Improving Property Graph Layouts by Leveraging Attribute Similarity for Structurally Equivalent Nodes

Patrick Mackey, Jacob Miller, and Liz Faultersack

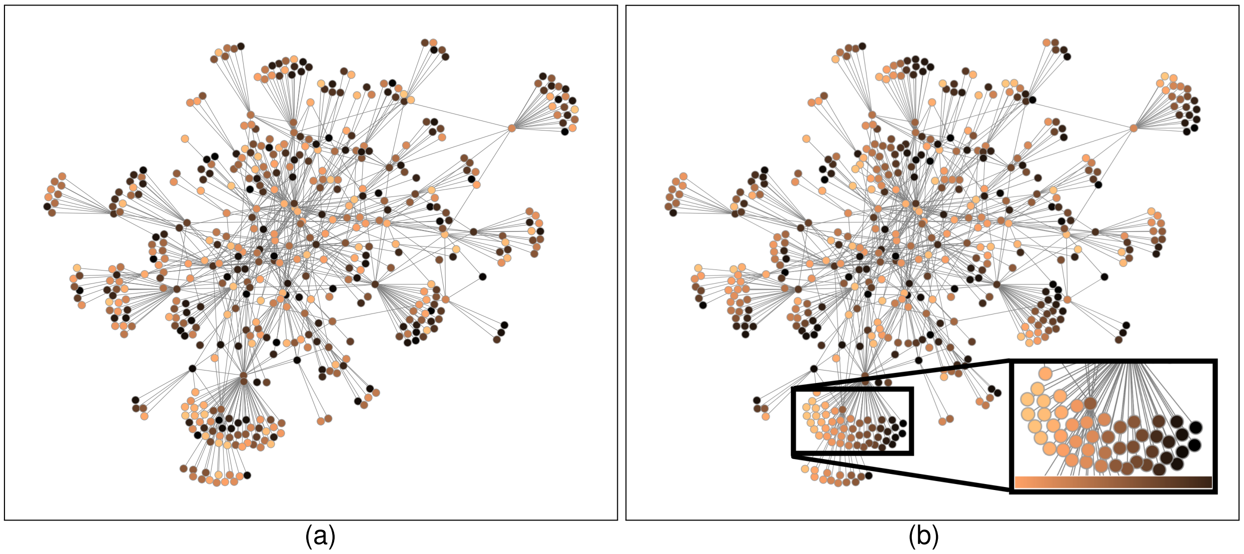

Fig. 1: An example property graph before (a) and after (b) having its structurally-equivalent nodes re-arranged based on their attribute similarity.

Abstract—Many real-world networks contain structurally-equivalent nodes. These are defined as vertices that share the same set of neighboring nodes, making them interchangeable with a traditional graph layout approach. However, many real-world graphs also have properties associated with nodes, adding additional meaning to them. We present an approach for swapping locations of structurally-equivalent nodes in graph layout so that those with more similar properties have closer proximity to each other. This improves the usefulness of the visualization from an attribute perspective without negatively impacting the visualization from a structural perspective. We include an algorithm for finding these sets of nodes in linear time, as well as methodologies for ordering nodes based on their attribute similarity, which works for scalar, ordinal, multidimensional, and categorical data.

Index Terms— graph drawing, network visualization, property graphs, attributed networks

1 Introduction

Graphs or networks are a common discrete data structure for representing relationships or interactions of real-world entities. Common examples include social networks, critical infrastructure and cybersecurity data. By visualizing this data as a graph, analysts can gain insights into these entities (also called nodes or vertices) and their importance and placement in the larger network, as well as understanding general characteristics of the network at large.

