# Comparing Force-Directed Graph Layout Algorithms

Nicolas Winsten

**Abstract**—A central problem to visualizing graph data in the traditional node-link format is determining its layout – where vertices will fall on the page. Force-directed graph layout algorithms are an intuitive solution to determining an effective and aesthetically pleasing layout for visualizing graph data. Their approach involves imposing an energy system on the graph and determining vertex placement by minimizing the total energy of the system (adjacent nodes apply attractive forces, non-adjacent vertices repel). Numerous performant force-directed graph layout algorithms exist, and their effectiveness varies with types of graph data. A tool is provided that allows one to directly compare different methods and view how they refine a graph layout throughout their runtimes. This will allow quick and easy comparisons not only against final layout but across the runtime of the algorithm as well facilitating the understanding and improvement of force-directed layout methods.

## 1 Introduction

The node-link diagram is the primary approach to visualizing graph data. The biggest aspect of producing such diagrams is determining where nodes will be drawn relative to one another. Nodes connected by an edge should be drawn closer together, and unrelated nodes should be drawn far apart. Creating a layout that is easy on the eyes and also highlights the structure of the graph data like clusters is paramount in making a decent node-link diagram.

Force-directed layout algorithms have been the most popular class of solutions to this problem. Some methods, however, focus on a specific quality metric and neglect others. Some methods also create good results for some data and bad results for other data. Some methods are designed for continuous settings in which a user can view the refinement process at work. This allows the user a fine-grained understanding of how such layout methods work, however many methods do not lend themselves to that user experience, and are not used as much in interactive tools. This project provides a tool that brings those layout methods together to take advantage of a continuous user experience in order to facilitate the understanding and comparison of these algorithms. The drawbacks and advantages of each method's implementation can be pinpointed and can inform future implementations in achieving more performant specialized or general solutions.

## 2 Background

Graph data takes the form of a set of vertices $V$ and a set of edges $E$ or "links" between vertices. One could visualize this data as an adjacency matrix, but node-link diagrams are much easier to view and are more intuitive. Node-link diagrams have the potential to convey the structure of the data more clearly.

A central problem to visualizing graph data in the traditional node-link format is determining its layout – where vertices will fall on the page. Several factors determine what makes a layout "aesthetically pleasing" and easy on the eyes. Ideally, all vertices should be well-spaced, but adjacent vertices should be in the same area. Dissimilar nodes, or node pairs with higher graph theoretic distance, should not be placed close together if we keep in mind the gestalt design principle of proximity. Most edges should be short. One's eyes shouldn't have to travel very far to find a node's neighbors. Edge-crossings should be minimized to make following edges with one's eyes easier. Groups of nodes that are highly connected should show up as clusters containing many edges and with few edges reaching out to other clusters. Force-directed graph layout algorithms are a popular class of methods used to solve this problem. These methods impose an energy system on the graph by assigning a force value to each pair of nodes and iteratively modifying node positions to minimize the energy of the system. A simple force-directed approach imagines that each edge acts like a spring that compels adjacent vertices to be a certain distance from each other using Hooke's Law to determine force magnitude and that non-adjacent vertices repel each other using Coulomb's Law to determine force magnitude.

The two traditional approaches to force-directed graph drawing are using spring-embedders as described previously or optimizing an energy function.

SPRING ALGORITHM
Input: graph $G$, affecting coefficient $C$, number of iterations $M$
place vertices of $G$ in random locations
**while** $M > 0$ **do**
  **for** vertex $v$ in graph $G$ **do**
    calculate net force $f$ on $v$
    move $v$ by $C * f$
  **end for**
  $M \leftarrow M - 1$
**end while**

This short code encapsulates the simplicity and intuitiveness of force-directed graph algorithms.

