GLO-STIX: Graph-Level Operations for Specifying Techniques and Interactive eXploration

Charles D. Stolper, Minsuk Kahng, Zhiyuan Lin, Florian Foerster, Aakash Goel, John Stasko, and Duen Horng Chau
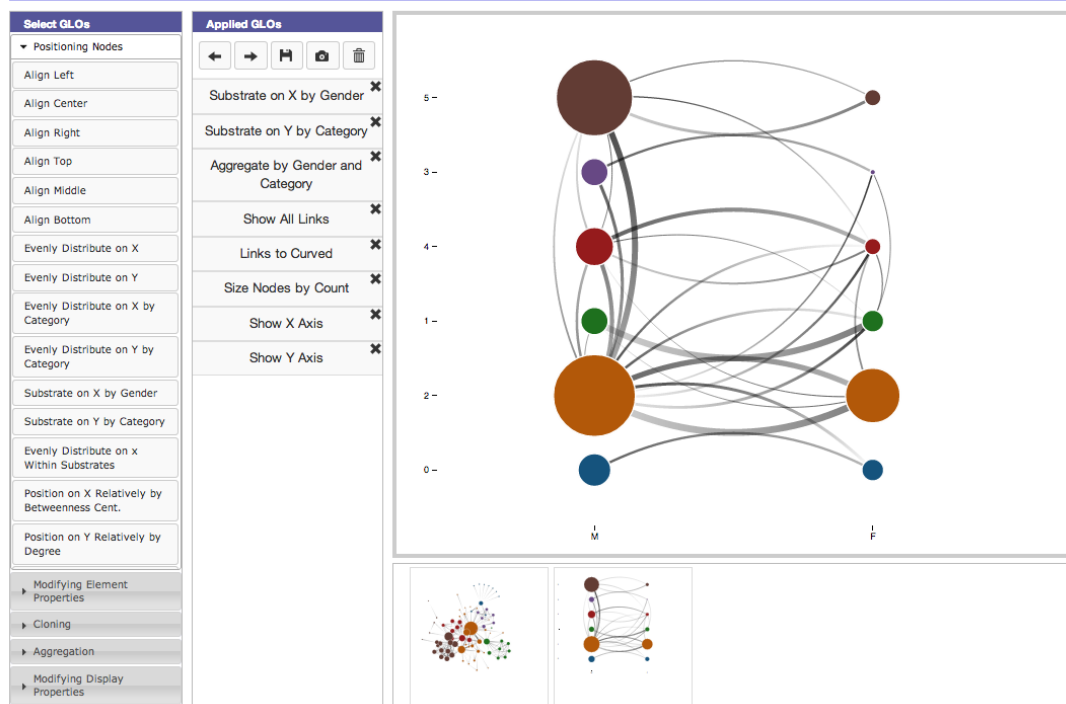
Fig. 1: A screenshot of the GLO-STIX user interface showing a user exploring the Les Misérables character co-occurrence graph using graph-level operations (GLOs). Nodes are characters, and an edge connects two characters if they co-occur in a chapter. The original node-link view of the graph is saved by the user as a snapshot in the bottom pane. From the list of operations available (shown in left-most column), applying those selected in the middle column transforms the original graph into the PivotGraph visualization [30] displayed in the main view. All graph figures in this paper were generated using GLO-STIX.

Abstract— The field of graph visualization has produced a wealth of visualization techniques for accomplishing a variety of analysis tasks. Therefore analysts often rely on a suite of different techniques, and visual graph analysis application builders strive to provide this breadth of techniques. To provide a holistic model for specifying network visualization techniques (as opposed to considering each technique in isolation) we present the *Graph-Level Operations* (*GLO*) model. We describe a method for identifying GLOs and apply it to identify five classes of GLOs, which can be flexibly combined to re-create six canonical graph visualization techniques. We discuss advantages of the GLO model, including potentially discovering new, effective network visualization techniques and easing the engineering challenges of building multi-technique graph visualization applications. Finally, we implement the GLOs that we identified into the GLO-STIX prototype system that enables an analyst to interactively explore a graph by applying GLOs.

Index Terms—Graph-level operations, graph visualization, visualization technique specification, graph analysis, information visualization.

◆

- *Charles D. Stolper is with the College of Computing, Georgia Institute of Technology. E-mail: chadstolper@gatech.edu.*
- *Minsuk Kahng is with the College of Computing, Georgia Institute of Technology. E-mail: kahng@gatech.edu.*
- *Zhiyuan Lin is with the College of Computing, Georgia Institute of Technology. E-mail: zlin48@gatech.edu.*
- *Florian Foerster is with the College of Computing, Georgia Institute of Technology. E-mail: florian.foerster@gatech.edu.*
- *Aakash Goel is with the College of Computing, Georgia Institute of Technology. E-mail: aakashgoel@gatech.edu.*
- *John Stasko is with the the College of Computing, Georgia Institute of Technology. E-mail: stasko@cc.gatech.edu.*
- *Duen Horng Chau is with the College of Computing, Georgia Institute of Technology. E-mail: polo@gatech.edu.*

1 INTRODUCTION

The field of graph visualization has provided analysts with a number of useful techniques for displaying the nodes and edges of a graph. Each of these techniques can be quite effective at showing aspects of the graph to the analyst. In other words, different techniques are effective at accomplishing different tasks. When analysts wish to perform multiple tasks, they often turn to multiple graph visualization techniques. Developers of graph visualization systems, in turn, must implement this variety of techniques in their applications.

