Force-Directed Graphs

Debjyoti Ghosh*, Muhammad Huzaifah†, Bastian Morath‡ and Smitha Sheshadri§
School of Computing,
National University of Singapore
Email: *e0029893@u.nus.edu, †e0029863@u.nus.edu, ‡e0386231@u.nus.edu, §e0338018@u.nus.edu

***Abstract*—Graph visualization of complex networks is tricky due to the number of nodes, the relationship between them that needs to be made clear and any additional constraints on the nodes. We provide an introduction to *force-directed graph drawing algorithms*, which try to model the nodes as physical systems. We take a peek at the evolution of the algorithm from the pioneer approaches to recent advancements and also introduce some tools and platforms which target rendering of force-directed layouts.**

I. Introduction

Graphs are a convenient way to show the relationship between objects, for example in social network analysis, cartography or bioinformatics. A graph visualization should be aesthetically pleasing. While there is no universal standard about aesthetics, it is widely agreed that edges with comparable length and high overall symmetry as advantageous. [DBETT94]. Science and engineering routinely take inspiration from the natural world while designing systems which are realistic and familiar. One such analogous model for visualizing large network graphs are *force-directed drawing algorithms*. Those algorithms calculate the layout of nodes by simulating a physical system of bodies. This results in a natural and stable node layout, as seen at an example in Figure 7.

Our work is mainly based on a paper by G. Kobourov [Kob12].

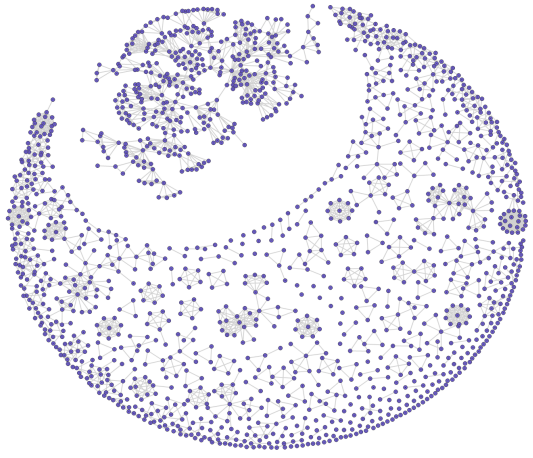

Fig. 1. Graph drawn with the Kamada-Kawai layout, taken from https://plot.ly/python/igraph-networkx-comparison

II. Overview

Force-directed algorithms typically integrate attractive forces between adjacent vertices with repulsive forces between pairs of vertices. An energy function is assigned to a state of the graph, measuring how aesthetically pleasant the graph is. The algorithm then seeks to find a state with minimum energy.

In the following, we briefly discuss the main works done in the area of force-directed algorithms.

A. Tutte 1963

Force-directed graphs date back to work done by Tutte [Tut63]. His algorithm guarantees that the drawn graph is crossing-free. The idea of his algorithm is to fix the vertices of one face of a graph, and then solving a system of linear equations in order to compute the position of all other vertices.

The force due to an edge (u, v) is proportional to the distance between the two vertices u and v. Thus, a force at a vertex v is described by the following force function

$$F(v) = \sum_{(u,v)\in E} (p_u - p_v),$$

where p_u and p_v are the positions of vertices u and v. Using partial derivatives, we can find the x and y-coordinate of the vertices.

B. Eades 1984

One of the first force-directed algorithms was invented by Eades in 1984 [Ead84]. Each edge is replaced with a logarithmic strength spring and each vertex with repellent steel rings. The vertices are placed in some (arbitrary) state and then let go. The spring and repellent forces move the system to a minimal energy state. Algorithm 1 shows the pseudo code of Eades algorithm, on which most of the later algorithms are based on.

Algorithm 1 Eades algorithm

```
1: procedure SPRING(G, graph)
2:     place vertices of G in random locations
3:     while minimal energy not achieved do
4:         calculate the force on each vertex
5:         move the vertex 0.1 * (force on vertex)
6:     end while
7:     draw graph
8: end procedure
```

C. Fruchtman and Reingold 1991

Fruchtman and Reingold's algorithm [FR91] additionally encapsulates the idea that the vertices should be evenly distributed. This is achieved by introducing the notion of *temperature*. Starting at a high temperature, it gradually decays to 0. The lower the temperature and thus the better the layout, the smaller the adjustments become.

D. Graph Theoretic Distance Approaches

While the before mentioned algorithms try to keep vertices connected by an edge close to each other, but ensuring that they are not too close, there is also an other approach, taken by Kamada and Kawai [KK$^+$89].

