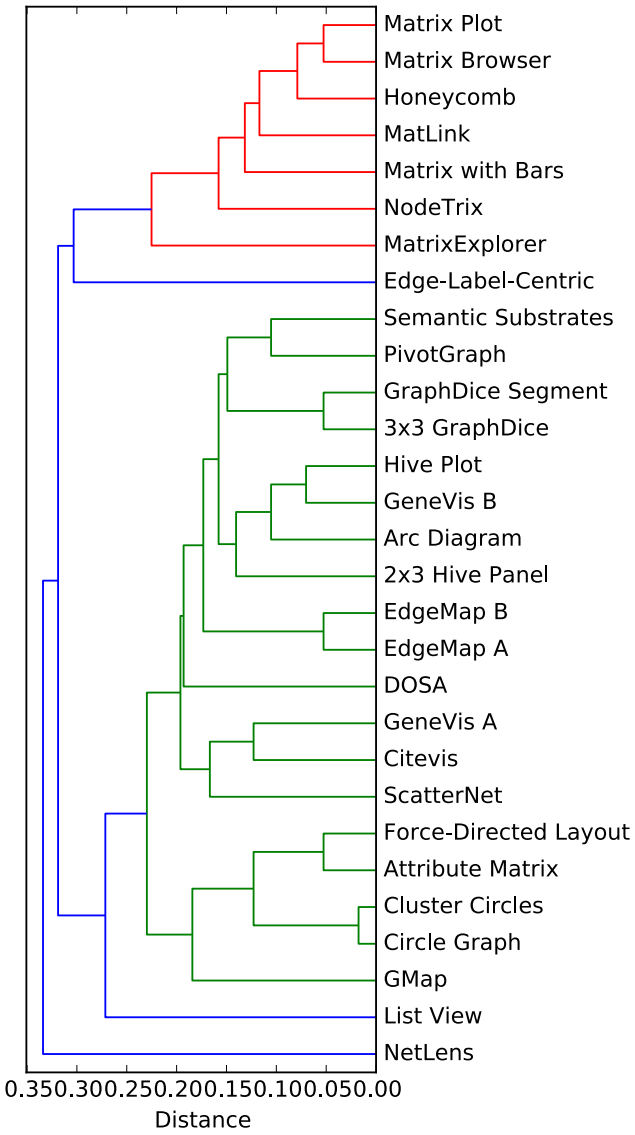


Figure 150: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and Hamming distance.

Weighted Method and Hamming Distance Cophenetic Correlation: 0.813288450195

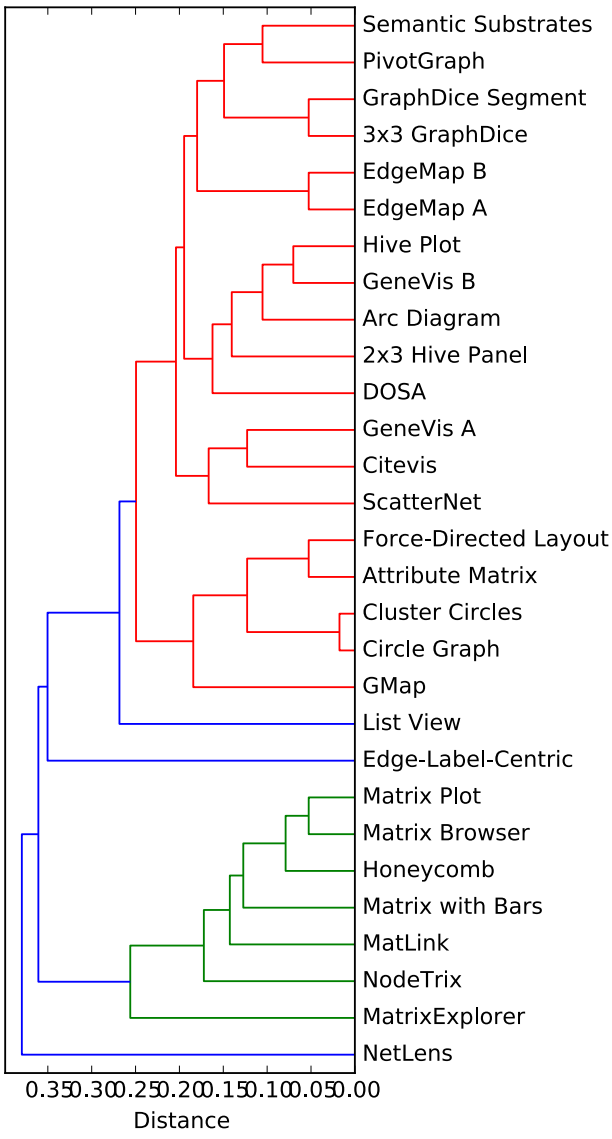


Figure 151: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and Hamming distance.

Complete Method and Hamming Distance Cophenetic Correlation: 0.780160002224

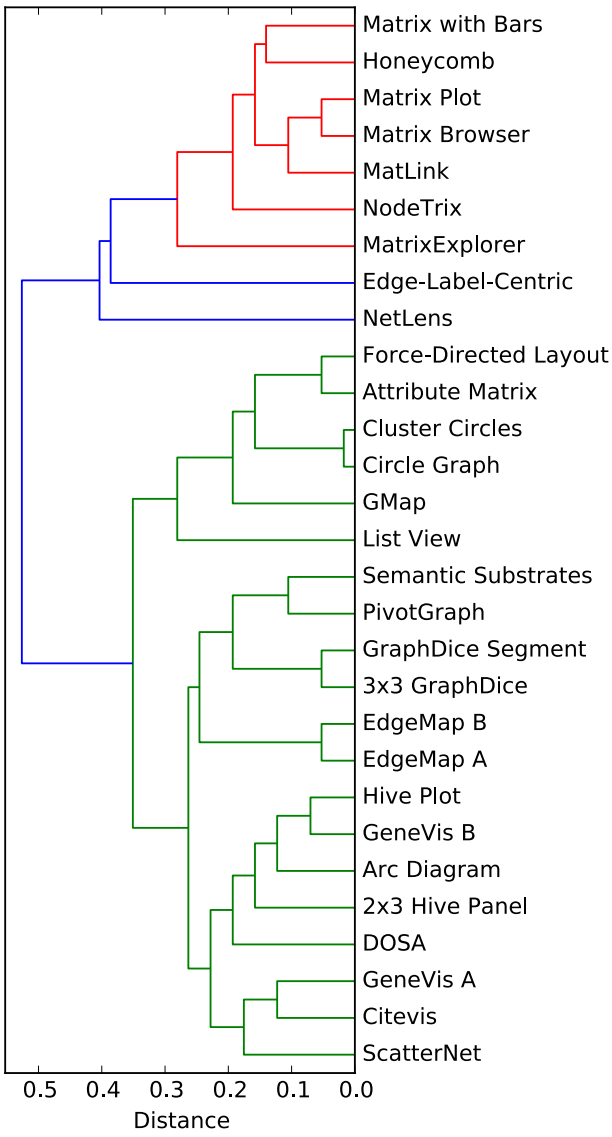


Figure 152: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and Hamming distance.

Average Method and Cosine Distance Cophenetic Correlation: 0.699558498029

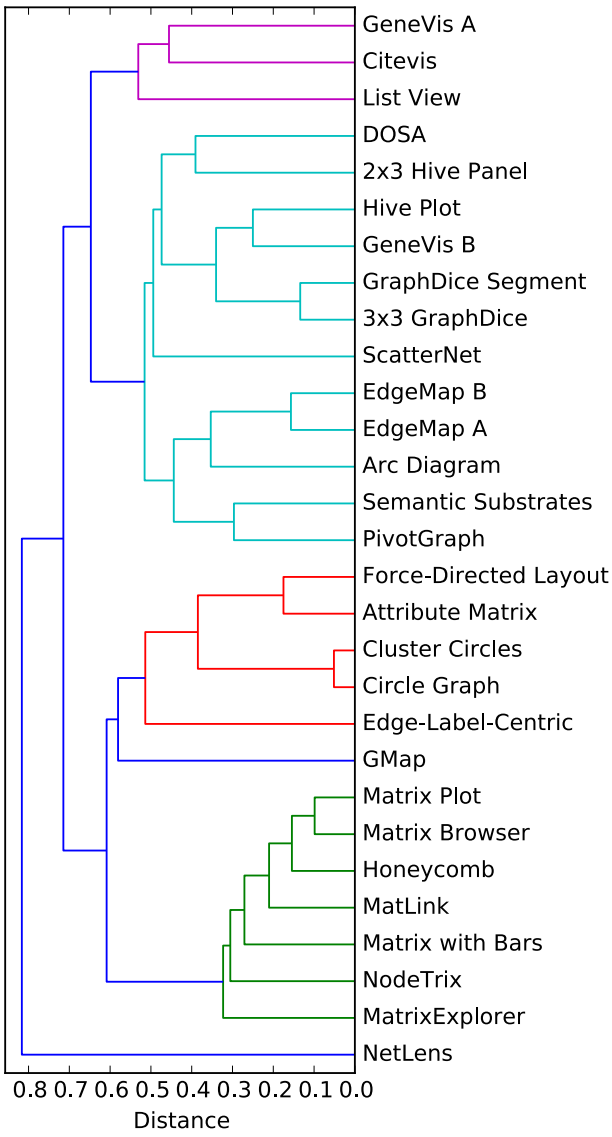


Figure 153: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and cosine distance.