We introduce the new idea of **graph-level operations** (GLOs), which provides an alternative to implementing each graph visualization technique in isolation. GLOs are encapsulated manipulations of a graph visualization. Let us consider an example GLO: *positioning each node's glyph relatively on an axis according to a continuous attribute of the node*. This GLO might be used to stratify the nodes according to their node types, as in a *semantic substrates* visualization [22], which places nodes in non-overlapping regions, one region for each node type, to help reduce visual complexity (see Table 1, fourth visualization). The same GLO may also be used to position nodes into a 2D grid, each dimension corresponds to a node attribute, as in a PivotGraph [30], which is designed for summarizing multivariate graphs (see Table 1, fifth visualization). It could further be applied to align the nodes into a scatterplot.

In fact, we can consider graph visualization techniques as sequences of GLOs. For example, the aforementioned PivotGraph technique can be represented as:

- Substrate Nodes on *x* by *attribute0*
- Substrate Nodes on *y* by *attribute1*
- Aggregate Nodes by *attribute0* and *attribute1*
- Display All Links
- Display Links as Curved
- Size Nodes by Count
- Show *x* Axis
- Show *y* Axis
- Set Target Generation 1
- Set Source Generation 1

This means that through flexible combinations of GLOs, we may create and specify familiar canonical visualization techniques and potentially create and specify new ones.

Our Contributions. Our work makes multiple major contributions:

- We present GLOs (Graph-Level Operations) — a new idea and model for specifying graph visualization techniques. We contribute a method to identify GLOs and demonstrate the flexibility of GLOs in re-creating canonical graph visualization techniques.

- We discuss the important properties of GLOs, and the benefits to graph visualization that GLOs provide, such as: (1) a new model of graph exploration; (2) the potential to discover new network visualization techniques; and (3) a reduced engineering challenge for graph visualization application developers.

- We demonstrate the feasibility and potential of the GLO approach through an implementation of each of the GLOs within GLO-STIX prototype system.

Notation. Throughout this paper we use the terms *graph* and *network* interchangeably to refer to data structures of *nodes* (or *vertices*) connected by *edges* (or *links*). We use the term *technique* or *visualization technique* to refer to methods of displaying data and *graph visualization techniques* or *network visualization techniques* to refer to methods of displaying graph data. *Visualization elements* (or *glyphs*) are the on-screen graphical representations of data such as circle elements representing nodes. Finally, we use the terms *graph-level operations*, *operations*, and *GLOs* interchangeably to refer to encapsulated manipulations of these visualization elements.

Paper Organization. This paper provides a significantly more detailed and rigorous description of graph-level operations than our preliminary work-in-progress extended abstract [24]. Section 2 presents repeatable methods for identifying GLOs provided a set of canonical graph visualization techniques and for specifying techniques using GLOs. Section 3 describes three advantages of graph-level operations over alternative network visualization technique specification models. Section 4 provides descriptions of GLOs that we identified by applying the method of Section 2 to the set of six graph canonical graph visualization techniques in Table 1. Section 5 presents general properties of GLOs. Section 6 presents the design rationale and implementation of GLO-STIX, an application for exploring a graph using GLOs and implementing the GLOs in Section 4. Section 7 discusses related work in graph visualization programming toolkits and graphical tools for graph analysis. We conclude in Sections 8 with a discussion of the limitations of the current GLO-STIX implementation and future research directions for the Graph-Level Operations model.

2 IDENTIFYING GRAPH-LEVEL OPERATIONS

In this section, we describe the method that we used to identify the graph-level operations necessary to specify a set of canonical graph visualization techniques. We then describe how we can use these GLOs to specify these techniques, or others.

2.1 Identifying GLOs

Graph-level operations are encapsulated manipulations of a graph visualization. This encapsulation introduces uncertainty as to the appropriate level of abstraction for each operation. For example, is positioning a node relatively along the *x* and *y* axes one operation or two? In order to mitigate this uncertainty, in this section we describe a repeatable method for identifying a set of operations necessary for specifying a given a set of techniques.

We began by choosing a set of techniques that were commonly used within the graph visualization community and provided a cross-section of network visualization methods. We settled on the six techniques shown in Table 1. We then, as a team, described operations necessary to transition from each technique to each other technique. Two example transitions are demonstrated in Figure 2. During this process, we took a "worst-case" approach and assumed that any variable parameters would be different. For example, when determining the operations required to transition from a *semantic substrates* visualization to a *PivotGraph* visualization, we assumed that the substrated attribute was not also one of the pivot attributes.