In many real-world graphs and networks, it is common to encounter many *structurally-equivalent nodes* [15]. These are defined as nodes that share identical sets of neighbors. In a traditional topologically driven graph layout, these nodes can be interchanged in the visualization without effecting the appearance of the graph. However, many real-world networks are also *property graphs* (or *attributed networks*) [14], meaning they have one or more attributes associated with the nodes or edges. These attributes can indicate meaningful differences in these nodes. However, most existing graph layout algorithms ignore this additional information. In this paper we propose a technique that allows us to take an existing graph layout, and swap the locations of the structurally-equivalent nodes so that the nodes that are more similar in their properties are placed closer to each other. This technique enables us to use any existing graph layout technique without altering its fundamental characteristics, while leveraging the properties of the nodes to improve the overall usefulness of the layout for analytical purposes.

• *Patrick Mackey is with Pacific Northwest National Laboratory. Email: patrick.mackey@pnnl.gov*
• *Jacob Miller is an intern at Pacific Northwest National Laboratory and a PhD candidate at University of Arizona. Email: jacob.miller@pnnl.gov*
• *Liz Faultersack is with Pacific Northwest National Laboratory. Email: liz.f@pnnl.gov*

2 Related Work

A large body of work exists on techniques for creating graph layouts [6]. The majority of this work focuses only on the connectivity or structure of the graph, and not the properties or attributes on the nodes or edges. In our approach, any of these algorithms can be used as the initial locations for the graph layout.

There exists a smaller body of work on property or attribute-driven graph layouts [7]. These algorithms use the similarity of the node attributes as part of the optimization equation for placing nodes. For example, Zhuang Xu, et al., use a structurally-based force-directed algorithm to initially place nodes in a graph, and then apply an additional force against the nodes based on their attribute similarity to further modify the layout [17]. Alternatively, *MVN-Reduce* creates a layout using a dimension-reduction approach, where both the topological dis-

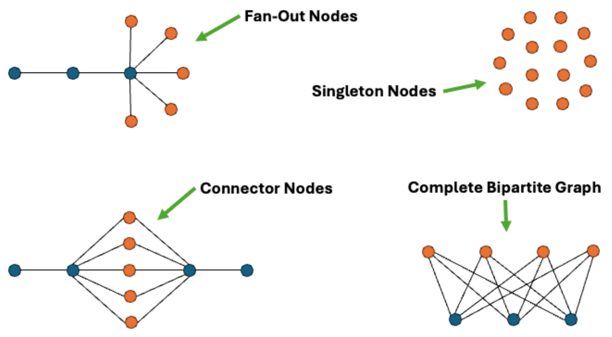

Fig. 2: Examples of common structural equivalence groupings (SEGs).

tance between nodes and their attribute similarity are included in the dimensionality reduction optimization [11]. These methods differ from our own approach, which always retains the original structurally-based node positions, but swaps locations of structurally-equivalent nodes. This enables us to retain the desired qualities of a traditional graph layout while also improving the overall interpretability from a property perspective.

3 Structural Equivalence

While concepts around structural equivalence originate with research by social scientists Lorain and White [10], the definition of structural-equivalent nodes we are using is based on the work of Lei Shi, et al. [15], which they refer to as a *structural equivalence grouping* (SEG). The authors given definitions of SEG for directed and weighted graphs as well as simple graphs. For our purposes, only the simple graph definition is needed, as we follow the common pattern of ignoring weight and directionality in computing a graph layout. In the case for simple graphs, an SEG is defined as the set of vertices that share an identical set of neighbors. An efficient algorithm exists to find these groups, which we describe in more detail in Section 4.

Figure 2 demonstrates some examples of structurally equivalent groupings. A common structurally equivalent pattern are "fans", where a single vertex connects to a large number of vertices with no other edges [1]. Another common pattern are "connectors", which is a set vertices that connect a pair of vertices together, without any edges to other vertices [1]. Any vertices without edges would also be structurally equivalent. Either side of a complete bipartite graph would also be structurally equivalent. As Shi, et al., point out, in many real-world networks these various structurally-equivalent nodes can represent the majority of vertices in a graph, particularly in cyber-security data [15], which is the primary use case motivating our research as well.

