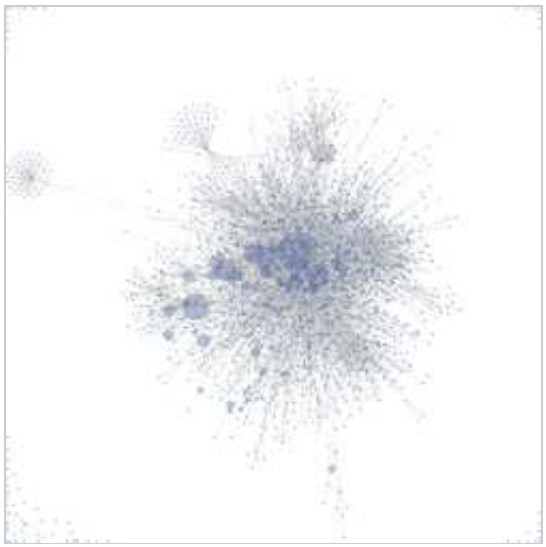
Force-directed graph drawing

From Wikipedia, the free encyclopedia

Force-directed graph drawing algorithms are a class of algorithms for drawing graphs in an aesthetically pleasing way. Their purpose is to position the nodes of a graph in two-dimensional or three-dimensional space so that all the edges are of more or less equal length and there are as few crossing edges as possible, by assigning forces among the set of edges and the set of nodes, based on their relative positions, and then using these forces either to simulate the motion of the edges and nodes or to minimize their energy.[1]

While graph drawing can be a difficult problem, force-directed algorithms, being physical simulations, usually require no special knowledge about graph theory such as planarity.



Visualization of links between pages on a wiki using a force-directed layout.

Contents

- 1 Forces
- 2 Methods
- 3 Advantages
- 4 Disadvantages
- 5 History
- 6 See also
- 7 References
- 8 Further reading
- 9 External links

Forces

Force-directed graph drawing algorithms assign forces among the set of edges and the set of nodes of a graph drawing. Typically, spring-like attractive forces based on Hooke's law are used to attract pairs of endpoints of the graph's edges towards each other, while simultaneously repulsive forces like those of electrically charged particles based on Coulomb's law are used to separate all pairs of nodes. In equilibrium states for this system of forces, the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart (because of the electrical repulsion). Edge attraction and vertex repulsion forces may be defined using functions that are not based on the physical behavior of springs and particles; for instance, some force-directed systems use springs whose attractive force is logarithmic rather than linear.

An alternative model considers a spring-like force for every pair of nodes (i, j) where the ideal length δ_{ij} of each spring is proportional to the graph-theoretic distance between nodes *i* and *j*, without using a separate repulsive force. Minimizing the difference (usually the squared difference) between Euclidean and ideal distances between nodes is then equivalent to a metric multidimensional scaling problem.

A force-directed graph can involve forces other than mechanical springs and electrical repulsion. A force analogous to gravity may be used to pull vertices towards a fixed point of the drawing space; this may be used to pull together different connected components of a disconnected graph, which would otherwise tend to fly apart from each other because of the repulsive forces, and to draw nodes with greater centrality to more central positions in the drawing;[2] it may also affect the vertex spacing within a single component. Analogues of magnetic fields may be used for directed graphs. Repulsive forces may be placed on edges as well as on nodes in order to avoid overlap or near-overlap in the final drawing. In drawings with curved edges such as circular arcs or spline curves, forces may also be placed on the control points of these curves, for instance to improve their angular resolution.[3]

Methods

Once the forces on the nodes and edges of a graph have been defined, the behavior of the entire graph under these sources may then be simulated as if it were a physical system. In such a simulation, the forces are applied to the nodes, pulling them closer together or pushing them further apart. This is repeated iteratively until the system comes to a mechanical equilibrium state; i.e., their relative positions do not change anymore from one iteration to the next. The positions of the nodes in this equilibrium are used to generate a drawing of the graph.

For forces defined from springs whose ideal length is proportional to the graph-theoretic distance, stress majorization gives a very well-behaved (i.e., monotonically convergent)[4] and mathematically elegant way to minimise these differences and, hence, find a good layout for the graph.

It is also possible to employ mechanisms that search more directly for energy minima, either instead of or in conjunction with physical simulation. Such mechanisms, which are examples of general global optimization methods, include simulated annealing and genetic algorithms.

Advantages

The following are among the most important advantages of force-directed algorithms:

Good-quality results
: At least for graphs of medium size (up to 50–100 vertices), the results obtained have usually very good results based on the following criteria: uniform edge length, uniform vertex distribution and showing symmetry. This last criterion is among the most important ones and is hard to achieve with any other type of algorithm.