Average Method and Jaccard Distance Cophenetic Correlation: 0.698051712306

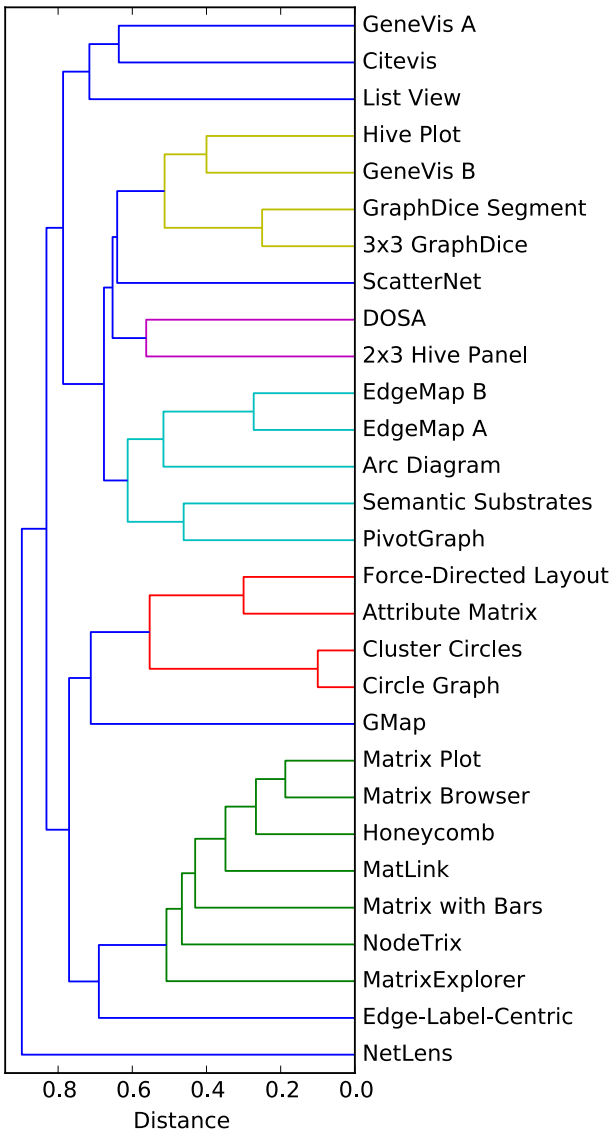


Figure 154: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using average method and Jaccard distance.

Weighted Method and Cosine Distance Cophenetic Correlation: 0.660528026618

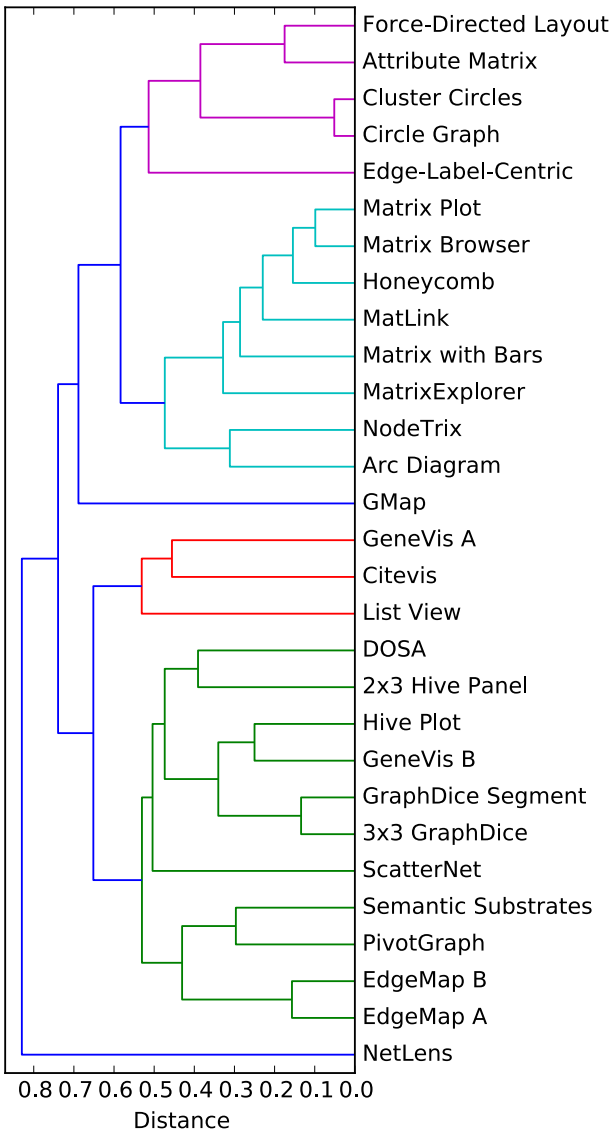


Figure 155: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and cosine distance.

Weighted Method and Jaccard Distance Cophenetic Correlation: 0.668918009918

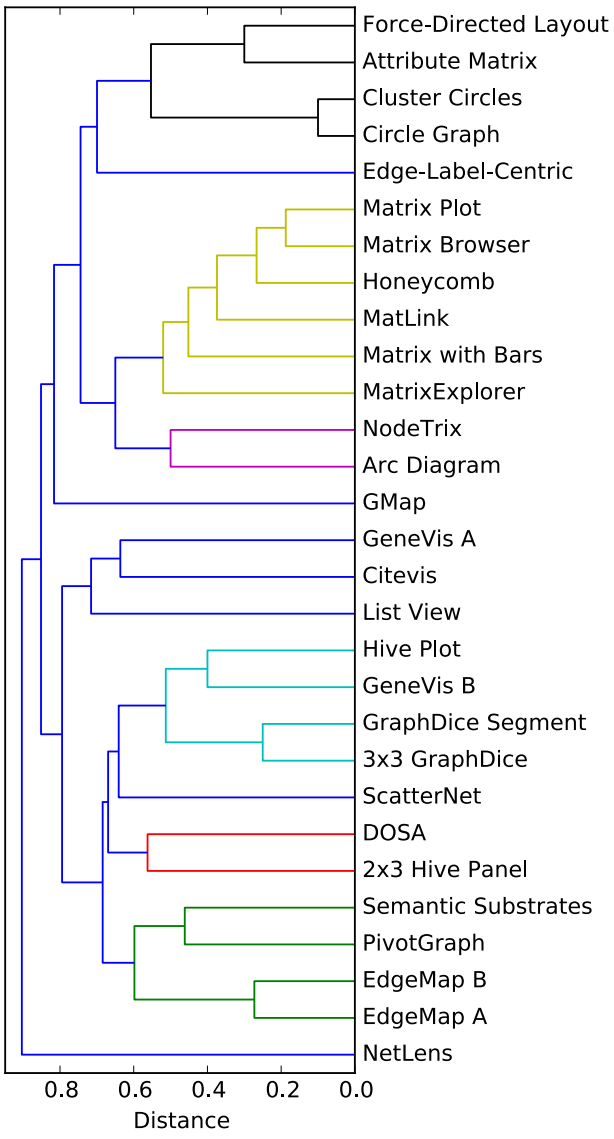


Figure 156: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using weighted method and Jaccard distance.

Single Method and Hamming Distance Cophenetic Correlation: 0.639731210892

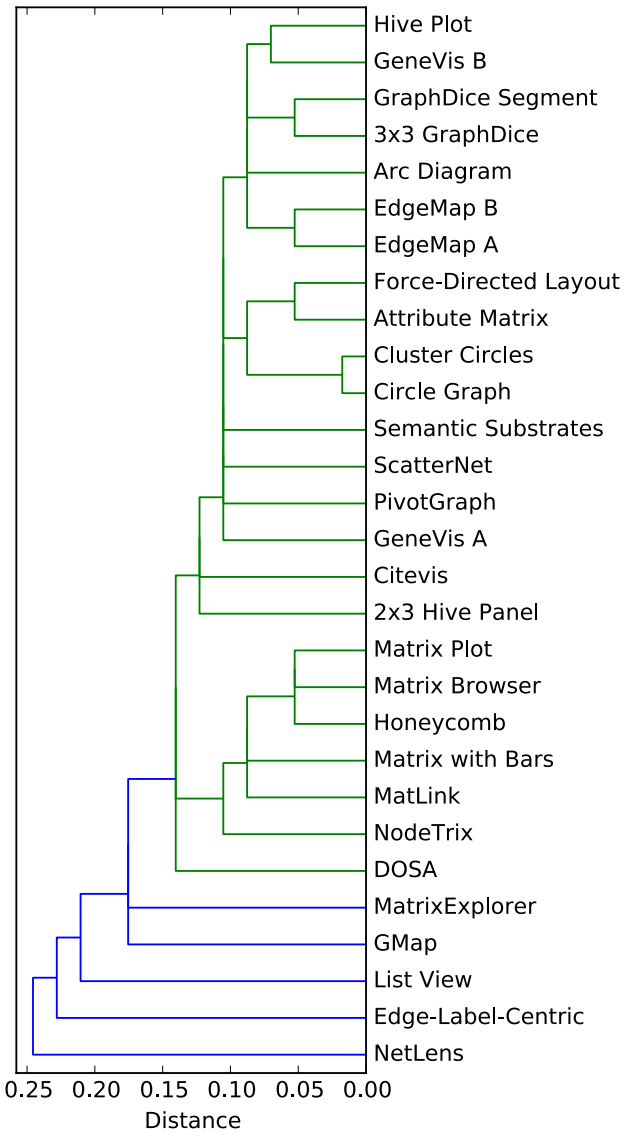


Figure 157: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and Hamming distance.