Having identified all of the transitions between each of the techniques, we collected each of the different operations used into a set. We then augmented this set with obvious parallel operations. For example, "Align *y* Top" never appears in Table 1, but given "Align *y* Bottom" does, it is logical to include this operation in the set. The collection of all of these operations represented the final set of of GLOs. This process allowed us to reproducibly grow our set of GLOs by iteratively expanding the number of visualization techniques to support. This resulted in the set of 34 graph-level operations (see Section 4). Finally, we classified the operations using *card sorting* [21], a technique to categorize and group information. We printed all GLOs on small cards and had three visualization experts sort the GLOs into categories and give them names. Subsequently we analyzed the groupings and combined them into multiple categories of GLOs. We note that while these GLO categories provide good coverage, they are by no means exhaustive.
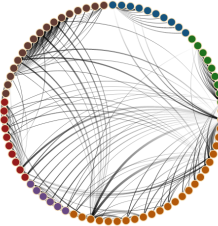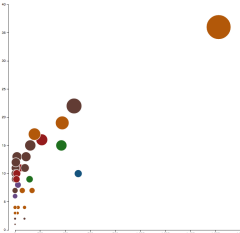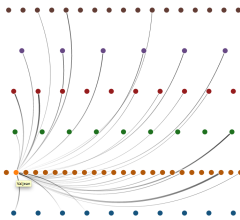
More formally: consider a set of techniques T. Let $S_{t_i \to t_j}$ be a set of operations necessary to transition from technique $t_i \in T$ to technique $t_j \in T$. Thus, the set of GLOs necessary to specify T (denoted O_T) can be defined as

$$O_T := \bigcup S_{t_i \to t_j}, \forall t_i, t_j \in T \qquad (1)$$

2.2 Specifying Techniques

We then set out to reverse this method, and use the GLOs that we identified to specify the techniques. We can do so by listing the GLOs

Table 1: The six canonical graph visualization techniques from which we derived the set of graph-level operations. For each technique, we include its name (left column), a screenshot of its implementaion using our GLO-STIX prototype system (center column), and the sequence of GLOs used (right column). We show in **bold** those GLOs requiring before or after dependencies. Those GLOs shown in *italics* must come before or after (though need not come immediately before or after) the bolded GLO.

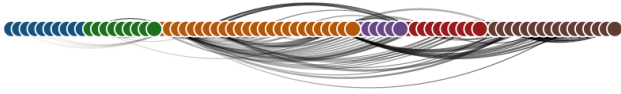| Technique Name	GLO-STIX Implementation	GLO Specification
Force-Directed Diagram		Apply Force-Directed Algorithm Display Links as Straight Size Nodes Relatively by *attribute* Display All Links Hide *x* Axis Hide *y* Axis Set Target Generation 0 Set Source Generation 0
Circle Plot		Size Nodes by Constant Position Nodes Along Plot Radius by {constant} Evenly Distribute Nodes Radially by *attribute* Display All Links Display Links as Curved Hide *x* Axis Hide *y* Axis Set Target Generation 0 Set Source Generation 0
Scatterplot		Posn. Nodes Relatively on *x* by *attribute0* Posn. Nodes Relatively on *y* by *attribute1* Show *x* Axis Show *y* Axis Hide Links Size Nodes Relatively by *attribute* Set Target Generation 0 Set Source Generation 0
Semantic Substrates [22]		**Substrate Nodes on *y* by *attribute1*** *Evenly Distribute Nodes on x Within Substrates* Display Selected Links Size Nodes by Constant Display Links as Curved Display All Links Hide *x* Axis Hide *y* Axis Set Target Generation 0 Set Source Generation 0
PivotGraph [30]		Substrate Nodes on *x* by *attribute0* Substrate Nodes on *y* by *attribute1* **Aggregate Nodes by *attribute0* and *attribute1*** *Size Nodes by Count* Show *x* Axis Show *y* Axis Display Links as Curved Display All Links *Set Target Generation 1* *Set Source Generation 1*
Adjacency Matrix		Size Nodes by Constant *Evenly Distribute Nodes on y* *Align Nodes Left* **Clone Active Generation** *Evenly Distribute Nodes on x* *Align Nodes Bottom* *Set Target Generation 1* Display Links as Circles Display All Links

Fig. 3: Arc diagram [29] created using the following sequence of GLOs: Size Nodes by Constant, Align Nodes Middle, Evenly Distribute Nodes on *x* by category, Display All Links, Display Links as Curved, Hide *x* Axis, Hide *y* Axis, Set Target Generation 0, Set Source Generation 0 .

necessary to transition to them. As we already have a list of operations for transitions between any two techniques, we can specify a technique by combining these sequences of GLOs for transitions to the given technique from all the other techniques. Ideally, the order in which GLOs are applied would not matter, and in some cases this is true. For instance, the results are the same regardless of the order "Position Relatively on x by attribute0" or "Size Nodes Relatively by attribute1" are applied for a transition to a scatterplot. However, in certain instances the GLO ordering does matter, namely some GLOs must be applied after a certain GLO is applied. This case occurs because of a construct we term *generations*. Notice that in the adjacency matrix in Table 1 there are two sets of node elements: those on the left and those on the bottom. Each set contains an element representating each node in the dataset. The set on the bottom was *generated* from the set on the left through the "Clone Active Generation" GLO. This GLO made a precise copy of the current *active generation* (in this case evenly-distributed on y, constantly-sized, left-aligned circles) and made the newly created elements (the new *generation*) into the active generation. The active generation is used to set the generation parameter of a GLO, described in Section 5, but in plain terms all GLOs are applied to only the active generation. Returning to the adjacency matrix, "Align Bottom" had to be applied after "Clone Active Generation" else it will be applied to the circles on the left, instead of a new set of circles.