Typically, Hooke's Law – $F(d) = k(d - j)$ – which maps distance to force linearly is too strong for nodes that become far apart. A more appropriate edge-force function then becomes $F(d) = k \log(d/j)$ where $k$ is the strength of the spring and $j$ is the appropriate distance between adjacent nodes (when $j = d$, the edge induces no force between the adjacent nodes in their current positions). Another implementation detail is the notion of *temperature* introduced by [7] that exponentially decays the affecting coefficient of $C$ in the above algorithm in each iteration. This modification is meant to allow less vertex adjustments as the layout approaches the optimum. Other avenues of improvement involve data structures and heuristics that may improve the runtime complexity and layout quality. These are a few small examples of the design space in implementing a force-directed graph layout algorithm. One may also choose to depart with the "edges as springs" approach. For example, one may determine forces between two nodes by their shortest path distance in the graph instead [19].

The advantages to the class of force-directed algorithms:

- simple
- intuitive
- easily produces great results for small graphs
- large design space for force models

The main disadvantages are

- high running time

• *Nicolas Winsten is a graduate student at the University of Arizona. E-mail:[nicolaswinsten]@arizona.edu.*

*Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx*

- poor local minima

Typically, force-directed layout algorithms are believed to run in $O(n^3)$ time with respect to the number of vertices. Calculating the pairwise forces is a quadratic operation, and the number of iterations needed is believed to be linear with respect to graph size. Heuristics, optimizations and phased layout techniques have been able to significantly reduce the runtime complexity in practice.

Simple force-directed algorithms have been shown to create very good results for small graphs (hundreds of nodes). As the graph gets larger however, the problem of poor local minima becomes hindering. One avenue to approach this problem is a multi-scale approach such as in [15] in which coarse representations of the graph are used to perform "coarse-scale relocations" of vertices before the finer adjustments are made.

### 2.1 Related Work

In this work, we focus on force-directed graph algorithms that produce 2D straight-line node-link diagrams. Force-directed methods have been applied to other types of graph visualizations that depart from this format such as [5], [3], [16]. Many force-directed graph visualizations today employ a layer of interactivity to make large datasets more digestible, dynamic, and fun.

Previous surveys [10] [22] have been done a decade ago that outline the history of force-directed graph drawing algorithms and evaluate different methods. Other treatments are focused specifically on evaluating the effectiveness of different methods. This can take the form of empirical metric-based evaluations [20] or task-based evaluations [29]. Work has also been done in developing quality metrics for assessing layouts such as [33] [27].

## 3 Methods

Popular force-directed layout algorithms are compared here based on how they produce layouts rather than just their resulting final layouts. A tool is provided to compare layout methods side-by-side to provide a better understanding of how each method differs in its operation and its effectiveness on different types of graph structures.

This tool may inform where current methods are failing compared to others, and how two different methods may be reconciled to produce better results. This tool facilitates the understanding and improvement of force-directed layout methods.

The basic usage flow for the interactive tool is as follows:

- Select a graph type to perform a layout upon. Modify any relevant settings such as graph size or density.
- Select the desired layout algorithms to display. Modify the algorithm parameters. Select the same algorithm multiple times to get a sense of their parameters.
- Hit play to start the algorithms.
- Brush certain nodes to color them across all the layouts to orient yourself
- Modify graph/algorithm parameters, or begin anew

The algorithms are run in-step with each other, but since an "iteration" is somewhat asymmetric across these methods, I opted to choose a quadratic operation to constitute a single iteration. This means that Fruchterman-Reingold style methods calculate the total forces on each node in a single iteration, but for Kamada-Kawai, a single iteration involves picking out the maximal energy node and computing its stable position (see section 5 for details on these algorithms).

### 3.1 Data

This tool provides a suite of types of graphs meant to cover a large breadth of distributions and structure. To name a few graph dataset characteristics:

- clustered vs uniform data
- dense vs sparse
- clear vs cluttered community structures

Real-world datasets are gathered from SNAP [1], such as data regarding social networks like Last.fm users [31] and cross-posting behavior of subreddits on Reddit[2] [23]. In order to cover other types of structures, the interactive tool contains the following preset graph types: random graphs, trees, meshes, and Girvan-Newman graphs.