Complete Method and Cosine Distance Cophenetic Correlation: 0.643819269884

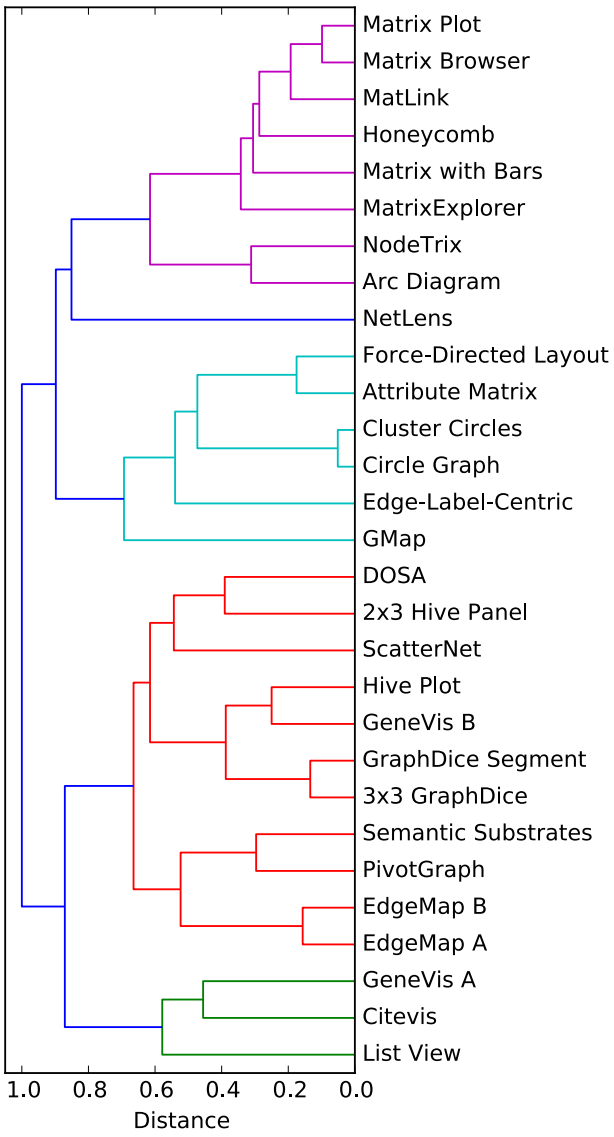


Figure 158: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and cosine distance.

Complete Method and Jaccard Distance Cophenetic Correlation: 0.63682320527

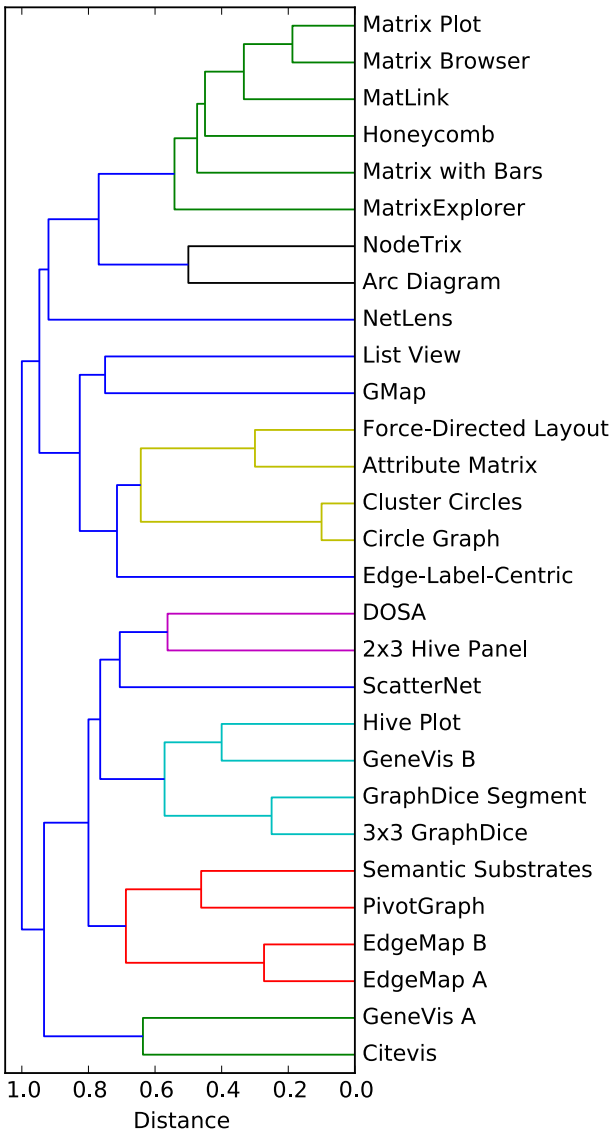


Figure 159: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using complete method and Jaccard distance.

Single Method and Jaccard Distance Cophenetic Correlation: 0.514020768581

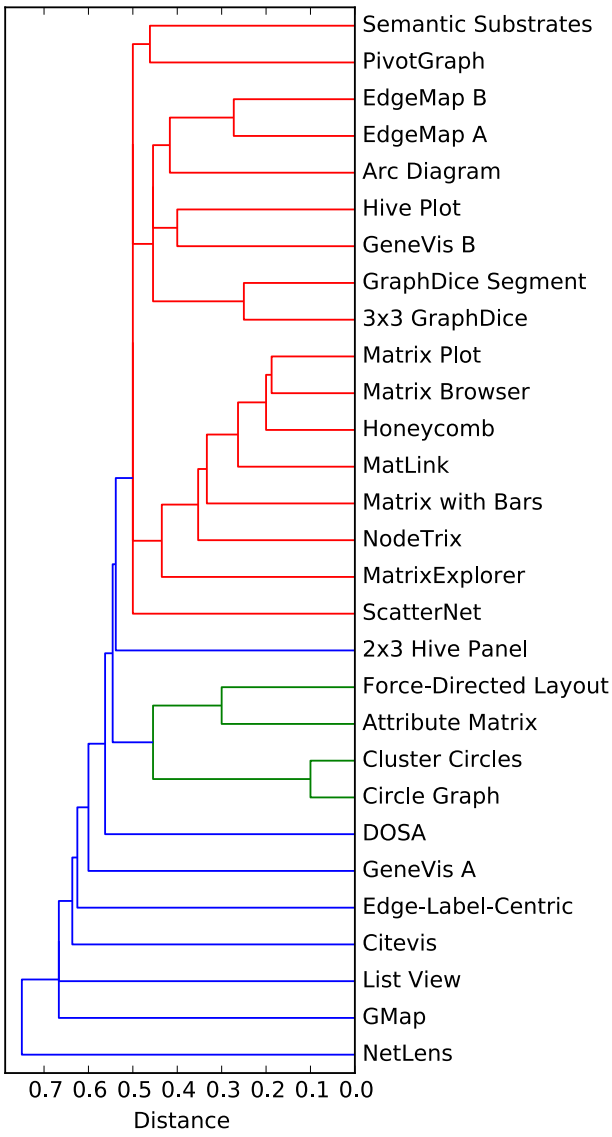


Figure 160: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and Jaccard distance.

Single Method and Cosine Distance Cophenetic Correlation: 0.471452345878

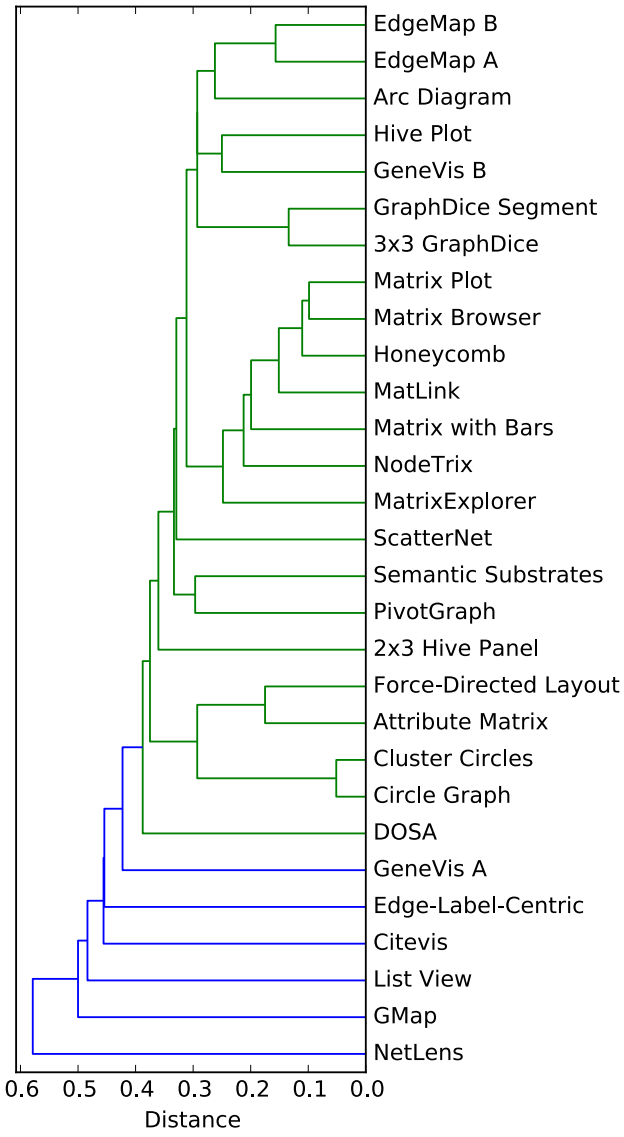


Figure 161: SciPy hierarchical clustering of GLO-Vectors with optional parameters as additional features using single method and cosine distance.

REFERENCES

[1] "Processing." http://processing.org.

[2] Abello, J., Finocchi, I., and Korn, J., "Graph sketches," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*, pp. 67–70, 2001.

[3] Abello, J. and Korn, J., "Visualizing massive multi-digraphs," in *IEEE Symposium on Information Visualization, 2000. InfoVis 2000*, pp. 39–47, 2000.

[4] Abello, J. and van Ham, F., "Matrix Zoom: A Visual Interface to Semi-External Graphs," in *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pp. 183–190, 2004.

[5] Abello, J., van Ham, F., and Krishnan, N., "ASK-GraphView: A Large Scale Graph Visualization System," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 669–676, Sept. 2006.

[6] Abello, J. and Krishnan, S., "Graph Surfaces," in *Proceedings of International Congress on Industrial and Applied Math*, pp. 234–244, 1999.

[7] Adar, E., "GUESS: A Language and Interface for Graph Exploration," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, (New York, NY, USA), pp. 791–800, ACM, 2006.

[8] Adobe Software, "Illustrator." https://www.adobe.com/products/illustrator.html.

[9] Ahmed, A., Dwyer, T., Murray, C., Song, L., and Wu, Y. X., "WilmaScope Graph Visualisation," in *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pp. r4–r4, Oct. 2004.