In their approach, a perfect graph drawing would be one in which the geometric distance between any two vertices matches the theoretic shortest path distance as much as possible. If a pair of vertices is geometrically closer or farther than their corresponding theoretic distance, the vertices repel and attract each other, respectively. There is no notion of attractive and repulsive forces as in the before mentioned algorithms.

E. Dealing with large graphs

The main problem with force-directed graph drawing algorithms is that they are relatively expensive and thus not suited for large graphs, that is graphs with more than about 1000 vertices. Hadany and Harel 1999 [HH01] introduced a technique that consists of first considering an abstraction of a graph, which is first drawn. This would produce a rough layout of the final graph. Then details are added and the layout is corrected. It is important to precisely define the coarse-scale representation, as essential features of the graph should be retained by the abstraction, while not-so important ones should only be considered in the second step.

III. Recent Advancements

Several modified and optimised algorithms which use a force-directed drawing algorithm heir core have been proposed. In this section we introduce some of them.

A. Hu, Y. 2005

One of the two limitations addressed by Yifan Hu [Hu05] is that physical systems tend to settle at a local minimum which might not be the optimal solution. The work also addresses the high computational complexity of the predecessors. [FR91] requires a $O(V^2)$ complexity and [KK$^+$89] has a $O(|V||E|)$ cost, where V is the number of vertices of the graph and E is the number of edges. His work proposes to have an efficiency comparable to [Wal00] and to give better results on several difficult problems. The algorithm implements an adaptive cooling scheme to find an optimal low energy state for the system.

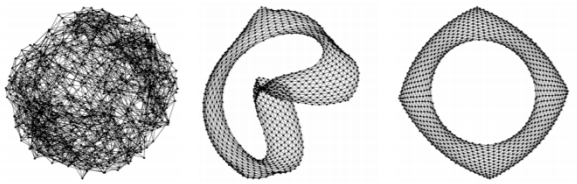

Fig. 2. This figure shows the initial state of the graph, the graph after 70 iterations of Hu's algorithm, and the optimal state, from left to right. Figure taken from [Hu05].

B. Holten and Van Wijk 2009

Visual clusters due to large number of nodes and edges is a crucial issue to be addressed by any graph drawing algorithm, particularly those which target large and complex networks. Holten and Van Wijk [HVW09] propose a self-organizing approach for edge bundling. They use *force based edge bundling (FBEB)*, which models edges as flexible springs that can attract each other, keeping the position of nodes fixed. They use a single parameter K to control the amount of bundling, which allows them to achieve a significant cluster reduction and high clarity.

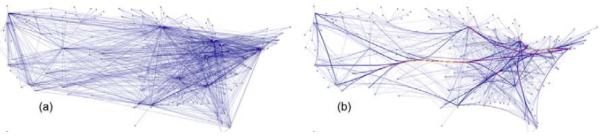

Fig. 3. A complex graph of US flights and airports which is a) unbundled and b) bundled using FDEB. Figure taken from [HVW09].

IV. Tools and platforms

This section lists and gives brief descriptions of several tools designed to support creation and rendering of force directed layouts.

A. Graphael

Grapheal[1] is a light weight platform to render force directed layouts and was developed in 2003. It also provides support for non-Euclidean geometries and allows for dynamic visualizations.

[1]Graphael.cs.arizona.edu. (2019). Graphael Homepage. [online] Available at: http://graphael.cs.arizona.edu/ [Accessed 30 Mar. 2019]

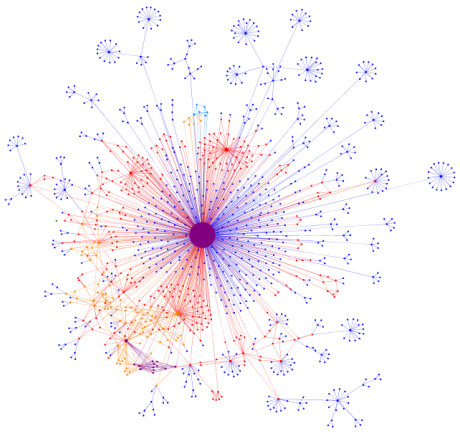

Fig. 4. Illustration of a force directed graph generated by Graphael, taken from their website[3].

B. Gephi

Gephi[4] is an open source and free software for graph visualizations. The thorough tutorials are a big plus for anyone to get started on.