For each technique we can specify a list of dependence (preceding) relationships between two GLOs denoted $o_a \prec o_b$, signifying that o_a must precede o_b. We then represent the technique, O_t, as a sequence of GLOs where each GLO is from the union of all of the sets of GLOs used to transition to the technique, and the order of GLOs satisfy all dependence conditions. Resulting sequences from this process for the six chosen techniques can be found in the GLO Specification column of Table 1. Applying a technique's sequence of operations results in a visualization of the technique on the active generation.

3 BENEFITS OF GRAPH-LEVEL OPERATIONS

Here we describe benefits the graph-level operations model provides as a holistic model of graph visualization. We look at two cases where the GLO model provides either additional capabilities or explicit advantages over existing approaches. We begin by discussing a new style of interactive graph exploration enabled by GLOs and how this could lead to the discovery of new, effective network visualization techniques. We then discuss the alternative coding paradigm that GLOs provide developers and the advantages of this paradigm.

3.1 Graph Exploration and Discovering New Techniques

One of the more interesting applications of graph-level operations is using them to interactively explore a graph without limiting an analyst to only supported techniques. As we mentioned in the introduction, different techniques support different graph analysis tasks. There is a chance, however, that the best view of the data is not one explicitly provided by the visualization application. By iteratively applying GLOs, an analyst can see different views onto his or her graph data beyond those predefined by an application's developer. This balance
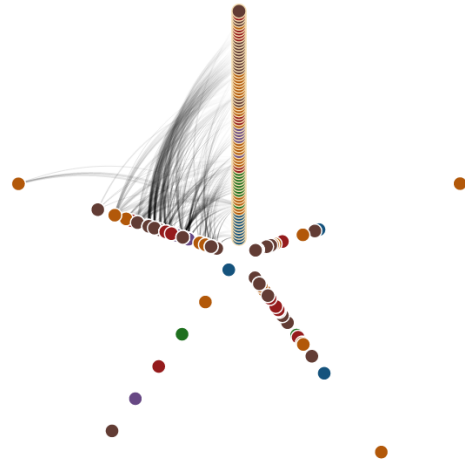

Fig. 4: Multi-dimensional technique akin to a hive plot [17] or star diagram [16]. Clockwise from the top, the chart visualizes the id, betweenness-centrality, page-rank, category, and degree attributes of the nodes. Edges are drawn from a single axis to a single other axis. The plot is specified by the following sequence of GLOs: Size Nodes by Constant, Clone Active Generation, Clone Active Generation, Clone Active Generation, Clone Active Generation, Position Nodes Radially by generation, Position Nodes Along Plot Radius by degree, Select Generation 3, Position Nodes Radially by generation, Substrate Nodes Along Plot Radius by category, Select Generation 2, Position Nodes Radially by generation, Position Nodes Along Plot Radius by page-rank, Select Generation 1, Position Nodes Radially by generation, Position Nodes Along Plot Radius by betweenness-centrality, Select Generation 0, Position Nodes Radially by generation, Evenly Distribute Nodes Along Plot Radius, Display All Links, Display Links as Curved, Set Source Generation 4, Set Target Generation 0, Hide *x* Axis, Hide *y* Axis .

allows the application developer to still specify common techniques, but empowers the analyst with the freedom to experiment with different displays.

Not only do GLOs provide a benefit to analysts by allowing them to explore their data in a more free-form manner, but they provide a benefit to the visualization community by potentially providing a novel method of identifying new techniques. Let us use a simple example to demonstrate this point. Suppose that we had built a system that supported the six techniques in Table 1 and allowed an analyst to apply GLOs freely to adjust the display. Let us say that the analyst were to distribute the nodes evenly on the *x* axis (an adjacency matrix GLO), then position the nodes in the middle of the screen (an augmented GLO from the adjacency matrix), and then choose to display the links as curved (a GLO used by semantic substrates). By applying these three GLOs, the analyst has successfully recreated the arc diagram technique [29] in Figure 3. In other words, the analyst was able to not only explore his or her graph but was able to 'discover' a new graph visualization technique in the process.

For a more complex example, consider the technique in Figure 4. This technique required creating four (for five total) generations of nodes, plotting each generation radially by generation, and then positioning each generation along the plot radius by different attributes. Edges can then be drawn between any two sets of axes using the source-generation and target-generation GLOs. The resulting chart is akin to Hive Plots [17] (where edges are drawn between nodes on different radial axes) or star diagrams [16] (where elements are positioned along multiple radial attribute axes). An analyst may not serendipitously encounter this precise technique, but he or she may encounter aspects of the technique such as positioning radially by generation or positioning different generations by different attributes along the same axis.