It is a well-known phenomenon that people equate proximate entities as being more similar than those more distant (the Gestalt law of proximity) [8]. This can potentially cause a user to assume some structurally identical nodes are more similar to each other than other structurally equivalent nodes depending on their proximity to each other in the visualization. An example can be seen in Figure 3(a), where nodes in one portion of the fan-out may appear more related than those on the opposite side, despite the fact that all nodes (except the high-degree center node) are structurally equivalent. The layout in Figure 3(b) shows the same graph after applying our node-swapping technique, enabling nearby nodes to be similar in their properties as well as structure. This approach also enables us to use any existing graph layout algorithm we choose, which can help avoid many of the well known issues in graph visualization (e.g., edge crossings, angular resolution, edge-length deviation, aspect ratio, etc. [12]).

4 Methodology

Any existing graph layout technique can be used for the initial node placement. In most of our examples we will be demonstrating our technique using force-directed based layouts (specifically, the Fruchterman-Reingold) [4], but other layout algorithms such as spectral methods [9], orthogonal layouts [2] and circular layouts [5] may also be used, among others.

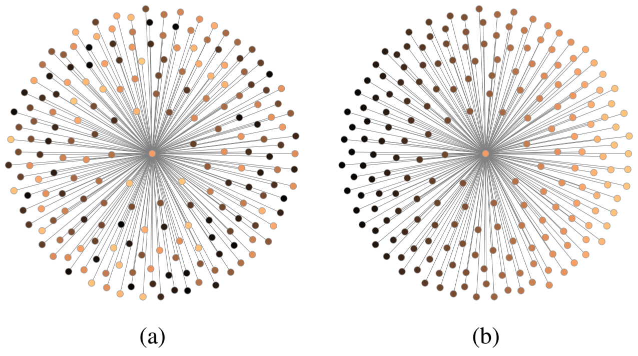

Fig. 3: Star subgraph often found in real-world data such as cyber-security data [15], nodes are colored according to a scalar property value. Without taking properties into account, a layout might look like (a), but our algorithm will modify the existing layout to (b).

4.1 Finding Structurally Equivalent Vertices

To determine which nodes are structurally equivalent, we have implemented a simple algorithm that works in linear time with regards to edges (see Algorithm 1). Structurally-equivalent groups are identified using hashcodes, which are generated by performing a bitwise exclusive-or operation on all neighboring nodes' hashcodes (which individually are defined by any given hashing scheme). This gives a unique hashcode for the given set of nodes which will be identical for all structurally-equivalent nodes.

There is a very small possibility that non-equivalent nodes could hash to the same hashcode, but this is extraordinarily unlikely given the typical size of hashes ($2^{64} - 2^{256}$) and the typical number of nodes in a graph visualization ($< 10,000$). For a 64-bit hash and a 10,000 node graph, the odds of a hash collision are roughly 2.7e−12. If such situations are of concern, the validity of the SEG hash can be confirmed by comparing the contents of each neighborhood for each matching vertex.

Algorithm 1 Find all simple structurally-equivalent groups (SEG) of vertices. Time complexity: $O(|E|)$

```
function FINDALLSEG($V, E$)                ▷ Where $V$=vertices, $E$=edges
    Let $S$ be a hashtable of integers to sets of vertices
    for $u \in V$ do
        $h = 0$                             ▷ For creating SEG hashcode
        for $v \in$ nbrs($u$) do            ▷ Create hash from neighbors
            $h = h$ bitwise-XOR hash($v$)
        end for
        $S[h] = S[h] \cup \{u\}$            ▷ Add $u$ to the SEG for this hash
    end for
    return $S$
```

4.2 Ordering Vertices by Attribute Similarity

To select the new positions of nodes based on their properties, many different techniques could be applied. We assign a linear relationship between all structurally equivalent vertices based on their associated attributes. While more sophisticated approaches could be applied, for our initial work, we are using a simple Principle Component Analysis (PCA) approach for ordering nodes based on their similarity [16]. Using PCA, we find a line through the point set which captures the largest amount of variance. All points are projected onto this line and ordered accordingly to $p_1, p_2, \ldots p_n$. Then, we want to label each node as $v_1, v_2, \ldots, v_n$ so that the position of v_i is p_i. Our method of doing so is dependent on the type of data present.