### 3.2 Evaluation

Metrics used in the past for evaluating layouts include the following:

- average edge length
- average adjacent node edge distance
- number of edge crossings
- angular resolution
- clustering quality
- symmetry

In truth however, evaluating a method's produced layout is very subjective and highly dependent on the type of graph data and what the layout is being used for. For example, a 2D layout of a mesh will place high emphasis on revealing its 3D structure, regular placement of nodes, and uniform edge lengths. On the other hand, a layout of a social network will emphasize community structure over uniform edge lengths. Creating a suite of metrics to grade layouts is a non-trivial task, and so this is considered out of scope for this project.

Instead, the aim of this tool is to facilitate the understanding of how different methods operate on graph data. Most layout evaluation is done by simply viewing the layout and assessing it subjectively. However, the tool provides one method to naively assess node placement. One can brush certain node groups and activate a heated color settings such that all nodes are colored according to their graph theoretic distance to the brushed nodes.

### 3.3 Technology

The tool is implemented in Javascript to be run in the browser, and layouts are drawn using d3.js [2].

## 4 Impacts

This work provides a tool for viewing a side-by-side comparison of each method's refinement process to effectively pinpoint where each method excels and struggles. It may also open up research directions in how different methods may be combined to produce the best results.

## 5 Algorithms

The algorithms implemented in this tool are outlined in this section. The suite of algorithms were chosen to provide a decent covering of the general design space of force directed layouts.

### 5.1 Eades

Eades' spring embedding approach [4] is typically considered the first force-directed graph layout algorithm, and it is the simplest. Eades' approach models a force system on the graph in which nodes apply forces to all other nodes. Non-adjacent nodes repel each other according to Coulomb's law, and adjacent nodes attract each other according to Hooke's law. To layout a graph, the forces being applied to each node are calculated, and each node's position is then moved based on that force. This process repeats until a satisfactory layout has been achieved (or some timeout is reached). All force-directed graph layout algorithms build off this idea of iteratively computing and reducing node-pair forces.

[1] https://snap.stanford.edu/data/index.html
[2] https://www.reddit.com/

### 5.2 Fruchterman-Reingold

The Fruchterman-Reingold approach [7] is very similar to Eades, except for a few main differences. The repulsive force between two nodes is calculated as an electric charge force as before, but this force is applied for every node-pair (not just non-adjacent node pairs). The attractive force for each edge is also not modeled as a spring force, but instead calculated as $f_a = d^2/c$ where $d$ is the distance between the adjacent nodes and $c$ is some constant. The most novel contribution is the concept of global temperature. The maximum force applied to a node is limited by a global temperature value that decays exponentially as the number of iterations increases. This adjustment is meant to help the layout converge on an energy minimum more quickly. Fruchterman and Reingold also provide a grid-based implementation that groups nodes based on proximity and isolates node-pair force computations based on those groupings. This grid-based variant does not improve quality and is only meant to improve running time which is why it is not considered in this treatment. Variants of the Fruchterman-Reingold approach are still popularly used today.

### 5.3 Kamada-Kawai

The Eades and Fruchterman-Reingold class of algorithms simply aim to place adjacent nodes some preferred length apart and push non-adjacent nodes away from each other. The Kamada-Kawai algorithm [19] takes a more sophisticated approach by determining a preferred display distance between each node pair as their graph-theoretic distance. They tackle the problem as an energy function optimization problem. A spring is modeled for each node-pair in the graph that is proportional to the graph theoretic distance of the node-pair. On each iteration, the potential energy of each node induced by its connected springs is calculated, and the node with the highest potential energy is chosen for relocation. The new stable position for the node is calculated using the Newton-Raphson method to minimize its energy function with respect to the current layout.

### 5.4 Harel and Koren

Force-directed layouts are prone to falling into bad local minima when dealing with larger graphs. The methods outlined thus far have difficulty resolving global entanglements even for very sparse graphs. The multi-scale (or "multi-level") approach introduced by [14] tackles this problem by performing layout refinement in phases. The general idea consists of laying out simpler, stand-in versions of the graph first to complete a rough global layout early, so that later phases may focus on the finer details. Choosing the stand-in versions of the graph is known as graph coarsening. The graph coarsening method and the layout method performed on the coarsened graphs is variable. The multi-scale method implemented in this paper is that of Harel and Koren [15] which constructs coarsened graphs by approximating graph centroids to stand-in for vertex neighborhoods. These centroids are then laid out using the Kamada-Kawai heuristic. Other multi-scale approaches include [8], [32], [13]. The added benefit of multi-scale methods is that they can reduce computational complexity by restricting the number of total force/energy calculations yielding faster layouts.