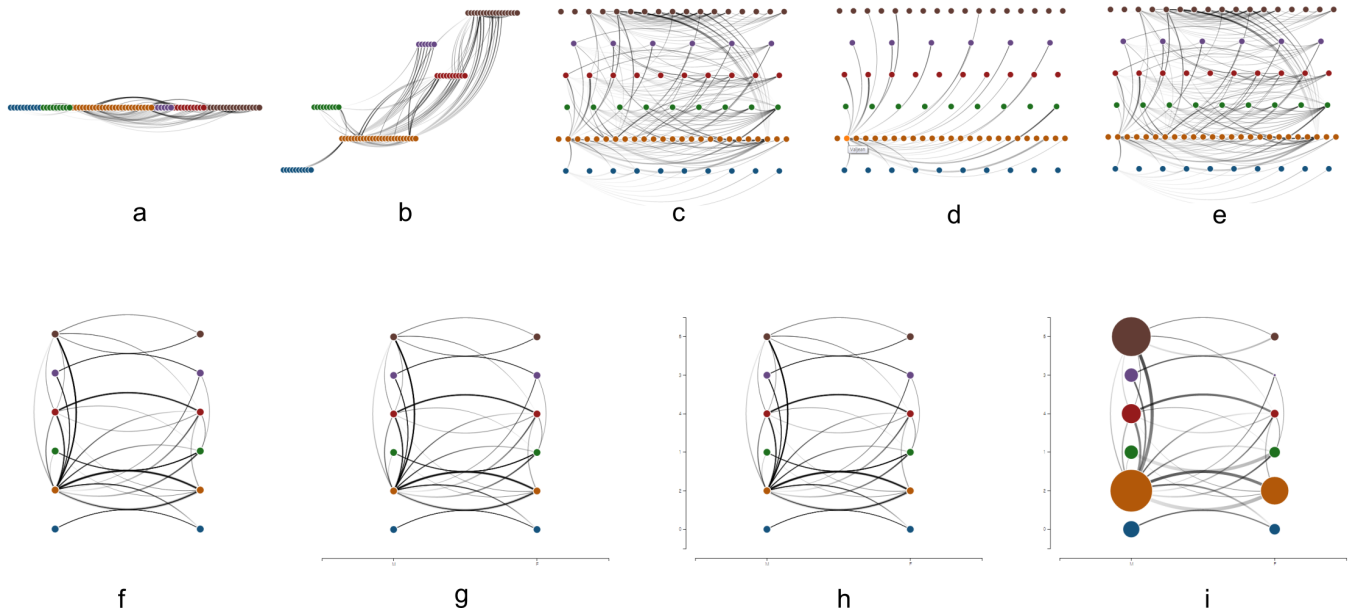While these two examples do not prove that new, effective network

Fig. 2: Results of the GLOs making up the transition from (a) arc diagram [29] to (d) semantic substrates [22], then to (i) PivotGraph [30]. The GLOs applied to achieve each display are: (b) Substrate on *y* by category; (c) Distribute on *x* Within Substrates; (d) Display Selected Links; (e) Display All Links; (f) Substrate on *x* by gender; (g) Show *x* Axis; (h) Show *y* Axis; (i) Aggregate by gender and category.

visualization techniques will be discovered using GLOs (arc diagrams are not a new technique and we have not evaluated the technique in Figure 4 for effectiveness) these cases do demonstrate the potential and the feasibility of such a discovery occurring.

3.2 Easing the Engineering Challenge

As we mentioned in our introduction, graph-level operations provide flexibility for developers of graph visualization applications. In the past, these developers have needed to implement each technique individually. As new techniques are identified or become popular, these developers implement them on the fly. In the worst case, this means defining each technique with no reusable code base, though this case is highly unlikely. Having a language for describing both existing techniques and potentially future techniques, as GLOs are, provides developers with a new target for their development.

Rather than building each technique in isolation, the developer can instead develop the code for each GLO in a set, just as they might for any interface in an object-oriented software system. These GLOs provide seamless reusability and rapidly speed up the implementation of the techniques themselves. By specifying techniques in terms of GLOs, the developer need only write short code blocks to apply the necessary GLOs for each technique. This applies to both existing techniques as well as any new techniques that might be discovered. Furthermore, it is trivial for a developer to enable the analyst exploration capability that we described above when the techniques are already implemented as GLOs. In other words, graph-level operations provide developers with an easier means to implement techniques that inherently provides a powerful and useful additional capability to their customers.

4 GRAPH-LEVEL OPERATIONS

Following the method in Section 2 using the techniques in Table 1, we identified 34 graph-level operations (GLOs) and categorized each into five classes: 1) positioning nodes, 2) modifying element properties, 3) cloning elements, 4) aggregating elements, and 5) modifying display properties.

In this section, we describe the five categories and their GLOs.

4.1 Positioning Nodes

The GLOs in this category each applies a different algorithm for determining the spacial properties (two-dimensional coordinate positions) of the node glyphs in the visualization.

- **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 attribute of the node.

- **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 leftmost 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.

4.2 Modifying Element Properties

The GLOs in this category each algorithmically 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.

4.3 Cloning Nodes

This category of GLOs allows for duplicating node glyphs and interacting with the various sets of duplicates. Each cloning operation (as well as each aggregation 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.

4.4 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 [30].

- **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.

4.5 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.