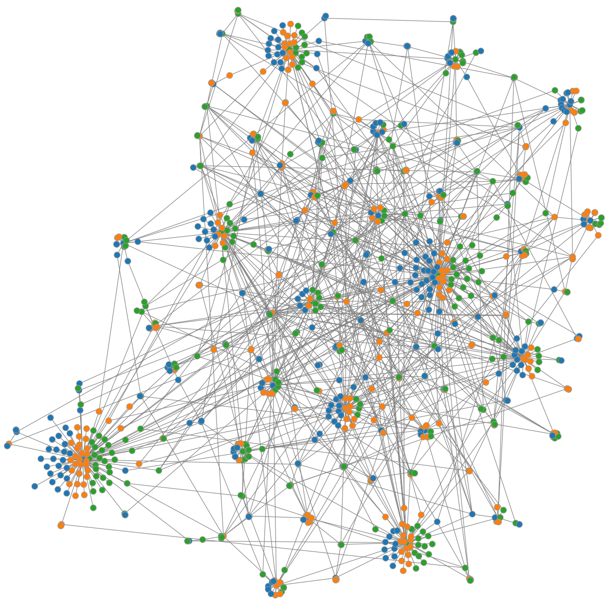

Fig. 4: An example of our property graph layout applied to a graph with categorical data.

Scalar and ordinal data In the simplest case, the properties of interest are simple real-valued scalars or other data with natural ordering. We can sort the values and label the vertices such that $v_1 \leq v_2 \leq \ldots \leq v_n$.

Multidimensional data It may be that there is a vector of real-values associated with each node in the graph, which is desired to be captured by the layout. We must find an ordering of these vectors that captures their similarity. Much like we did with the point set, we use PCA to project these vectors onto a line so that the variance between them is maximized. Given this one dimensional line, we can order the vertices along it.

Categorical data When there is categorical data associated with the vertices of a graph, and each vertex belongs to one and only one category, we arbitrarily order the categories to convert to ordinal data.

Mixed categorical data In some cases, there is mixed categorical data. Vertices can belong to multiple categories, or there are weights associated with the categories (e.g., a vertex has 2 bananas, 2 apples, and 1 orange). We first convert these into percentages (e.g., 40% banana, 40% apple, 20% orange) and apply an arbitrary ordering of the categories.

Algorithm 2 Swap positions of vertices.

```
function PositionSwap($SEG, P$)    ▷ Where $SEG$ is a structural equivalent group, $P$ are positions of vertices
    Let $L$ be a line of best fit through the positions
    Sort $P$ along $L$, label $p_1, \ldots, p_n$
    Sort $SEG$ by attribute value, label $v_1, \ldots, v_n$
    for $v_i \in v_1, \ldots, v_n$ do
        Assign $v_i$ to position $p_i$
    end for
```

5 Examples

We discuss the examples that appear throughout the paper. Fig. 3 shows a simple example of our algorithm applied to a small star property graph with scalar values associated with the nodes. The layout algorithm in

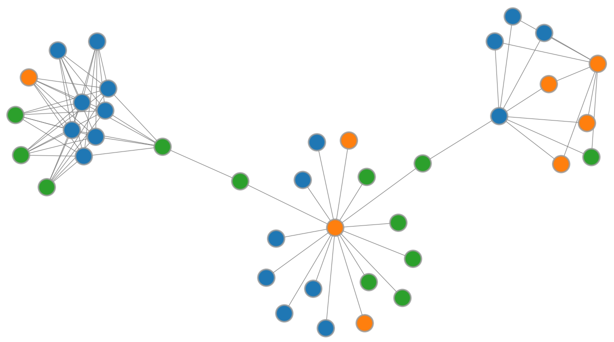

Fig. 5: A small synthetic example of categorical ordering of a graph with several common SEGs. On the left is a biclique, the center shows a star, and the right a connector node pattern. Here, the patterns are drawn via the Fructerman-Reingold algorithm, but a clearer example of their structure is shown in Fig. 2.