### 5.5 Stress Majorization

As will be shown, the Kamada-Kawai method suffers from its optimization technique (the Newton-Raphson method) causing severe slowdown and instability on some graph structures. The approach taken by Gansner and Koren [9] borrow Kamada-Kawai's energy function and apply a different optimization technique well-known in the field of multidimensional scaling called "stress majorization". This contribution provides a robust improvement to the Kamada-Kawai method.

### 5.6 ForceAtlas2

ForceAtlas2 [18] is the layout method used in the network visualization software Gephi [1]. It was designed by the Gephi team to be used in a continuous setting in which users can manipulate the graph layout to their needs, typically interacting with social/biological networks. ForceAtlas2 represents the modern force-directed layout algorithm because it is built for a specific use case, and it is a composition of several techniques. ForceAtlas2 borrows heavily from the works [6], [17], [28]. It is a Fruchterman-Reingold style implementation with a modified force model specialized to reveal community structures. It also implements both a global and local adaptive temperature scheme to produce a smooth experience. Also in service of a smooth user experience, ForceAtlas2 does not utilize any phased methods such as multi-scale algorithms. Another example of a modern layout algorithm is implemented in OpenOrd [26]. The implementation of ForceAtlas2 is borrowed from the Graphology standard library [30].

## 6 Results

Some observations and comparisons illuminated by the interactive tool are provided in this section.

### 6.1 Random trees

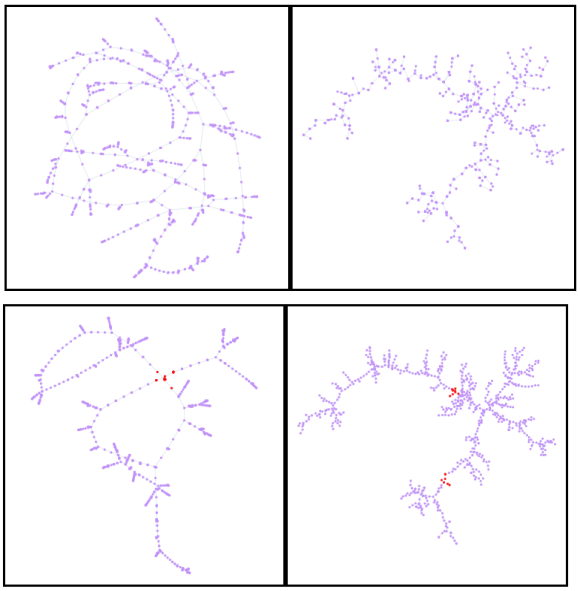

Fig. 1. Illustration of the partial layouts computed by ForceAtlas2 (left) and Harel-Koren (right) on a large random tree (688 vertices) at equal iterations: in the early stage of refinement (top) and later (bottom)

An excellent illustration of the benefits to a multi-scale scheme is shown in Figure 1. Harel and Koren's approach resolves global entanglements very early on while ForceAtlas2 fails to resolve them all even after a protracted runtime. The main issue is that Fruchterman-Reingold style methods will spend much time trying to resolve large branch crossings by trying to snake one branch through another. The friction caused by each branch's nodes makes this process slow, revealing that ForceAtlas2's local temperature schedule is not robust enough to handle these situations. Ideally, if a particular vertex has been moving in one direction over many iterations, then that vertex's temperature should reliably increase accordingly to speed things up.