Flexibility
: Force-directed algorithms can be easily adapted and extended to fulfill additional aesthetic criteria. This makes them the most versatile class of graph drawing algorithms. Examples of existing extensions include the ones for directed graphs, 3D graph drawing,[5] cluster graph drawing, constrained graph drawing, and dynamic graph drawing.

Intuitive
: Since they are based on physical analogies of common objects, like springs, the behavior of the algorithms is relatively easy to predict and understand. This is not the case with other types of graph-drawing algorithms.

Simplicity
: Typical force-directed algorithms are simple and can be implemented in a few lines of code. Other classes of graph-drawing algorithms, like the ones for orthogonal layouts, are usually much more involved.

Interactivity

Another advantage of this class of algorithm is the interactive aspect. By drawing the intermediate stages of the graph, the user can follow how the graph evolves, seeing it unfold from a tangled mess into a good-looking configuration. In some interactive graph drawing tools, the user can pull one or more nodes out of their equilibrium state and watch them migrate back into position. This makes them a preferred choice for dynamic and online graph-drawing systems.

Strong theoretical foundations

While simple *ad-hoc* force-directed algorithms often appear in the literature and in practice (because they are relatively easy to understand), more reasoned approaches are starting to gain traction. Statisticians have been solving similar problems in multidimensional scaling (MDS) since the 1930s, and physicists also have a long history of working with related n-body problems - so extremely mature approaches exist. As an example, the stress majorization approach to metric MDS can be applied to graph drawing as described above. This has been proven to converge monotonically.[4] Monotonic convergence, the property that the algorithm will at each iteration decrease the stress or cost of the layout, is important because it guarantees that the layout will eventually reach a local minimum and stop. Damping schedules cause the algorithm to stop, but cannot guarantee that a true local minimum is reached.

Disadvantages

The main disadvantages of force-directed algorithms include the following:

High running time

The typical force-directed algorithms are in general *considered* to have a running time equivalent to $O(n^3)$, where n is the number of nodes of the input graph. This is because the number of iterations is estimated to be O(n), and in every iteration, all pairs of nodes need to be visited and their mutual repulsive forces computed. This is related to the N-body problem in physics. However, since repulsive forces are local in nature the graph can be partitioned such that only neighboring vertices are considered. Common techniques used by algorithms for determining the layout of large graphs include high-dimensional embedding,[6] multi-layer drawing and other methods related to N-body simulation. For example, the Barnes–Hut simulation-based method FADE[7] can improve running time to n*log(n) per iteration. As a rough guide, in a few seconds one can expect to draw at most 1,000 nodes with a standard n^2 per iteration technique, and 100,000 with a n*log(n) per iteration technique.[7] Force-directed algorithm, when combined with a multilevel approach, can draw graphs of millions of nodes.[8]

Poor local minima

It is easy to see that force-directed algorithms produce a graph with minimal energy, in particular one whose total energy is only a local minimum. The local minimum found can be, in many cases, considerably worse than a global minimum, which translates into a low-quality drawing. For many algorithms, especially the ones that allow only *down-hill* moves of the vertices, the final result can be strongly influenced by the initial layout, that in most cases is randomly generated. The problem of poor local minima becomes more important as the number of vertices of the graph increases. A combined application of different algorithms is helpful to solve this problem.[9] For example, using the Kamada–Kawai algorithm[10] to quickly generate a reasonable initial layout and then the Fruchterman–Reingold algorithm[11] to improve the placement of neighbouring nodes. Another technique to achieve a global minimum is to use a multilevel approach.

History

Force-directed methods in graph drawing date back to the work of Tutte (1963), who showed that polyhedral graphs may be drawn in the plane with all faces convex by fixing the vertices of the outer face of a planar embedding of the graph into convex position, placing a spring-like attractive force on each edge, and letting the system settle into an equilibrium.[12] Because of the simple nature of the forces in this case, the system cannot get stuck in local minima, but rather converges to a unique global optimum configuration. Because of this work, embeddings of planar graphs with convex faces are sometimes called Tutte embeddings.