Fig. 3(a) has a difficult task of finding two dimensional positions of these nodes which are all structurally equivalent. However, we have more information available to us via node properties. We trust that the given layout has done as good of a job as it can in finding placements of nodes which are structurally equivalent (in this case, all but one node in the graph). We swap node placements so that nearby nodes in the 2D drawing space are similar in attributes. If we apply a color scheme according to this attribute, we see there is in fact a nice gradient from left to right across the graph.

In larger, more complex graphs there may be many nodes which are not structurally equivalent. Still, even in real-world networks SEGs occur frequently enough to warrant consideration. Fig. 1 shows a graph generated to have similar properties as the Internet graph [3]. We can visually see several fan-out, star structures in Fig. 1 (a). These can be quickly detected programmatically using Algorithm 1, after which we can repeatedly apply the algorithm described in Section 4.2 on these sub-graphs. We see quite clearly that the placement of vertices with respect to their attributes within SEGs is vastly improved, highlighted by the zoomed box in Fig. 1 (b).

As noted in Section 4.2, our method is not limited to singular scalar values but can be applied to ordinal and categorical attributes as well. Fig. 5 shows such an example on a small synthetic graph with categorical attributes. This graph also contains additional SEGs besides the typical star motif: the biclique and connector nodes. In each of these SEG motifs, nodes of the same color are grouped together.

A larger categorical example is found in Fig. 4. This graph (another graph generated by [3]) has several high degree "hub" nodes in which many of its neighbors have a degree of one. The ordering of these groupings is very visually salient, from blue to orange to green as one scans the visualization.

We extend the complexity to mixed categorical data, so that nodes may belong to one or multiple categories. Visually represented as pie charts on nodes, Fig. 6 (a) shows an unstructured layout of a synthetic graph from [3]. The large star patterns in the bottom right appear unordered and unrelated. However, our algorithm generalizes to such mixed data as can be seen in Fig. 6 (b). In the same fan-outs there is a smooth transition from nodes of one type to another, and nearby nodes tend to belong to the same category even if not exactly the same set of categories.

Finally, the node properties may contain many values representing categories. For instance, the properties may count how many fruits belong to a node (e.g., 3 bananas, 2 apples, 1 orange). We can still visually represent this membership as a pie chart (for relatively small sets of categories). Fig. 7 shows an example of a large star graph with these attributes. While the trend is less visually salient than the scalar or categorical data, there is still a clear trend (orange to green, to blue) that shows nearby nodes are similar.