This drawback can also be observed in any graph with elongated structures such as a thin torus (see Figure 2)

### 6.2 Mesh structures

Absolute drawbacks of some methods compared to others are also easily illuminated in the interactive tool. For regular mesh structures such as grids and 3D shapes, Kamada-Kawai falters against the simpler Fruchterman-Reingold style methods; refer to Figure 3. Kamada-Kawai's optimization technique (Newton-Raphson method) only allows one node position to be refined per iteration. This causes one node to inch in one direction, affecting its many close neighbor's springs only slightly, and thus causing them to only inch slightly in the right direction in later iterations. This results in severe slowdown overall, but particularly in uniform graphs where nodes have larger neighborhoods.

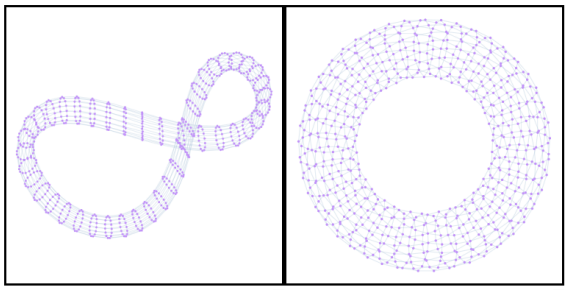

Fig. 2. Illustration of partial layouts computed by ForceAtlas2 without gravitational effects (left) and Gansner (right) on a thin torus

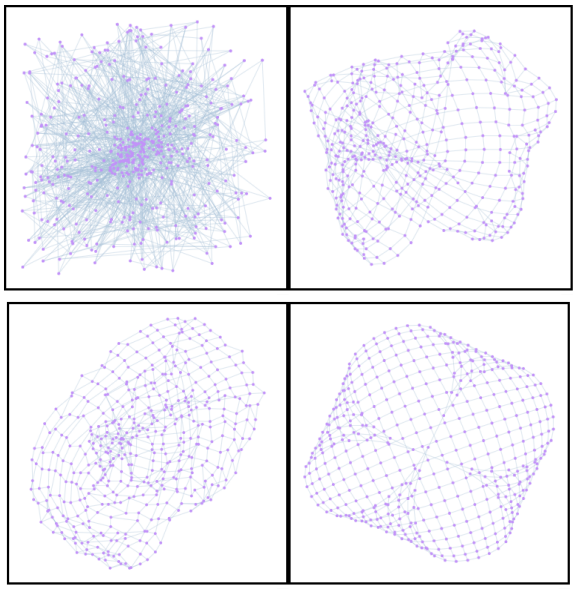

Fig. 3. Illustration of the partial layouts computed by Kamada-Kawai (left) and Fruchterman-Reingold (right) on a folded fabric at equal iterations: in the early stage of refinement (top) and later (bottom)

The improvement made by Gansner's method utilizes a more robust and efficient optimization technique that allows all node positions to be refined per iteration thus entirely eliminating Kamada's drawback. This improvement allows Gansner to efficiently create extremely appealing layouts for regular mesh structures when compared to Fruchterman-Reingold style layouts because it utilizes Kamada's energy function which is suited for mesh structures (see Figure 4). Gansner produces uniform edge lengths and a regular shape, while Fruchterman-Reingold becomes stuck at a local minima and exhibits shorter edge lengths at the fabric's edges.

### 6.3 Community structures

Girvan-Newman graphs [11] are synthetic graphs meant to be used for studying community structures often found in social and biological networks. Because ForceAtlas2 was designed with such structures in mind, we can see how its specialized force model outperforms those of Fruchterman-Reingold and Kamada-Kawai (see Figure 5). ForceAtlas2 computes degree-dependent repulsion forces causing clusters to separate out, allowing the communities to be identified much more easily.

The same observations can be made when looking at the real-world data of Last.fm users' social networks [31] (see Figure 6). Another observation is a drawback of the classical Fruchterman-Reingold force model which tends to splay out visualizations unnecessarily. ForceAtlas2's force model incorporates gravitational forces in its default settings to keep things relatively compact.