[10] Aiken, A., Chen, J., Lin, M., Spalding, M., Stonebraker, M., and Woodruff, A., "The Tioga-2 database visualization environment," in *Database Issues for Data Visualization* (Wierse, A., Grinstein, G., and Lang, U., eds.), vol. 1183 of *Lecture Notes in Computer Science*, pp. 181–207, Springer Berlin / Heidelberg, 1996.

[11] Alper, B., Bach, B., Henry Riche, N., Isenberg, T., and Fekete, J.-D., "Weighted Graph Comparison Techniques for Brain Connectivity Analysis," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, (New York, NY, USA), pp. 483–492, ACM, 2013.

[12] Andrews, K. and Heidegger, H., "Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs," in *Proc of IEEE Infovis 98 late breaking Hot Topics*, pp. 9–11, 1998.

[13] Andrews, K., Wolte, J., and Pichler, M., "Information PyramidsTM: A new approach to visualizing large hierarchies," in *Proceedings of the IEEE Visualization97*, pp. 49–52, 1997.

[14] Archambault, D., Munzner, T., and Auber, D., "Smashing Peacocks Further: Drawing Quasi-Trees from Biconnected Components," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 813–820, Sept. 2006.

[15] Archambault, D., Munzner, T., and Auber, D., "GrouseFlocks: Steerable Exploration of Graph Hierarchy Space," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 900–913, July 2008.

[16] Archambault, D., "Structural Differences Between Two Graphs Through Hierarchies," in *Proceedings of Graphics Interface 2009*, GI '09, (Toronto, Ont., Canada, Canada), pp. 87–94, Canadian Information Processing Society, 2009.

[17] Archambault, D., Munzner, T., and Auber, D., "Grouse: Feature-Based, Steerable Graph Hierarchy Exploration," in *Eurographics/ IEEE-VGTC Symposium on Visualization* (Museth, K., Moeller, T., and Ynnerman, A., eds.), The Eurographics Association, 2007.

[18] Aris, A. and Shneiderman, B., "Designing Semantic Substrates for Visual Network Exploration," *Information Visualization*, vol. 6, pp. 281–300, Dec. 2007.

[19] Auber, D., "Tulip A Huge Graph Visualization Framework," in *Graph Drawing Software* (Jnger, M. and Mutzel, P., eds.), Mathematics and Visualization, pp. 105–126, Springer Berlin Heidelberg, 2004.

[20] Bach, B., Pietriga, E., and Fekete, J.-D., "Visualizing Dynamic Networks with Matrix Cubes," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, (New York, NY, USA), pp. 877–886, ACM, 2014.

[21] Bae, J. and Watson, B., "Developing and Evaluating Quilts for the Depiction of Large Layered Graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2268–2275, Dec. 2011.

[22] Bagrow, J. P., Bollt, E. M., Skufca, J. D., and ben Avraham, D., "Portraits of complex networks," *EPL (Europhysics Letters)*, vol. 81, p. 68004, Mar. 2008.

[23] Baker, C., Carpendale, M., Prusinkiewicz, P., and Surette, M., "GeneVis: visualization tools for genetic regulatory network dynamics," in *IEEE Visualization, 2002. VIS 2002*, pp. 243–250, Nov. 2002.

[24] Balzer, M. and Deussen, O., "Hierarchy Based 3d Visualization of Large Software Structures," in *IEEE Visualization, 2004*, pp. 4p–4p, Oct. 2004.

[25] Band, Z. and White, R. W., "PygmyBrowse: A Small Screen Tree Browser," in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, (New York, NY, USA), pp. 514–519, ACM, 2006.

[26] Baranovskiy, D., "Raphael."

[27] Basole, R., Clear, T., Hu, M., Mehrotra, H., and Stasko, J., "Understanding Interfirm Relationships in Business Ecosystems with Interactive Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2526–2535, Dec. 2013.

[28] Bastian, M., Heymann, S., and Jacomy, M., "Gephi: An Open Source Software for Exploring and Manipulating Networks," in *Proceedings of the Third International ICWSM Conference*, pp. 361–362, AAAI, 2009.

[29] Batagelj, V. and Mrvar, A., "Pajek Analysis and Visualization of Large Networks," in *Graph Drawing Software* (Jnger, M., Mutzel, P., Farin, G., Hege, H.-C., Hoffman, D., Johnson, C. R., Polthier, K., and Rumpf, M., eds.), Mathematics and Visualization, pp. 77–103, Springer Berlin Heidelberg, 2004.

[30] Baumgartner, J., Boerner, K., Deckard, N. J., and Sheth, N., "Poster: An XML Toolkit for an Information Visualization Software Repository.," in *Poster Compendium, IEEE Information Visualization Conference*, pp. 72–73, 2003.

[31] Bavoil, L., Callahan, S., Crossno, P., Freire, J., Scheidegger, C., Silva, C., and Vo, H., "VisTrails: enabling interactive multiple-view visualizations," in *IEEE Visualization, 2005. VIS 05*, pp. 135–142, Oct. 2005.

[32] Beard, D. V. and Walker II, J. Q., "Navigational techniques to improve the display of large two-dimensional spaces," *Behaviour & Information Technology*, vol. 9, pp. 451–466, Nov. 1990.

[33] Beaudoin, L., Parent, M.-A., and Vroomen, L., "Cheops: a compact explorer for complex hierarchies," in *Visualization '96. Proceedings.*, pp. 87–92, Oct. 1996.

[34] Becker, R., Eick, S., and Wilks, A., "Visualizing network data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, pp. 16–28, Mar. 1995.

[35] Bederson, B. B., "PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps," in *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, (New York, NY, USA), pp. 71–80, ACM, 2001.

[36] Bezerianos, A., Chevalier, F., Dragicevic, P., Elmqvist, N., and Fekete, J., "GraphDice: A System for Exploring Multivariate Social Networks," *Computer Graphics Forum*, vol. 29, pp. 863–872, June 2010.

[37] Bezerianos, A., Dragicevic, P., Fekete, J., Bae, J., and Watson, B., "GeneaQuilts: A System for Exploring Large Genealogies," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 1073–1081, Nov. 2010.

[38] Blanch, R. and Lecolinet, E., "Browsing Zoomable Treemaps: Structure-Aware Multi-Scale Navigation Techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1248–1253, Nov. 2007.

[39] Borgatti, S. P., Everett, M. G., and Freeman, L. C., "UCINet for Windows: Software for Social Network Analysis," 2002.

[40] Bostock, M. and Heer, J., "Protovis: A Graphical Toolkit for Visualization," *IEEE Trans. on Visualization and Computer Graphics*, vol. 15, pp. 1121 –1128, Dec. 2009.

[41] Bostock, M., Ogievetsky, V., and Heer, J., "D3: Data-Driven Documents," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, pp. 2301 –2309, Dec. 2011.

[42] Boutin, F. and Hascot, M., "Focus Dependent Multi-level Graph Clustering," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, (New York, NY, USA), pp. 167–170, ACM, 2004.

[43] Brandes, U. and Corman, S., "Visual unrolling of network evolution and the analysis of dynamic discourse," in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002*, pp. 145–151, 2002.

[44] Brandes, U. and Nick, B., "Asymmetric Relations in Longitudinal Social Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2283–2290, Dec. 2011.

[45] Brandes, U. and Willhalm, T., "Visualization of Bibliographic Networks with a Reshaped Landscape Metaphor," in *Proceedings of the Symposium on Data Visualisation 2002*, VISSYM '02, (Aire-la-Ville, Switzerland, Switzerland), pp. 159–ff, Eurographics Association, 2002.

[46] Branke, J., "Dynamic Graph Drawing," in *Drawing Graphs* (Kaufmann, M. and Wagner, D., eds.), no. 2025 in Lecture Notes in Computer Science, pp. 228–246, Springer Berlin Heidelberg, 2001.

[47] Burch, M. and Diehl, S., "TimeRadarTrees: Visualizing Dynamic Compound Digraphs," *Computer Graphics Forum*, vol. 27, pp. 823–830, May 2008.

[48] Burch, M., Vehlow, C., Beck, F., Diehl, S., and Weiskopf, D., "Parallel Edge Splatting for Scalable Dynamic Graph Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2344–2353, Dec. 2011.

[49] Burch, M., Beck, F., and Diehl, S., "Timeline Trees: Visualizing Sequences of Transactions in Information Hierarchies," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '08, (New York, NY, USA), pp. 75–82, ACM, 2008.

[50] Burch, M., Vehlow, C., Konevtsova, N., and Weiskopf, D., "Evaluating Partially Drawn Links for Directed Graph Edges," in *Graph Drawing* (Kreveld, M. v. and Speckmann, B., eds.), no. 7034 in Lecture Notes in Computer Science, pp. 226–237, Springer Berlin Heidelberg, Sept. 2011.

[51] Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., and Vo, H. T., "VisTrails: Visualization Meets Data Management," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, (New York, NY, USA), pp. 745–747, ACM, 2006.

[52] Card, S. K., Mackinlay, J. D., and Shneiderman, B., *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[53] Card, S. K. and Nation, D., "Degree-of-interest Trees: A Component of an Attention-reactive User Interface," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '02, (New York, NY, USA), pp. 231–245, ACM, 2002.

[54] Carpendale, M. S. T., Cowperthwaite, D. J., Fracchia, F. D., and Shermer, T., "Graph folding: Extending detail and context viewing into a tool for subgraph comparisons," in *Graph Drawing* (Brandenburg, F. J., ed.), no. 1027 in Lecture Notes in Computer Science, pp. 127–139, Springer Berlin Heidelberg, Sept. 1995.

[55] Carriere, J. and Kazman, R., "Research report. Interacting with huge hierarchies: beyond cone trees," in *Information Visualization, 1995. Proceedings.*, pp. 74–81, Oct. 1995.

[56] Centrifuge Systems, "Centrifuge." http://centrifugesystems.com/.