These examples show that our technique can be applied to a wide

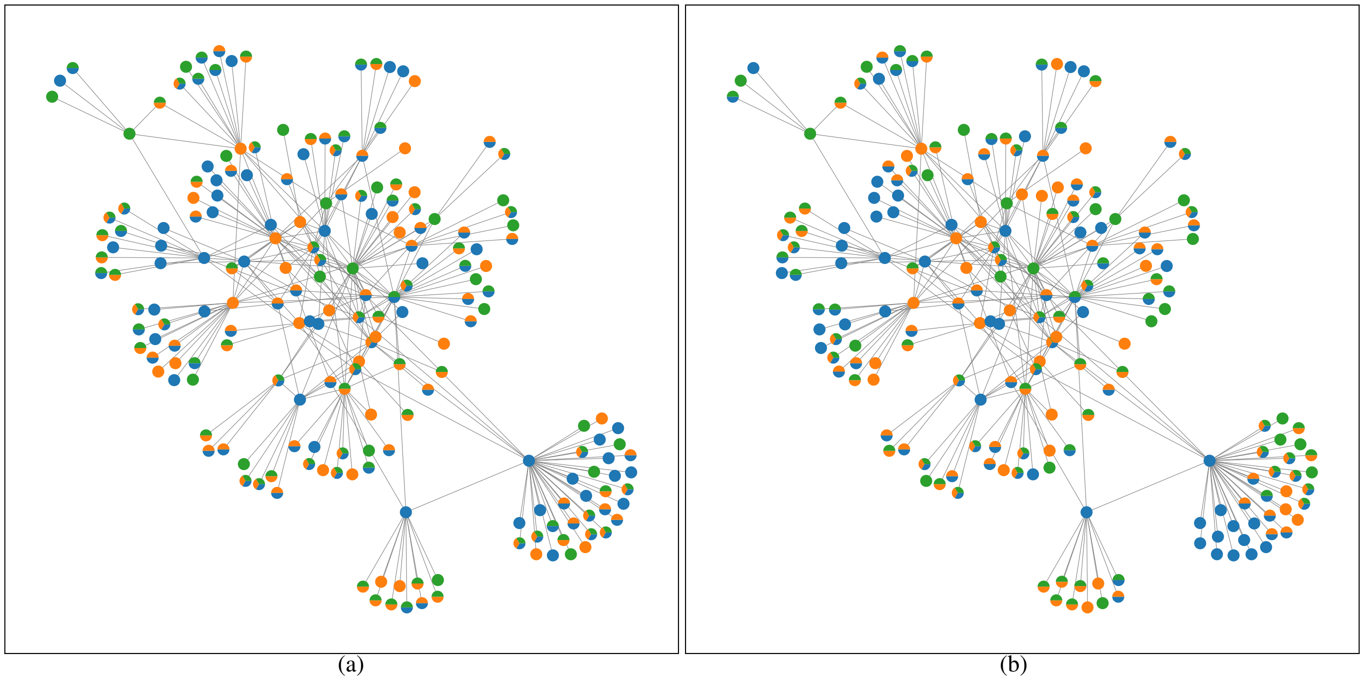

Fig. 6: Our method also works with mixed categorical data, represented here by pie charts as nodes. Nearby structurally equivalent nodes are possibly unrelated in (a), but after our algorithms rearrangement, there is more structure (b)

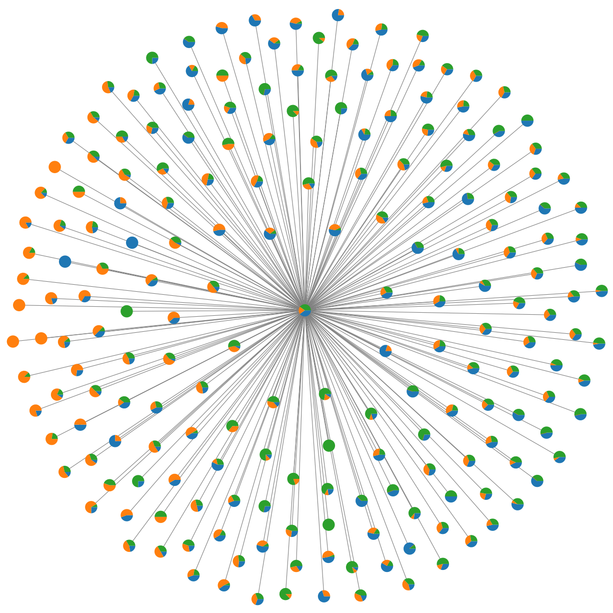

Fig. 7: A slightly more complicated synthetic example where each node has multiple numerical attributes associated with it. Notice how, in general, the colors go from orange, to green, to blue from left to right.

variety of property data types with relatively little complexity; improving the graph layout for very little cost in comparison to computing the layout in the first place.