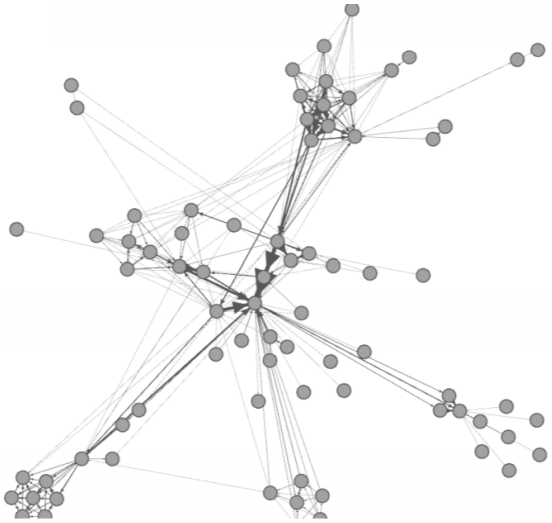

Fig. 5. Illustration of a force directed graph generated by Gephi, taken from their website[6]

C. Springy

Springy[7] is a force-directed graph drawing tool written in JavaScript. It is designed to be simple and light. It can be integrated with multiple formats including canvas, SVG and WebGL. The website provides a interactive graph which adjusts the layout as soon as the user changes a node position.

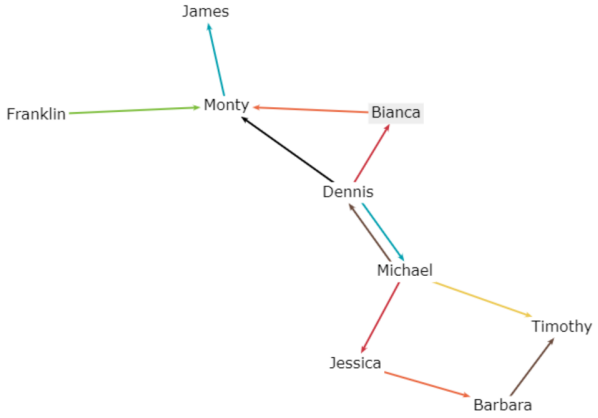

Fig. 6. Illustration of a force directed graph generated by Springy

D. Vlamis Force Directed Graph Plugin

Created in collaboration with Oracle, *Vlamis Force Directed Graph Plugin*[8] allows the creation of an interactive node and link diagram. The smooth transitions when node positions are changed makes the platform suitable for several applications like product hierarchies, field service organizations, and time hierarchies.

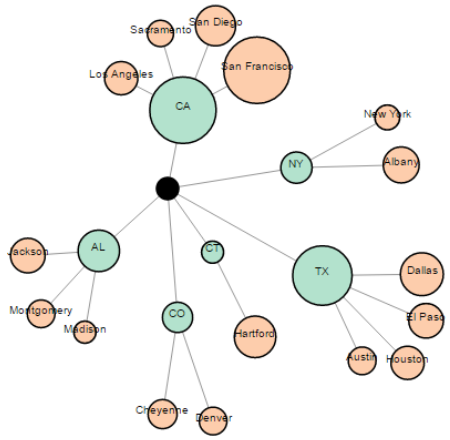

Fig. 7. A simple graph created using Vlamis Force Directed Graph Plugin

V. Concluding Remarks

Force-directed algorithms are a convenient way to draw aesthetically pleasing graphs. The extremely natural and intuitive mechanism behind the algorithm, which models physical systems, results in clear and comprehensible visualizations.

[3]Graphael.cs.arizona.edu. (2019). Graphael Homepage. [online] Available at: http://graphael.cs.arizona.edu/ [Accessed 30 Mar. 2019].

[4]Gephi.org. (2019). Gephi Homepage. [online] Available at: https://gephi.org/ [Accessed 30 Mar. 2019].

[6]Gephi.org. (2019). Tutorial Layouts. [online] Available at: https://gephi.org/users/tutorial-layouts/ [Accessed 30 Mar. 2019].

[7]Getspringy.com. (2019). Springy - A force directed graph layout algorithm in JavaScript.. [online] Available at: http://getspringy.com [Accessed 30 Mar. 2019].

[8]Vlamis Software Solutions. (2019). Force Directed Graph Plugin Vlamis Software Solutions. [online] Available at: http://www.vlamis.com/force-directed-graph-plugin [Accessed 30 Mar. 2019].

Although initial algorithms proposed in the 1980s were quite slow and only managed to handle tens of nodes, more recent developments facilitate more complex structures. There are several tools specifically designed for creating force directed layouts as illustrated in the previous section.

REFERENCES

[DBETT94] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[Ead84] Peter Eades. A heuristic for graph drawing. *Congressus numerantium*, 42:149–160, 1984.

[FR91] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[HH01] Ronny Hadany and David Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113(1):3–21, 2001.

[Hu05] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[HVW09] Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

[KK+89] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[Kob12] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[Tut63] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.

[Wal00] Chris Walshaw. A multilevel algorithm for force-directed graph drawing. In *International Symposium on Graph Drawing*, pages 171–182. Springer, 2000.