[57] Chen, C. and Carr, L., "Visualizing the Evolution of a Subject Domain: A Case Study," in *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, VIS '99, (Los Alamitos, CA, USA), pp. 449–452, IEEE Computer Society Press, 1999.

[58] Chen, J., MacEachren, A., and Peuquet, D., "Constructing Overview + Detail Dendrogram-Matrix Views," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 889–896, Nov. 2009.

[59] Chi, E. H., Pitkow, J., Mackinlay, J., Pirolli, P., Gossweiler, R., and Card, S. K., "Visualizing the Evolution of Web Ecologies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '98, (New York, NY, USA), pp. 400–407, ACM Press/Addison-Wesley Publishing Co., 1998.

[60] Chignell, M. H., Poblete, F., and Zuberec, S., "An Exploration in the Design Space of Three Dimensional Hierarchies," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 37, pp. 333–337, Oct. 1993.

[61] Childs, H., Brugger, E., Bonnell, K., Meredith, J., Miller, M., Whitlock, B., and Max, N., "A contract based system for large data visualization," in *Visualization, 2005. VIS 05. IEEE*, pp. 191 – 198, Oct. 2005.

[62] Chuah, M. C., Roth, S. F., Kolojejchick, J., Mattis, J., and Juarez, O., "SageBook: searching data-graphics by content," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, (New York, NY, USA), pp. 338–345, ACM Press/Addison-Wesley Publishing Co., 1995.

[63] Claessen, J. and van Wijk, J., "Flexible Linked Axes for Multivariate Data Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2310 –2316, Dec. 2011.

[64] Cruz, I. F. and Leveille, P. S., "As You Like It: Personalized Database Visualization Using a Visual Language," *Journal of Visual Languages & Computing*, vol. 12, pp. 525–549, Oct. 2001.

[65] Cui, W., Zhou, H., Qu, H., Wong, P. C., and Li, X., "Geometry-Based Edge Clustering for Graph Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1277–1284, Nov. 2008.

[66] Dachselt, R. and Ebert, J., "Collapsible cylindrical trees: a fast hierarchical navigation technique," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*, pp. 79–86, 2001.

[67] Diehl, S. and Grg, C., "Graphs, They Are Changing," in *Graph Drawing* (Goodrich, M. T. and Kobourov, S. G., eds.), no. 2528 in Lecture Notes in Computer Science, pp. 23–31, Springer Berlin Heidelberg, Aug. 2002.

[68] Dinkla, K., Westenberg, M., and van Wijk, J., "Compressed Adjacency Matrices: Untangling Gene Regulatory Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 2457–2466, Dec. 2012.

[69] Dorusz, U., Madden, B., and Madden, P., "Circular layout in the Graph Layout toolkit," in *Graph Drawing* (North, S., ed.), no. 1190 in Lecture Notes in Computer Science, pp. 92–100, Springer Berlin Heidelberg, 1997.

[70] Draper, G. and Riesenfeld, R., "Interactive Fan Charts: A Space-saving Technique for Genealogical Graph Exploration," in *Proceedings of the 8th Annual Workshop on Technology for Family History and Genealogical Research (FHTW 2008)*, 2008.

[71] Dunne, C. and Shneiderman, B., "Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, (New York, NY, USA), pp. 3247–3256, ACM, 2013.

[72] Dwyer, T., Riche, N., Marriott, K., and Mears, C., "Edge Compression Techniques for Visualization of Dense Directed Graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2596–2605, Dec. 2013.

[73] Dyer, D., "A dataflow toolkit for visualization," *IEEE Computer Graphics and Applications*, vol. 10, pp. 60 –69, July 1990.

[74] Drk, M., Carpendale, S., and Williamson, C., "Visualizing explicit and implicit relations of complex information spaces," *Information Visualization*, vol. 11, pp. 5–21, Jan. 2012.

[75] Eades, P., "Drawing Free Trees," *Bulletin of the ICA*, vol. 5, pp. 10–36, 1992.

[76] Eades, P. and Feng, Q.-W., "Multilevel visualization of clustered graphs," in *Graph Drawing* (North, S., ed.), no. 1190 in Lecture Notes in Computer Science, pp. 101–112, Springer Berlin Heidelberg, 1997.

[77] Elmqvist, N. and Fekete, J., "Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 439–454, May 2010.

[78] Elmqvist, N., Riche, Y., Henry-Riche, N., and Fekete, J., "M #x0e9;lange: Space Folding for Visual Exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 468–483, May 2010.

[79] Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., and Telea, A., "Skeleton-Based Edge Bundling for Graph Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2364–2373, Dec. 2011.

[80] Farrugia, M., Hurley, N., and Quigley, A., "Exploring Temporal Ego Networks Using Small Multiples and Tree-ring Layouts," in *ACHI 2011 : The Fourth International Conference on Advances in Computer-Human Interactions*, pp. 79–88, IARIA, 2011.

[81] Federico, P., Aigner, W., Miksch, S., Windhager, F., and Zenk, L., "A Visual Analytics Approach to Dynamic Social Networks," in *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, (New York, NY, USA), pp. 47:1–47:8, ACM, 2011.

[82] Fekete, J., Wang, D., Dang, N., Aris, A., and Plaisant, C., "Interactive poster: Overlaying graph links on treemaps," in *Proceedings of the IEEE Symposium on Information Visualization Conference Compendium (InfoVis 03)*, 2003.

[83] Fekete, J.-D., "The InfoVis Toolkit," in *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pp. 167 –174, 2004.

[84] Fekete, J.-D., Wijk, J. J. v., Stasko, J. T., and North, C., "The Value of Information Visualization," in *Information Visualization* (Kerren, A., Stasko, J. T., Fekete, J.-D., and North, C., eds.), no. 4950 in Lecture Notes in Computer Science, pp. 1–18, Springer Berlin Heidelberg, 2008.

[85] Foulser, D., "IRIS Explorer: a framework for investigation," *SIGGRAPH Comput. Graph.*, vol. 29, pp. 13–16, May 1995.

[86] Frecon, E. and Smith, G., "WEBPATH-a three dimensional Web history," in *IEEE Symposium on Information Visualization, 1998. Proceedings*, pp. 3–10, 148, Oct. 1998.

[87] Freeman, L. C., "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, pp. 35–41, Mar. 1977.

[88] Freeman, L. C., "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[89] Freire, M., Plaisant, C., Shneiderman, B., and Golbeck, J., "ManyNets: An Interface for Multiple Network Analysis and Visualization," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, (New York, NY, USA), pp. 213–222, ACM, 2010.

[90] Fruchterman, T. M. J. and Reingold, E. M., "Graph drawing by force-directed placement," *Softw: Pract. Exper.*, vol. 21, pp. 1129–1164, Nov. 1991.

[91] Fua, Y.-H., Ward, M., and Rundensteiner, E., "Navigating hierarchies with structure-based brushes," in *1999 IEEE Symposium on Information Visualization, 1999. (Info Vis '99) Proceedings*, pp. 58–64, 146, 1999.

[92] Furnas, G. W. and Zacks, J., "Multitrees: Enriching and Reusing Hierarchical Structure," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, (New York, NY, USA), pp. 330–336, ACM, 1994.

[93] Gansner, E. R. and Koren, Y., "Improved Circular Layouts," in *Graph Drawing* (Kaufmann, M. and Wagner, D., eds.), no. 4372 in Lecture Notes in Computer Science, pp. 386–398, Springer Berlin Heidelberg, Sept. 2006.

[94] Gansner, E., Hu, Y., and Kobourov, S., "GMap: Visualizing graphs and clusters as maps," in *Visualization Symposium (PacificVis), 2010 IEEE Pacific*, pp. 201–208, Mar. 2010.

[95] Gansner, E., Hu, Y., North, S., and Scheidegger, C., "Multilevel agglomerative edge bundling for visualizing large graphs," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pp. 187–194, Mar. 2011.

[96] Giacomo, E. D., Didimo, W., Liotta, G., and Palladino, P., "Visual Analysis of One-to-Many Matched Graphs," in *Graph Drawing* (Tollis, I. G. and Patrignani, M., eds.), no. 5417 in Lecture Notes in Computer Science, pp. 133–144, Springer Berlin Heidelberg, Sept. 2008.

[97] Gomez, J., Buck-Coleman, A., Plaisant, C., and Shneiderman, B., "TreeVersity: Comparing tree structures by topology and node's attributes differences," in *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 275–276, Oct. 2011.

[98] Gou, L. and Zhang, X., "TreeNetViz: Revealing Patterns of Networks over Tree Structures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2449–2458, Dec. 2011.

[99] Gove, R., Gramsky, N., Kirby, R., Sefer, E., Sopan, A., Dunne, C., Shneiderman, B., and Taieb-Maimon, M., "NetVisia: Heat Map #x0026; Matrix Visualization of Dynamic Social Network Statistics #x0026; Content," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom)*, pp. 19–26, Oct. 2011.

[100] Graham, M. and Kennedy, J., "Exploring Multiple Trees through DAG Representations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1294–1301, Nov. 2007.

[101] Greilich, M., Burch, M., and Diehl, S., "Visualizing the Evolution of Compound Digraphs with TimeArcTrees," *Computer Graphics Forum*, vol. 28, pp. 975–982, June 2009.

[102] Grivet, S., Auber, D., Domenger, J. P., and Melancon, G., "Bubble tree drawing algorithm," in *Computer Vision and Graphics* (Wojciechowski, K., Smolka, B., Palus, H., Kozera, R. S., Skarbek, W., and Noakes, L., eds.), no. 32 in Computational Imaging and Vision, pp. 633–641, Springer Netherlands, 2006.

[103] Guerra-Gomez, J., Pack, M., Plaisant, C., and Shneiderman, B., "Visualizing Change over Time Using Dynamic Hierarchies: TreeVersity2 and the StemView," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2566–2575, Dec. 2013.

[104] Hadlak, S., Schulz, H., and Schumann, H., "In Situ Exploration of Large Dynamic Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2334–2343, Dec. 2011.