5 PROPERTIES OF GRAPH-LEVEL OPERATIONS

Here we present five properties of graph-level operations that we identified while developing the model:

- **Duplication of GLOs.** With the exception of the "Clone Active Generation" GLO, applying the same GLO twice has no effect. For example, applying the "Display Links as Curved" GLO twice has the same effect as applying it once: the edges are transitioned to and displayed as curved.

- **Parameterized GLOs.** There are a number of GLOs that take parameters, such as an attribute or an axis. In the case where two instances of the GLO are applied with the same parameter, these are considered duplicate GLOs as described above, and therefore has no effect. However, in the cases where the parameter is different, these are not duplicate GLOs and are treated as distinct. For example, evenly distributing nodes along the x axis is distinct from evenly distributing nodes along the y axis.

- **Complementary GLOs.** We do not consider GLOs to be reversible manipulations. However, this is not to say that an analyst has no means to undo applying a GLO. Instead, GLOs have complementary GLOs. Complementary GLOs overwrite certain previously applied GLOs. For example, any GLO that positions nodes along a particular axis will overwrite any previous GLOs that positioned the nodes along that axis. Showing links as either straight, curved, or hidden overwrite each other. Setting the source or target of a generation overwrites previous source or target GLOs. When saving GLOs as techniques, the overwritten GLOs can safely be ignored. Thus, to undo a given GLO, the system or the analyst must merely reapply the most recently overwritten GLO complementary to the given GLO.

- **The Generation Parameter.** It is important to note that all GLOs (including cloning and aggregation) have an implied generation parameter. In other words, performing the same GLO successively on two different generations *does* have an effect. For example, distributing generation 2 evenly on x and then distributing generation 3 evenly on x is not equivalent to only distributing generation 2 evenly on x. Because aggregating nodes discards the original nodes, it is impossible to call an aggregation on the same generation twice without applying the complementary deaggregation GLO. In contrast, duplications of the "Clone Active Generation" GLO must be treated separately because, if called twice with the same generation parameter, two new generations are created.

- **Application of Techniques.** As GLOs are applied to the active generation using the Generation Parameter, techniques are effectively applied to the active generation as well. For example, if the active generation has a node element for each node, then applying the circle plot technique will generate a circle plot of the nodes. However, if a "Aggregate Nodes by *attribute*" GLO has been applied to the set (resulting in an active generation of aggregate supernodes) then applying the technique would result in a circle plot of the supernodes, rather than of the nodes themselves.

6 GLO-STIX

In this section we describe the design and implementation of GLO-STIX (**G**raph-**L**evel **O**perations for **S**pecifying **T**echniques and **I**nteractive e**X**ploration), a prototype application for exploring graphs using GLOs.

6.1 Design Goals and Rationale

GLO-STIX, a prototype implementation the GLO model, serves as a proof-of-concept of the benefits of GLOs described in Section 3. Our goals in designing the prototype were focused on enabling a graph analyst to experiment with and interactively explore a newly encountered graph dataset. We envisioned an analyst, upon first receiving a dataset, wishing to better understand the graph's features. We see the analyst using GLO-STIX to apply GLO-specified techniques. These techniques may have been specified by the analyst or pre-built into the system. We also foresee the analyst exploring the network further by applying individual GLOs, with the possibility of identifying new network visualization techniques and saving them for future use. In addition, developing the GLO-STIX prototype provided a testbed for evaluating the viability of GLOs and the GLO model.

We adopted the design guidelines of Shneiderman and Aris for semantic substrates [22] and their principle of iteratively applying and evaluating the creation of a visualization. The analyst begins with an idea, applies it to the visualization and evaluates if it creates the desired outcome. In the case of semantic substrates, if the idea was ineffective, the analyst can apply other attributes to the visualization until reaching an ideal understanding of the graph. Applying this principle to GLOs implied that the user interface should enable the analyst to explore a graph by experimenting with applying various GLOs until a deeper understanding of the graph was reached. This approach allows the analyst to observe changes to the display by means of animated transitions as GLOs are applied, enabling a deeper understanding of the benefits and limitations of various techniques.

Based on this principle we generated a number of requirements for a GLO-driven graph visualization prototype:

- Since these are the operations necessary to specify our set of canonical techniques, the prototype should implement the full set of GLOs identified in Section 4.

- As our intent is 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 only GLOs necessary to recreate the display, as opposed to the full path he or she took to reach the display. As we described in Section 5, applying complementary GLOs can override previously-applied GLOs. Therefore the prototype should communicate to the analyst which GLOs in the history are no longer relevant to the current visualization due to complementary GLOs having been applied.

- If an analyst has identified an effective technique, he or she will likely wish to apply it to the same or a different graph in the future. Therefore the prototype should enable an analyst to save a GLO history as a technique to apply to other graphs.

- 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 [7] and Graphical Histories [10].

6.2 Interface Development and Implementation

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 un-applying and re-applying a GLO to a graph, saving the current configuration of GLOs for later use as a technique.

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 1. 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.