6 Conclusion and Future Work

We believe our technique provides a helpful methodology for improving the meaningfulness of graph layouts for property graphs with many structurally equivalent nodes. We believe there are many real-world examples of these, especially including cyber-security data, for which this effort was funded. However, there are situations where our technique may not be well suited, such as networks where structural-equivalence is minimal. A potential future work would be to explore a "fuzzy equivalence" in such graphs, allowing us to swap locations for nodes that are nearly equivalent. This would be particularly helpful for dense graphs, in which vertex placement can seem somewhat arbitrary, and there may be few nodes that are completely structurally equivalent. Existing algorithms for fuzzy structural-equivalence grouping would be worth exploring in our future work [13, 15].

Additional future work includes improving our approach for selecting node placement. Our technique applies a linear approach, which is efficient in practice, but may be sub-optimal in certain cases. Additionally, our technique could benefit from additional research on how best to combine properties from nodes (or their adjacent edges) for determining node similarity, as many property graphs have multiple properties per node or edge. This could likely benefit from additional research on how users are using this information and what would be most important for their particular use case needs.

Acknowledgments

We would like to thank Claire Cassalnova, James Patterson, Victoria Wallace and Jordan Eberst for their support of this effort as well as their guidance on applying this research for the cyber-security response use case.

This manuscript has been authored by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830 with the U.S. Department of Energy on behalf of the Cybersecurity and Infrastructure Security Agency (CISA). The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] Cody Dunne and Ben Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 3247–3256, 2013. 2

[2] Markus Eiglsperger, Sándor P Fekete, and Gunnar W Klau. Orthogonal graph drawing. In *Drawing Graphs: Methods and Models*, pages 121–171. Springer, 2001. 2

[3] Ahmed Elmokashfi, Amund Kvalbein, and Constantine Dovrolis. On the scalability of bgp: The role of topology growth. *IEEE Journal on Selected Areas in Communications*, 28(8):1250–1261, 2010. 3

[4] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991. 2

[5] Emden R Gansner and Yehuda Koren. Improved circular layouts. In *International Symposium on Graph Drawing*, pages 386–398. Springer, 2006. 2

[6] Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization*, 12(3-4):324–357, 2013. 1

[7] Steffen Hadlak, Heidrun Schumann, and Hans-Jörg Schulz. A survey of multi-faceted graph visualization. In *EuroVis (STARs)*, pages 1–20, 2015. 1

[8] Stephen G Kobourov, Tamara Mchedlidze, and Laura Vonessen. Gestalt principles in graph drawing. In *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers 23*, pages 558–560. Springer, 2015. 2

[9] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003. 2

[10] François Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971. 2

[11] Rafael Messias Martins, Johannes F Kruiger, Rosane Minghim, Alexandru C Telea, and Andreas Kerren. MVN-Reduce: Dimensionality reduction for the visual analysis of multivariate networks. In *EuroVis (Short Papers)*, pages 13–17, 2017. 2

[12] Gavin J Mooney, Helen C Purchase, Michael Wybrow, and Stephen G Kobourov. The multi-dimensional landscape of graph drawing metrics. In *17th IEEE Pacific Visualization Symposium (PACIFICVIS)*. IEEE, 2024. 2

[13] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432, 2008. 4

[14] Marko A Rodriguez and Peter Neubauer. Constructions from dots and lines. *arXiv preprint arXiv:1006.2361*, 2010. 1

[15] Lei Shi, Qi Liac, Xiaohua Sun, Yarui Chen, and Chuang Lin. Scalable network traffic visualization using compressed graphs. In *2013 IEEE International Conference on Big Data*, pages 606–612, 2013. 1, 2, 4

[16] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987. 2

[17] Zhuang Xu, Tingyun Mao, Guangluan Xu, Yang Wang, and Daoyu Lin. Multivariate network layout using force-directed method with attribute constraints. *Applied Sciences*, 12(9):4561, 2022. 1