[105] Hagberg, A. A., Schult, D. A., and Swart, P. J., "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, (Pasadena, CA USA), pp. 11–15, Aug. 2008.

[106] Ham, F. v., Schulz, H.-J., and Dimicco, J. M., "Honeycomb: Visual Analysis of Large Scale Social Networks," in *Human-Computer Interaction INTERACT 2009* (Gross, T., Gulliksen, J., Kotz, P., Oestreicher, L., Palanque, P., Prates, R. O., and Winckler, M., eds.), no. 5727 in Lecture Notes in Computer Science, pp. 429–442, Springer Berlin Heidelberg, Aug. 2009.

[107] Hamming, R. W., "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, pp. 147–160, Apr. 1950.

[108] Heer, J. and Boyd, D., "Vizster: visualizing online social networks," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 32–39, Oct. 2005.

[109] Heer, J., Mackinlay, J., Stolte, C., and Agrawala, M., "Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1189–1196, Nov. 2008.

[110] Heer, J. and Perer, A., "Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks," in *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 51–60, Oct. 2011.

[111] Heer, J., "Flare."

[112] Heer, J. and Card, S. K., "DOITrees Revisited: Scalable, Space-constrained Visualization of Hierarchical Data," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, (New York, NY, USA), pp. 421–424, ACM, 2004.

[113] Heer, J., Card, S. K., and Landay, J. A., "Prefuse: A Toolkit for Interactive Information Visualization," in *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems, (CHI 2005)*, (New York, NY, USA), pp. 421 –430, ACM, 2005.

[114] Heer, J. and Perer, A., "Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks," *Information Visualization*, vol. 13, pp. 111–133, Apr. 2014.

[115] Henr, N., Bezerianos, A., and Fekete, J., "Improving the Readability of Clustered Social Networks using Node Duplication," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1317–1324, Nov. 2008.

[116] Henry, N. and Fekete, J., "MatrixExplorer: a Dual-Representation System to Explore Social Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 677–684, Sept. 2006.

[117] Henry, N., Fekete, J., and McGuffin, M., "NodeTrix: a Hybrid Visualization of Social Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1302–1309, Nov. 2007.

[118] Henry, N. and Fekete, J.-D., "MatLink: Enhanced Matrix Visualization for Analyzing Social Networks," in *Human-Computer Interaction INTERACT 2007* (Baranauskas, C., Palanque, P., Abascal, J., and Barbosa, S. D. J., eds.), no. 4663 in Lecture Notes in Computer Science, pp. 288–302, Springer Berlin Heidelberg, Sept. 2007.

[119] Herman, I., Melancon, G., and Marshall, M., "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 24–43, Mar. 2000.

[120] Holten, D., "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 741–748, Sept. 2006.

[121] Holten, D., Isenberg, P., van Wijk, J., and Fekete, J., "An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pp. 195–202, Mar. 2011.

[122] Holten, D. and Van Wijk, J. J., "Visual Comparison of Hierarchically Organized Data," *Computer Graphics Forum*, vol. 27, pp. 759–766, May 2008.

[123] Holten, D. and Van Wijk, J. J., "Force-Directed Edge Bundling for Graph Visualization," *Computer Graphics Forum*, vol. 28, pp. 983–990, June 2009.

[124] Holten, D. and van Wijk, J. J., "A User Study on Visualizing Directed Edges in Graphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, (New York, NY, USA), pp. 2299–2308, ACM, 2009.

[125] Hong, J. Y., D'Andries, J., Richman, M., and Westfall, M., "Zoomology: comparing two large hierarchical trees," in *Posters Compendium of Information Visualization*, pp. 120–121, 2003.

[126] Hong, S.-H. and Murtagh, T., "Visualisation of Large and Complex Networks Using PolyPlane," in *Graph Drawing* (Pach, J., ed.), no. 3383 in Lecture Notes in Computer Science, pp. 471–481, Springer Berlin Heidelberg, Sept. 2004.

[127] Huang, M. L., Huang, T.-H., and Zhang, J., "TreemapBar: Visualizing Additional Dimensions of Data in Bar Chart," in *Information Visualisation, 2009 13th International Conference*, pp. 98–103, July 2009.

[128] Hudak, P., "Building Domain-specific Embedded Languages," *ACM Comput. Surv.*, vol. 28, Dec. 1996.

[129] Hunter, J., "Matplotlib: A 2d Graphics Environment," *Computing in Science Engineering*, vol. 9, pp. 90 –95, June 2007.

[130] Hurter, C., Telea, A., and Ersoy, O., "MoleView: An Attribute and Structure-Based Semantic Lens for Large Element-Based Plots," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2600–2609, Dec. 2011.

[131] Isenberg, P., Heimerl, F., Koch, S., Isenberg, T., Xu, P., Stolper, C., Sedlmair, M., Chen, J., Mller, T., and Stasko, J., *Visualization Publication Dataset*. 2015. Published: Dataset: http://vispubdata.org/http://vispubdata.org/\, Published Jun.\ 2015.

[132] Itoh, T., Yamaguchi, Y., Ikehata, Y., and Kajinaga, Y., "Hierarchical data visualization using a fast rectangle-packing algorithm," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, pp. 302–313, May 2004.

[133] John, M., Schulz, H., Schumann, H., Uhrmacher, A., and Unger, A., "Exploring time-varying hypergraphs," in *Poster Compendium of IEEE Conference on Information Visualization*, 2009.

[134] Johnson, B. and Shneiderman, B., "Tree-maps: a space-filling approach to the visualization of hierarchical information structures," in *, IEEE Conference on Visualization, 1991. Visualization '91, Proceedings*, pp. 284–291, Oct. 1991.

[135] Jones, E., Oliphant, T., Peterson, P., and others, "SciPy: Open source scientific tools for Python," 2001.

[136] Jusufi, I., Dingjie, Y., and Kerren, A., "The Network Lens: Interactive Exploration of Multivariate Networks Using Visual Filtering," in *Information Visualisation (IV), 2010 14th International Conference*, pp. 35–42, July 2010.

[137] Jusufi, I., Kerren, A., and Zimmer, B., "Multivariate Network Exploration with JauntyNets," in *Information Visualisation (IV), 2013 17th International Conference*, pp. 19–27, July 2013.

[138] Kamada, T. and Kawai, S., "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, pp. 7–15, Apr. 1989.

[139] Kang, H., Getoor, L., and Singh, L., "C-GROUP: A Visual Analytic Tool for Pairwise Analysis of Dynamic Group Membership," in *IEEE Symposium on Visual Analytics Science and Technology, 2007. VAST 2007*, pp. 211–212, Oct. 2007.

[140] Kang, H., Plaisant, C., Lee, B., and Bederson, B., "NetLens: Iterative Exploration of Content-Actor Network Data," in *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pp. 91–98, Oct. 2006.

[141] Kerr, B., "Thread Arcs: an email thread visualization," in *IEEE Symposium on Information Visualization, 2003. INFOVIS 2003*, pp. 211–218, Oct. 2003.

[142] Kim, N. W., Card, S. K., and Heer, J., "Tracing Genealogical Data with TimeNets," in *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, (New York, NY, USA), pp. 241–248, ACM, 2010.

[143] Kleiberg, E., van de Wetering, H., and van Wijk, J., "Botanical visualization of huge hierarchies," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*, pp. 87–94, 2001.

[144] Krzywinski, M., Birol, I., Jones, S. J., and Marra, M. A., "Hive plotsrational approach to visualizing networks," *Brief Bioinform*, vol. 13, pp. 627–644, Sept. 2012.

[145] Kumar, H. P., Plaisant, C., and Shneiderman, B., "Browsing hierarchical data with multi-level dynamic queries and pruning," *International Journal of Human-Computer Studies*, vol. 46, pp. 103–124, Jan. 1997.

[146] Lambert, A., Bourqui, R., and Auber, D., "3d Edge Bundling for Geographical Data Visualization," in *Information Visualisation (IV), 2010 14th International Conference*, pp. 329–335, July 2010.

[147] Lambert, A., Bourqui, R., and Auber, D., "Winding Roads: Routing edges into bundles," *Computer Graphics Forum*, vol. 29, pp. 853–862, June 2010.

[148] Lamping, J., Rao, R., and Pirolli, P., "A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, (New York, NY, USA), pp. 401–408, ACM Press/Addison-Wesley Publishing Co., 1995.

[149] Lee, B., Parr, C., Plaisant, C., Bederson, B., Veksler, V., Gray, W., and Kotfila, C., "TreePlus: Interactive Exploration of Networks with Enhanced Tree Layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 1414–1426, Nov. 2006.

[150] Lee, B., Robertson, G. G., Czerwinski, M., and Parr, C. S., "CandidTree: Visualizing Structural Uncertainty in Similar Hierarchies," *Information Visualization*, vol. 6, pp. 233–246, Sept. 2007.

[151] Levandowsky, M. and Winter, D., "Distance between Sets," *Nature*, vol. 234, pp. 34–35, Nov. 1971.

[152] Li, S., Crouser, R. J., Griffin, G., Gramazio, C., Schulz, H.-J., Childs, H., and Chang, R., "Exploring hierarchical visualization designs using phylogenetic trees," vol. 9397, pp. 939709–939709–14, 2015.

[153] Liu, Z., Navathe, S., and Stasko, J., "Network-based visual analysis of tabular data," in *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 41–50, Oct. 2011.

[154] Liu, Z., Navathe, S. B., and Stasko, J. T., "Ploceus: Modeling, visualizing, and analyzing tabular data as networks," *Information Visualization*, vol. 13, pp. 59–89, Jan. 2014.

[155] Lucas, B., Abram, G. D., Collins, N. S., Epstein, D. A., Gresh, D. L., and McAuliffe, K. P., "An architecture for a scientific visualization system," in *Proceedings of the 3rd conference on Visualization '92*, VIS '92, (Los Alamitos, CA, USA), pp. 107–114, IEEE Computer Society Press, 1992.