We then implemented the functions and UI as a a browser-based application that we have dubbed GLO-STIX. GLO-STIX is written in JavaScript using D3.js [6], jQuery[1], Bootstrap[2], and jQueryUI[3]. Figure 1 is a screenshot of the prototype during an analysis of the Les Misérables character co-occurrence graph included with D3.js based on Donald Knuth's jean.dat file[4]. Nodes are characters, and an edge connects two characters if they co-occur in a chapter. Furthermore, all of the graph images in this paper were generated using the prototype.

7 RELATED WORK

7.1 Graph Visualization Programming Toolkits

There have been many software systems and toolkits developed for graph visualization. Some visualization toolkits, such as Protovis [9] and D3 [6], provide a declarative language for developing new visualizations, including network visualizations, enabling developers to create data visualizations more easily. There are also some software libraries for graph visualization, such as JUNG [19]. However, these tools require users to have programming skills, which makes it difficult for many analysts to use them. The Ploceus [18] and Orion [11] projects enable analysts to transform and analyze relational data as graphs.

GUESS [1] introduced a system for navigating graphs, which allows analysts to rapidly prototype visualization by using its Python-based interpreted language. It introduced functions and operators for manipulating graph data. This type of interpreted language is better than toolkits in terms of accessibility to non-programmers or practitioners, but we believe its command-line input still makes it more difficult for them to use. With GLO-STIX, users can specify a number of graph visualization techniques by choosing a set of operations (GLOs) with our graphical user interface.

7.2 Graphical Tools for Graph Analysis

There exist several graphical tools, such as UCINet [5], Pajek [4], and Gephi [3], for network analysis, and they make it possible for analysts to make visualizations of graphs without programming. NodeXL [23] enables analysts to do so in a commercial spreadsheet. Users can easily import, transform, analyze, and visualize network data with it. However, these systems tend to provide only a small number of visualization techniques, usually focusing on node-link diagrams, while we provide a variety of operations to manipulate visual elements to create a more diverse range of visualization techniques.

8 LIMITATIONS AND FUTURE WORK

We feel that the Graph-Level Operation (GLO) model provides a novel avenue for the visualization research community. Section 3 demonstrates the benefits that GLOs can provide for network analysts, visualization researchers, and graph visualization system developers. However, GLOs are a new paradigm. We present in this section limitations that GLOs have in our current GLO-STIX implementation. We note that many of these limitations, such as not supporting sub-graph selection or edge bundling, are merely of the current implementation and input techniques. Other limitations, such as the dependence on the source technique set, are of the GLO model itself. However, we expect that future work on the part of the authors and other researchers will find solutions for the majority these issues.

While the identification process described in Section 2 lowers the uncertainty of the level atomicity, it does not remove it entirely. For

example, are "Evenly Distributing Nodes by {attribute}" and "Evenly Distribute Nodes" one operation or two? While we settled on considering these as two distinct operations, one might interpret the latter operation the same as the former with a *null* attribute parameter. As long as there is consistency in the process, however, the precise level of atomicity is largely irrelevant.

A more relevant limitation of the process is the dependence on an input set of visualization techniques. While using this method has the advantage of better identifying the atomic units of a techniques, it also has the effect of restricting the GLO set to features of the input set. If a GLO is not present in the input set, it cannot manifest in any output visualizations. For example, the technique described in Figure 4 would not have been possible had the input set not included some form of radial-based layout. This can extend to otherwise simple operations, such as the ability to show curved edges or size nodes by attributes. "Bundle Edges" [13, 8, 14] (equivalent to "Show Edges as Curved") or "Highlight Nodes Matching {predicate}" would be worthwhile and implementable GLOs, yet our input set of techniques did not include a technique that used either.

The set of GLOs we have presented in Section 4 act on sets of elements representing nodes and edges. Since the techniques we describe in Table 1 and Section 3.1 only require that the operations be applied to every node or edge in the set, we are able to use the current generation as the subset of elements that applied GLOs act upon. However, techniques such as hive plots [17], NodeTrix [12], or SoccerStories [20] require that GLOs be applied to arbitrary user- or data-defined subsets of elements. We expect that by considering subsets as generations (as they share many of the properties of generations such as a single node in the data may be represented by multiple generations), implementations should be able to take advantage of the Generation Parameter described in Section 5 to represent the subset.

A limitation merely of our current implementation is that we only draw a single glyph for each edge and require that edges be drawn from a source node to a target node. Thus we currently only support directed graphs (GLO-STIX converts undirected graphs into directed graphs by arbitrarily designating one vertex the source and the other the target). Proper support for undirected graphs, as well as more advanced directed- and undirected-graph data structures such as trees and nested graphs, remain to be implemented. A related point of future work is that we have focused exclusively on static graph visualization techniques. GLO support for dynamic graph data and dynamic graph visualization techniques such as Matrix Cubes [2] or small multiples [26] would be worthwhile to explore. As techniques such as parallel coordinates [15] and parallel scatterplot matrices [28] require drawing links not between source and target nodes, but rather from the same node in different generations, these cannot yet be expressed using GLO-STIX. (Each also requires more than one link for each node.) GLO-STIX also currently has no support for alternative node representations such as data-driven icons.