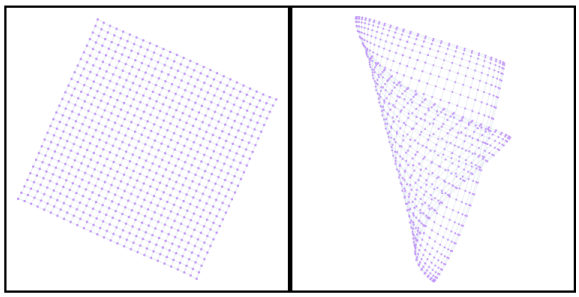

Fig. 4. Illustration of the final layouts computed by Gansner (left) and Fruchterman-Reingold (right) on a flat fabric

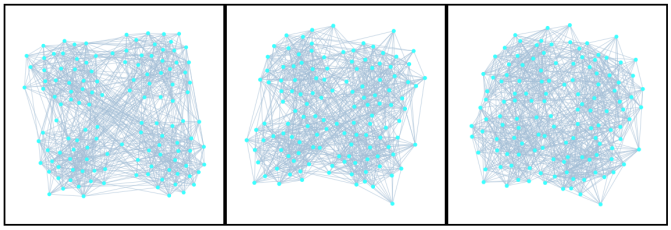

Fig. 5. Illustration of the layouts computed by ForceAtlas2 (left), Fruchterman-Reingold (center), and Gansner (right) on a Girvan-Newman graph

## 7 Tool Access

The interactive tool is provided in the src/ subdirectory. It is meant to be run in the browser by opening index.html. Different graph types can be chosen, and multiple layout methods can be simulated side-by-side. Add a layout method by clicking its corresponding button (multiple of the same method can be added), and remove a layout method by right-clicking its corresponding button. To keep track of certain nodes, brush them in one of the layout views to color them in the other views.

## 8 Conclusion

Interactive tools exist for analyzing data with force-directed methods such as Gephi [1], but to my knowledge there are no interactive tools for analyzing the force-directed algorithms themselves. This work has made it apparent that such a tool could be a boon for experts in deciding which layout methods are should be used for certain data types and exactly how they are good. Despite the lack of sophistication, the interactive tool provided by this paper can provide users insights into these layout methods quite effectively.

## 9 Future Work

The interactive tool provided allows users to examine and compare the runtimes of various layout algorithms against various types of graph data. However, most modern algorithms used in practice are amalgamations of these various techniques essentially choosing from a pool of force models, adaptive temperature schemes, and multi-scale phasing schemes. The most utility this tool could possibly provide would be an interactive setting in which users can create their own force-directed layout algorithm operating on their own data and easily compare different schemes they have created. This would also allow newer methods [24] [21] [34] [25] [12] to be easily used since they do not constitute entire algorithms themselves but components to be used in a larger layout algorithm.

On top of this, it might be useful to add more layers of interactivity enjoyed by other similar tools such as the ability to manipulate the graphs directly, or extend it to handle more classes of graphs such as weighted, directed, or multi-edge graphs.

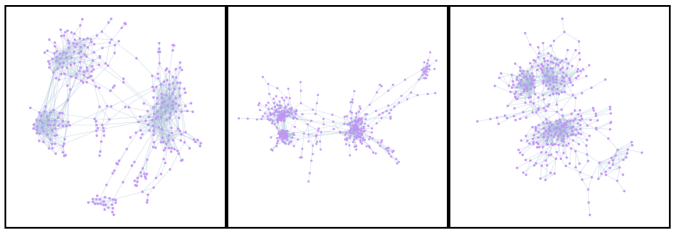

Fig. 6. Illustration of the layouts computed by ForceAtlas2 (left), Fruchterman-Reingold (center), and Gansner (right) on a social network of Last.fm users

## REFERENCES


[1] M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Proceedings of the international AAAI conference on web and social media*, vol. 3, pp. 361–362, 2009.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec 2011. doi: 10.1109/TVCG.2011.185

[3] R. Chernobelskiy, K. I. Cunningham, M. T. Goodrich, S. G. Kobourov, and L. Trott. Force-directed lombardi-style graph drawing. In *Graph Drawing: 19th International Symposium, GD 2011, Eindhoven, The Netherlands, September 21-23, 2011, Revised Selected Papers 19*, pp. 320–331. Springer, 2012.