[156] Luo, S.-J., Liu, C.-L., Chen, B.-Y., and Ma, K.-L., "Ambiguity-Free Edge-Bundling for Interactive Graph Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 810–821, May 2012.

[157] L, H. and Fogarty, J., "Cascaded Treemaps: Examining the Visibility and Stability of Structure in Treemaps," in *Proceedings of Graphics Interface 2008*, GI '08, (Toronto, Ont., Canada, Canada), pp. 259–266, Canadian Information Processing Society, 2008.

[158] Mackinlay, J., Hanrahan, P., and Stolte, C., "Show Me: Automatic Presentation for Visual Analysis," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, pp. 1137 –1144, Dec. 2007.

[159] Mackinlay, J., "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, pp. 110–141, Apr. 1986.

[160] MathWorks, Inc., "Matlab." http://www.mathworks.com/products/matlab/.

[161] McGuffin, M. and Balakrishnan, R., "Interactive visualization of genealogical graphs," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 16–23, Oct. 2005.

[162] McGuffin, M., Davison, G., and Balakrishnan, R., "Expand-Ahead: A Space-Filling Strategy for Browsing Trees," in *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pp. 119–126, 2004.

[163] McGuffin, M. and Jurisica, I., "Interaction Techniques for Selecting and Manipulating Subgraphs in Network Visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 937–944, Nov. 2009.

[164] Microsoft Corp., "Excel." http://products.office.com/en-US/excel.

[165] Microsoft Corp., "Visio." http://products.office.com/en-us/visio/.

[166] Moscovich, T., Chevalier, F., Henry, N., Pietriga, E., and Fekete, J.-D., "Topology-aware Navigation in Large Networks," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, (New York, NY, USA), pp. 2319–2328, ACM, 2009.

[167] Muelder, C. and Ma, K.-L., "Rapid Graph Layout Using Space Filling Curves," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1301–1308, Nov. 2008.

[168] Muelder, C. and Ma, K.-L., "A Treemap Based Method for Rapid Layout of Large Graphs," in *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pp. 231–238, Mar. 2008.

[169] Munzner, T., "H3: laying out large directed graphs in 3d hyperbolic space," in *, IEEE Symposium on Information Visualization, 1997. Proceedings*, pp. 2–10, Oct. 1997.

[170] Netzel, R., Burch, M., and Weiskopf, D., "Comparative Eye Tracking Study on Node-Link Visualizations of Trajectories," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 2221–2230, Dec. 2014.

[171] Neumann, P., Schlechtweg, D. S., and Carpendale, S., "ArcTrees: Visualizing Relations in Hierarchical Data," in *EUROVIS 2005: Eurographics / IEEE VGTC Symposium on Visualization* (Brodlie, K., Duke, D., and Joy, K., eds.), The Eurographics Association, 2005.

[172] Nocaj, A. and Brandes, U., "Organizing Search Results with a Reference Map," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 2546–2555, Dec. 2012.

[173] O'Madadhain, J., Fisher, D., Smyth, P., White, S., and Boey, Y.-B., "Analysis and Visualization of Network Data Using JUNG," *Journal of Statistical Software*, vol. 10, no. 2, pp. 1 –35, 2005.

[174] Ortiz, S. and Cid, V. P., "Use cases of Impure, an information interface," in *VisWeek 2010 Discovery Exhibition*, (Salt Lake City, UT), 2010.

[175] Osawa, N., "A multiple-focus graph browsing technique using heat models and force-directed layout," in *Fifth International Conference on Information Visualisation, 2001. Proceedings*, pp. 277–283, 2001.

[176] Page, L., Brin, S., Motwani, R., and Winograd, T., "The PageRank Citation Ranking: Bringing Order to the Web.," Technical Report 1999-66, Stanford InfoLab, Nov. 1999. Previous number = SIDL-WP-1999-0120.

[177] Paiva, J., Florian, L., Pedrini, H., Telles, G., and Minghim, R., "Improved Similarity Trees and their Application to Visual Data Classification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2459–2468, Dec. 2011.

[178] Parker, S. and Johnson, C., "SCIRun: A Scientific Programming Environment for Computational Steering," in *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, p. 52, 1995.

[179] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[180] Pirolli, P., Card, S. K., and Van Der Wege, M. M., "The Effects of Information Scent on Visual Search in the Hyperbolic Tree Browser," *ACM Trans. Comput.-Hum. Interact.*, vol. 10, pp. 20–53, Mar. 2003.

[181] Plaisant, C., Grosjean, J., and Bederson, B., "SpaceTree: supporting exploration in large node link tree, design evolution and empirical evaluation," in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002*, pp. 57–64, 2002.

[182] Pretorius, A. J. and Van Wijk, J. J., "Visual Inspection of Multivariate Graphs," *Computer Graphics Forum*, vol. 27, pp. 967–974, May 2008.

[183] Pretorius, A. and van Wijk, J., "Visual Analysis of Multivariate State Transition Graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 685–692, Sept. 2006.

[184] Reda, K., Tantipathananandh, C., Johnson, A., Leigh, J., and Berger-Wolf, T., "Visualizing the Evolution of Community Structures in Dynamic Social Networks," *Computer Graphics Forum*, vol. 30, pp. 1061–1070, June 2011.

[185] Reingold, E. M. and Tilford, J., "Tidier Drawings of Trees," *IEEE Transactions on Software Engineering*, vol. SE-7, pp. 223–228, Mar. 1981.

[186] Riche, N. H., Dwyer, T., Lee, B., and Carpendale, S., "Exploring the Design Space of Interactive Link Curvature in Network Diagrams," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, (New York, NY, USA), pp. 506–513, ACM, 2012.

[187] Robertson, G. G., Mackinlay, J. D., and Card, S. K., "Cone Trees: Animated 3d Visualizations of Hierarchical Information," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, (New York, NY, USA), pp. 189–194, ACM, 1991.

[188] Rost, U. and Bornberg-Bauer, E., "TreeWiz: interactive exploration of huge trees," *Bioinformatics*, vol. 18, pp. 109–114, Jan. 2002.

[189] Rosvall, M. and Bergstrom, C. T., "Mapping Change in Large Networks," *PLoS ONE*, vol. 5, p. e8694, Jan. 2010.

[190] Roth, S. F., Kolojejchick, J., Mattis, J., Chuah, M. C., Goldstein, J., and Juarez, O., "SAGE tools: a knowledge-based environment for designing and perusing data visualizations," in *Conference companion on Human factors in computing systems*, CHI '94, (New York, NY, USA), pp. 27–28, ACM, 1994.

[191] Rufiange, S. and McGuffin, M., "DiffAni: Visualizing Dynamic Graphs with a Hybrid of Difference Maps and Animation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2556–2565, Dec. 2013.

[192] Santos, E., Lins, L., Ahrens, J., Freire, J., and Silva, C., "VisMashup: Streamlining the Creation of Custom Visualization Applications," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, pp. 1539 –1546, Dec. 2009.

[193] Saraiya, P., Lee, P., and North, C., "Visualization of graphs with associated timeseries data," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 225–232, Oct. 2005.

[194] Sarkar, M. and Brown, M. H., "Graphical Fisheye Views of Graphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, (New York, NY, USA), pp. 83–91, ACM, 1992.

[195] Satyanarayan, A., Russell, R., Hoffswell, J., and Heer, J., "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, pp. 659–668, Jan. 2016.

[196] Satyanarayan, A. and Heer, J., "Lyra: An Interactive Visualization Design Environment," *Computer Graphics Forum*, vol. 33, no. 3, pp. 351–360, 2014.

[197] Schroeder, W., Avila, L., Martin, K., Hoffman, W., and Law, C., *The Visualization Toolkit-Users Guide*. Kitware, Inc., 2001.

[198] Schulz, H. J., Hadlak, S., and Schumann, H., "The Design Space of Implicit Hierarchy Visualization: A Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 393–411, Apr. 2011.

[199] Schulz, H.-J., John, M., Unger, A., and Schumann, H., "Visual Analysis of Bipartite Biological Networks," in *Eurographics Workshop on Visual Computing for Biomedicine* (Botha, C., Kindlmann, G., Niessen, W., and Preim, B., eds.), The Eurographics Association, 2008.

[200] Selassie, D., Heller, B., and Heer, J., "Divided Edge Bundling for Directional Network Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 2354–2363, Dec. 2011.

[201] Shamir, A. and Stolpnik, A., "Interactive visual queries for multivariate graphs exploration," *Computers & Graphics*, vol. 36, pp. 257–264, June 2012.

[202] Sharara, H., Sopan, A., Namata, G., Getoor, L., and Singh, L., "G-PARE: A visual analytic tool for comparative analysis of uncertain graphs," in *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 61–70, Oct. 2011.

[203] Shi, L., Wang, C., and Wen, Z., "Dynamic network visualization in 1.5d," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pp. 179–186, Mar. 2011.

[204] Shneiderman, B. and Aris, A., "Network Visualization by Semantic Substrates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 733–740, Sept. 2006.

[205] Siirtola, H. and Mkinen, E., "Constructing and Reconstructing the Reorderable Matrix," *Information Visualization*, vol. 4, pp. 32–48, Mar. 2005.

[206] Singhal, A., "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[207] Six, J. M. and Tollis, I. G., "A Framework for Circular Drawings of Networks," in *Graph Drawing* (Kratochvyl, J., ed.), no. 1731 in Lecture Notes in Computer Science, pp. 107–116, Springer Berlin Heidelberg, Sept. 1999.

[208] Skau, D. and Kosara, R., "Interactive Poster: Designing New Visualizations from Scratch without Programming," in *InfoVis 2011 Poster Compendium*, 2011.

[209] Smith, M. A., Shneiderman, B., Milic-Frayling, N., Mendes Rodrigues, E., Barash, V., Dunne, C., Capone, T., Perer, A., and Gleave, E., "Analyzing (Social Media) Networks with NodeXL," in *Proc. of the Fourth International Conference on Communities and Technologies, (C&T '09)*, (New York, NY, USA), pp. 255 –264, ACM, 2009.