We defined GLOs in Section 2 as "encapsulated manipulations of a graph visualization." In this paper we have kept to GLOs for manipulating graph *visualization elements*. An interesting extension of this paradigm would be to discuss operations for manipulating graph *data*. The most important of these "data GLOs" are filtering GLOs such as "Filter *attribute* by *predicate*" that restrict the dataset. Other potential data GLOs include "compute distance from *target*", required for creating even a simple Sugiyama hierarchical layout [25] or "compute most-interesting neighbors" according to an algorithm such as van Ham and Perer's graph degree-of-interest algorithm [27].

Finally, we must mention the potential usability limitations introduced by the generation construct. For instance, understanding the dependency relationships within a technique and keeping track of generation numbers could all potentially confuse a user of a GLO implementation. Furthermore, there is the simple fact that cloned generations look exactly like the source generation. We also do not know without a user study how much confusion long sequences of GLOs can introduce. While we enable an analyst to remove "out-dated" GLOs based on complementary GLOs, the lengths of even active GLOs can still become quite long. However, many of these issues can be miti-

---

gated. For example, order-dependence can be mitigated by assuming that each GLO in an analyst-designated technique has a dependence relationship with the GLO preceding it. UI approaches such as highlighting the active generation (demonstrated in Figure 4) can mitigate the confusion as well. We feel that the greatly expanded expressability of the GLO model provided by the generation construct are more than worth these small limitations.

9 CONCLUSION

We presented GLOs (graph-level operations), a new model for specifying graph visualization techniques. We contributed a method to identify GLOs and demonstrated GLOs' flexibility in re-creating canonical graph visualization techniques. We discussed the important properties of GLOs and the benefits to graph visualization that GLOs provide. We presented GLO-STIX — an implementation of our GLO set and an interface for exploring a graph using GLOs. Finally, we discussed the limitations of the GLO and GLO-STIX implementations at time of publication and potential future research directions concerning the Graph-Level Operation model in general.

REFERENCES

[1] E. Adar. GUESS: a language and interface for graph exploration. In *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems, (CHI 2006)*, pages 791 –800. ACM, 2006.

[2] B. Bach, E. Pietriga, and J.-D. Fekete. Visualizing dynamic networks with matrix cubes. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems, (CHI 2014)*, pages 877–886. ACM, 2014.

[3] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *ICWSM '09*, pages 361 –362. AAAI, 2009.

[4] V. Batagelj and A. Mrvar. Pajek– analysis and visualization of large networks. In *Graph Drawing Software*, Mathematics and Visualization, pages 77 –103. Springer Berlin Heidelberg, 2004.

[5] S. P. Borgatti, M. G. Everett, and L. C. Freeman. UCINet for windows: Software for social network analysis, 2002.

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301 –2309, Dec. 2011.

[7] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: visualization meets data management. In *Proc. of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 745–747. ACM, 2006.

[8] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, Nov. 2008.

[9] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010.

[10] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, Nov. 2008.

[11] J. Heer and A. Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. In *Proc. of IEEE VAST 2011*, pages 51 –60, 2011.

[12] N. Henry, J.-D. Fekete, and M. McGuffin. NodeTrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302 –1309, 2007.

[13] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sept. 2006.

[14] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, June 2009.

[15] A. Inselberg. Multidimensional detective. In *Proc. of IEEE Infovis 1997*, pages 100 –107, Oct. 1997.

[16] E. Kandogan. Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In *Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 107–116. ACM, 2001.

[17] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra. Hive plotsrational approach to visualizing networks. *Brief Bioinform*, 13(5):627–644, Sept. 2012.

[18] Z. Liu, S. Navathe, and J. Stasko. Network-based visual analysis of tabular data. In *Proc. of IEEE VAST 2011*, pages 41 –50, Oct. 2011.

[19] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Analysis and visualization of network data using JUNG. *Journal of Statistical Software*, 10(2):1 –35, 2005.

[20] C. Perin, R. Vuillemot, and J.-D. Fekete. SoccerStories: a kick-off for visual soccer analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2506–2515, Dec. 2013.

[21] G. Rugg and P. McGeorge. The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 22(3):94 –107, July 2005.

[22] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733 –740, 2006.

[23] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave. Analyzing (social media) networks with NodeXL. In *Proc. of the Fourth International Conference on Communities and Technologies, (C&T '09)*, pages 255 –264. ACM, 2009.

[24] C. Stolper, F. Foerster, M. Kahng, Z. Lin, A. Goel, J. Stasko, and D. Chau. GLOs: graph-level operations for exploratory network visualization. In *ACM SIGCHI 2014 Work-In-Progress Extended Abstracts*, 2014.

[25] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, Feb. 1981.

[26] E. R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Conn., 1995.

[27] F. van Ham and A. Perer. Search, show context, expand on demand: Supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):953 –960, Nov. 2009.

[28] C. Viau, M. McGuffin, Y. Chiricota, and I. Jurisica. The FlowVizMenu and parallel scatterplot matrix: Hybrid multidimensional visualizations for network exploration. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1100 –1108, 2010.

[29] M. Wattenberg. Arc diagrams: Visualizing structure in strings. In *Proc. of IEEE Infovis 2002*, pages 110 –116, 2002.

[30] M. Wattenberg. Visual exploration of multivariate graphs. In *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems, (CHI 2006)*, pages 811 –819. ACM, 2006.