[4] P. Eades. A heuristic for graph drawing. 1984.

[5] A. Efrat, Y. Hu, S. G. Kobourov, and S. Pupyrev. Mapsets: Visualizing embedded and clustered graphs. In *Graph Drawing: 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers 22*, pp. 452–463. Springer, 2014.

[6] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In *Graph Drawing: DIMACS International Workshop, GD'94 Princeton, New Jersey, USA, October 10–12, 1994 Proceedings 2*, pp. 388–403. Springer, 1995.

[7] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[8] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. In *Graph Drawing'00 Conference Proceedings*, pp. 211–221, 2000.

[9] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Graph Drawing: 12th International Symposium, GD 2004, New York, NY, USA, September 29-October 2, 2004, Revised Selected Papers 12*, pp. 239–250. Springer, 2005.

[10] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization*, 12(3-4):324–357, 2013.

[11] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[12] R. Gove. A random sampling o (n) force-calculation algorithm for graph layouts. In *Computer Graphics Forum*, vol. 38, pp. 739–751. Wiley Online Library, 2019.

[13] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Graph Drawing: 12th International Symposium, GD 2004, New York, NY, USA, September 29-October 2, 2004, Revised Selected Papers 12*, pp. 285–295. Springer, 2005.

[14] R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '99, p. 262–277. Springer-Verlag, Berlin, Heidelberg, 1999.

[15] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, p. 282–285. Association for Computing Machinery, New York, NY, USA, 2000. doi: 10.1145/345513.345353

[16] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*, vol. 28, pp. 983–990. Wiley Online Library, 2009.

[17] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.

[18] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014.

[19] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[20] R. Klapaukh. An empirical evaluation of force-directed graph layout. 2014.

[21] Y.-J. Ko and H.-C. Yen. Drawing clustered graphs using stress majorization and force-directed placements. In *2016 20th International Conference Information Visualisation (IV)*, pp. 69–74. IEEE, 2016.

[22] S. G. Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[23] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*, pp. 933–943, 2018.

[24] C.-C. Lin and H.-C. Yen. A new force-directed graph drawing method based on edge–edge repulsion. *Journal of Visual Languages & Computing*, 23(1):29–42, 2012.

[25] F. Lipp, A. Wolff, and J. Zink. Faster force-directed graph drawing with the well-separated pair decomposition. In *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers 23*, pp. 52–59. Springer, 2015.

[26] S. Martin, W. M. Brown, R. Klavans, and K. W. Boyack. Openord: an open-source toolbox for large graph layout. In *Visualization and Data Analysis 2011*, vol. 7868, pp. 45–55. SPIE, 2011.

[27] A. Meidiana, S.-H. Hong, P. Eades, and D. Keim. A quality metric for visualization of clusters in graphs. In *Graph Drawing and Network Visualization: 27th International Symposium, GD 2019, Prague, Czech Republic, September 17–20, 2019, Proceedings 27*, pp. 125–138. Springer, 2019.

[28] A. Noack. Energy models for graph clustering. *J. Graph Algorithms Appl.*, 11(2):453–480, 2007.

[29] A. Ortega Mattsson. Comparing the readability of the force-directed and orthogonal graph layout, 2020.

[30] G. Plique. Graphology, a robust and multipurpose Graph object for JavaScript., Mar. 2023. doi: 10.5281/zenodo.7695163

[31] B. Rozemberczki and R. Sarkar. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pp. 1325–1334, 2020.

[32] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Graph Drawing: 8th International Symposium, GD 2000 Colonial Williamsburg, VA, USA, September 20–23, 2000 Proceedings 8*, pp. 171–182. Springer, 2001.

[33] E. Welch and S. Kobourov. Measuring symmetry in drawings of graphs. In *Computer Graphics Forum*, vol. 36, pp. 341–351. Wiley Online Library, 2017.

[34] H.-Y. Wu, M. Nöllenburg, and I. Viola. Multi-level area balancing of clustered graphs. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2682–2696, 2020.