[210] Sokal, R. R. and Michener, C., "A statistical method for evaluating systematic relationships," *Univ Kans Sci Bull*, vol. 38, pp. 1409–1438, 1958.

[211] Sokal, R. R. and Rohlf, F. J., "The Comparison of Dendrograms by Objective Methods," *Taxon*, vol. 11, no. 2, pp. 33–40, 1962.

[212] Stasko, J. and Zhang, E., "Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations," in *IEEE Symposium on Information Visualization, 2000. InfoVis 2000*, pp. 57–65, 2000.

[213] Stasko, J., Choo, J., Hu, M., Pileggi, H., Sadana, R., and Stolper, C. D., "Poster: Citevis: Exploring Conference Paper Citation Data Visually," in *IEEE 2013 Infovis Poster Compendium*, 2013.

[214] Stasko, J., G\"{o}rg, C., and Liu, Z., "Jigsaw: Supporting Investigative Analysis through Interactive Visualization," *Information Visualization*, vol. 7, pp. 118–132, June 2008.

[215] Stein, K., Wegener, R., and Schlieder, C., "Pixel-Oriented Visualization of Change in Social Networks," in *2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 233–240, Aug. 2010.

[216] Stolper, C., Kahng, M., Lin, Z., Foerster, F., Goel, A., Stasko, J., and Chau, D., "GLO-STIX: Graph-Level Operations for Specifying Techniques and Interactive eXploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 2320–2328, Dec. 2014.

[217] Stolte, C., Tang, D., and Hanrahan, P., "Polaris: a System for Query, Analysis, and Visualization of Multidimensional Relational Databases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52 –65, 2002.

[218] Sugiyama, K., Tagawa, S., and Toda, M., "Methods for Visual Understanding of Hierarchical System Structures," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, pp. 109–125, Feb. 1981.

[219] Tableau Software, "Tableau." http://www.tableausoftware.com/.

[220] Tan, D., Smith, G., Lee, B., and Robertson, G., "AdaptiviTree: Adaptive Tree Visualization for Tournament-Style Brackets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1113–1120, Nov. 2007.

[221] Telea, A. and Ersoy, O., "Image-Based Edge Bundles: Simplified Visualization of Large Graphs," *Computer Graphics Forum*, vol. 29, pp. 843–852, June 2010.

[222] Teoh, S. T. and Kwan-Liu, M., "RINGS: A Technique for Visualizing Large Hierarchies," in *Graph Drawing* (Goodrich, M. T. and Kobourov, S. G., eds.), no. 2528 in Lecture Notes in Computer Science, pp. 268–275, Springer Berlin Heidelberg, Aug. 2002.

[223] Tibco Software, "Spotfire." http://spotfire.tibco.com/.

[224] TouchGraph, LLC, "TouchGraph Navigator 2." http://www.touchgraph.com/navigator.

[225] Trifacta Inc., "Vega." http://trifacta.github.io/vega/.

[226] Tsiaras, V., Triantafilou, S., and Tollis, I. G., "Treemaps for Directed Acyclic Graphs," in *Graph Drawing* (Hong, S.-H., Nishizeki, T., and Quan, W., eds.), no. 4875 in Lecture Notes in Computer Science, pp. 377–388, Springer Berlin Heidelberg, Sept. 2007.

[227] Tu, Y. and Shen, H.-W., "Visualizing Changes of Hierarchical Data using Treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1286–1293, Nov. 2007.

[228] Tu, Y. and Shen, H.-W., "Balloon Focus: a Seamless Multi-Focus+Context Method for Treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1157–1164, Nov. 2008.

[229] Tufte, E. R., *Envisioning information*. Cheshire, Conn.: Graphics Press, 1995.

[230] Tuttle, C., Nonato, L., and Silva, C., "PedVis: A Structured, Space-Efficient Technique for Pedigree Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 1063–1072, Nov. 2010.

[231] Upson, C., Faulhaber, T.A., J., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., and van Dam, A., "The application visualization system: a computational environment for scientific visualization," *Computer Graphics and Applications, IEEE*, vol. 9, pp. 30 –42, July 1989.

[232] van den Elzen, S. and van Wijk, J., "Multivariate Network Exploration and Presentation: From Detail to Overview via Selections and Aggregations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 2310–2319, Dec. 2014.

[233] van Ham, F., van de Wetering, H., and van Wijk, J., "Visualization of state transition graphs," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*, pp. 59–66, 2001.

[234] van Ham, F. and van Wijk, J., "Beamtrees: compact visualization of large hierarchies," in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002*, pp. 93–100, 2002.

[235] van Ham, F., Wattenberg, M., and Viegas, F., "Mapping Text with Phrase Nets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1169–1176, Nov. 2009.

[236] van Wijk, J. and van de Wetering, H., "Cushion treemaps: visualization of hierarchical information," in *1999 IEEE Symposium on Information Visualization, 1999. (Info Vis '99) Proceedings*, pp. 73–78, 147, 1999.

[237] Viegas, F., Wattenberg, M., van Ham, F., Kriss, J., and McKeon, M., "ManyEyes: a Site for Visualization at Internet Scale," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, pp. 1121 –1128, Dec. 2007.

[238] Voigt, M., Pietschmann, S., Grammel, L., and Meiner, K., "Context-aware Recommendation of Visualization Components," pp. 101–109, Jan. 2012.

[239] Voorhees, E. M., "Implementing agglomerative hierarchic clustering algorithms for use in document retrieval," *Information Processing & Management*, vol. 22, pp. 465–476, Jan. 1986.

[240] W3C, "SVG Specification."

[241] Wang, W., Wang, H., Dai, G., and Wang, H., "Visualization of Large Hierarchical Data by Circle Packing," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, (New York, NY, USA), pp. 517–520, ACM, 2006.

[242] Ware, C. and Franck, G., "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions," *ACM Trans. Graph.*, vol. 15, pp. 121–140, Apr. 1996.

[243] Wattenberg, M., "Arc Diagrams: Visualizing Structure in Strings," in *Proc. of IEEE Infovis 2002*, pp. 110 –116, 2002.

[244] Wattenberg, M., "Visual Exploration of Multivariate Graphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, (New York, NY, USA), pp. 811–819, ACM, 2006.

[245] Wickham, H., *ggplot2 Elegant Graphics for Data Analysis*. Use R!, New York, NY, USA: Springer, 2009. DOI: 10.1007/978-0-387-98141-3.

[246] Wickham, H., "A Layered Grammar of Graphics," *Journal of Computational and Graphical Statistics*, vol. 19, no. 1, pp. 3–28, 2010.

[247] Wilkinson, L., *The grammar of graphics*. New York: Springer, 2005.

[248] Wills, G., "Selection: 524,288 ways to say ldquo;this is interesting rdquo;," in *Proceedings IEEE Symposium on Information Visualization '96*, pp. 54–60, 120, Oct. 1996.

[249] Wong, N., Carpendale, S., and Greenberg, S., "Edgelens: an interactive method for managing edge congestion in graphs," in *IEEE Symposium on Information Visualization, 2003. INFOVIS 2003*, pp. 51–58, Oct. 2003.

[250] Wong, N. and Carpendale, S., "Supporting interactive graph exploration using edge plucking," vol. 6495, pp. 649508–649508–12, 2007.

[251] Wong, P. C., Foote, H., Mackey, P., Chin, G., Sofia, H., and Thomas, J., "A Dynamic Multiscale Magnifying Tool for Exploring Large Sparse Graphs," *Information Visualization*, vol. 7, pp. 105–117, June 2008.

[252] Wong, P. C., Mackey, P., Perrine, K., Eagan, J., Foote, H., and Thomas, J., "Dynamic visualization of graphs with extended labels," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 73–80, Oct. 2005.

[253] Wongsuphasawat, K., Moritz, D., Anand, A., Mackinlay, J., Howe, B., and Heer, J., "Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, pp. 649–658, Jan. 2016.

[254] Wood, J. and Dykes, J., "Spatially Ordered Treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 1348–1355, Nov. 2008.

[255] Wood, J., Isenberg, P., Isenberg, T., Dykes, J., Boukhelifa, N., and Slingsby, A., "Sketchy Rendering for Information Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 2749–2758, Dec. 2012.

[256] Woodruff, A., Olston, C., Aiken, A., Chu, M., Ercegovac, V., Lin, M., Spalding, M., and Stonebraker, M., "DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data," *J. of Visual Languages & Computing*, vol. 12, pp. 551–571, Oct. 2001.

[257] Xiao, L., Yeh, R., and Hanrahan, P., "Flow map layout," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 219–224, Oct. 2005.

[258] Xiong, R. and Brittain, E., "LiveWeb: Visualizing Live User Activities on the Web," in *ACM SIGGRAPH 99 Conference Abstracts and Applications*, SIGGRAPH '99, (New York, NY, USA), pp. 254–, ACM, 1999.

[259] Yi, J. S., Elmqvist, N., and Lee, S., "TimeMatrix: Analyzing Temporal Social Networks Using Interactive Matrix-Based Visualizations," *International Journal of Human-Computer Interaction*, vol. 26, pp. 1031–1051, Nov. 2010.

[260] Zhao, S., McGuffin, M., and Chignell, M., "Elastic hierarchies: combining treemaps and node-link diagrams," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*, pp. 57–64, Oct. 2005.

[261] Zhou, H., Yuan, X., Cui, W., Qu, H., and Chen, B., "Energy-Based Hierarchical Edge Clustering of Graphs," in *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, pp. 55–61, Mar. 2008.

[262] Ziegler, L., Kunz, C., Botsch, V., and Schneeberger, J., "Visualizing and exploring large networked information spaces with matrix browser," in *Sixth International Conference on Information Visualisation, 2002. Proceedings*, pp. 361–366, 2002.