The combination of attractive forces on adjacent vertices, and repulsive forces on all vertices, was first used by Eades (1984);[13] additional pioneering work on this type of force-directed layout was done by Fruchterman & Reingold (1991).[11] The idea of using only spring forces between all pairs of vertices, with ideal spring lengths equal to the vertices' graph-theoretic distance, is from Kamada & Kawai (1989).[10]

See also

- Cytoscape, software for visualising biological networks. The base package includes force-directed layouts as one of the built-in methods.
- Gephi, an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.
- Graphviz, software that implements a multilevel force-directed layout algorithm (among many others) capable of handling very large graphs.
- Tulip, software that implements most of the force-directed layout (GEM, LGL, GRIP, FM³).
- Prefuse

References

1. ^ Kobourov, Stephen G. (2012), *Spring Embedders and Force-Directed Graph Drawing Algorithms*, arXiv:1201.3011 (//arxiv.org/abs/1201.3011).
2. ^ Bannister, M. J.; Eppstein, D.; Goodrich, M. T.; Trott, L. (2012), "Force-directed graph drawing using social gravity and scaling", *Proc. 20th Int. Symp. Graph Drawing*, arXiv:1209.0748 (//arxiv.org/abs/1209.0748).
3. ^ Chernobelskiy, R.; Cunningham, K.; Goodrich, M. T.; Kobourov, S. G.; Trott, L. (2011), "Force-directed Lombardi-style graph drawing" (http://www.cs.arizona.edu/~kobourov/fdl.pdf), *Proc. 19th Symposium on Graph Drawing*, pp. 78–90.
4. ^ ***a*** ***b*** de Leeuw, Jan (1988), "Convergence of the majorization method for multidimensional scaling", *Journal of Classification* (Springer) **5** (2): 163–180, doi:10.1007/BF01897162 (http://dx.doi.org/10.1007%2FBF01897162).
5. ^ Vose, Aaron. "3D Phylogenetic Tree Viewer" (http://www.aaronvose.com/phytree3d/). Retrieved 3 June 2012.
6. ^ Harel, David; Koren, Yehuda (2002), "Graph drawing by high-dimensional embedding" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.5390), *Proceedings of the 9th International Symposium on Graph Drawing*, pp. 207–219, ISBN 3-540-00158-1
7. ^ ***a*** ***b*** Quigley, Aaron; Eades, Peter (2001), "FADE: Graph Drawing, Clustering, and Visual Abstraction" (http://www.cs.ucd.ie/staff/aquigley/home/downloads/aq-gd2000.pdf) (PDF), *Proceedings of the 8th International Symposium on Graph Drawing*, pp. 197–210, ISBN 3-540-41554-8.
8. ^ "A Gallery of Large Graphs" (http://www2.research.att.com/~yifanhu/GALLERY/GRAPHS/). Retrieved 1 July 2012.
9. ^ Collberg, Christian; Kobourov, Stephen; Nagra, Jasvir; Pitts, Jacob; Wampler, Kevin (2003), "A System for Graph-based Visualization of the Evolution of Software"

(http://www.researchgate.net/publication/2851716_A_System_for_Graph-Based_Visualization_of_the_Evolution_of_Software/file/32bfe510fcc3a2ac65.pdf), *Proceedings of the 2003 ACM Symposium on Software Visualization (SoftVis '03)*, New York, NY, USA: ACM, p. 77–86; figures on p. 212, doi:10.1145/774833.774844 (http://dx.doi.org/10.1145%2F774833.774844), ISBN 1-58113-642-0, "To achieve an aesthetically pleasing layout of the graph it is also necessary to employ modified Fruchterman–Reingold forces, as the Kamada–Kawai method does not achieve satisfactory methods by itself but rather creates a good approximate layout so that the Fruchterman-Reingold calculations can quickly "tidy up" the layout."

10. ^ ***a*** ***b*** Kamada, Tomihisa; Kawai, Satoru (1989), "An algorithm for drawing general undirected graphs", *Information Processing Letters* (Elsevier) **31** (1): 7–15, doi:10.1016/0020-0190(89)90102-6 (http://dx.doi.org/10.1016%2F0020-0190%2889%2990102-6).
11. ^ ***a*** ***b*** Fruchterman, Thomas M. J.; Reingold, Edward M. (1991), "Graph Drawing by Force-Directed Placement", *Software – Practice & Experience* (Wiley) **21** (11): 1129–1164, doi:10.1002/spe.4380211102 (http://dx.doi.org/10.1002%2Fspe.4380211102).
12. ^ Tutte, W. T. (1963), "How to draw a graph", *Proceedings of the London Mathematical Society* **13** (52): 743–768.
13. ^ Eades, Peter (1984), "A Heuristic for Graph Drawing", *Congressus Numerantium* **42** (11): 149–160.

Further reading

- di Battista, Giuseppe; Peter Eades, Roberto Tamassia, Ioannis G. Tollis (1999), *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, ISBN 978-0-13-301615-4
- Kaufmann, Michael; Wagner, Dorothea, eds. (2001), *Drawing graphs: methods and models*, Lecture Notes in Computer Science 2025, Springer, doi:10.1007/3-540-44969-8 (http://dx.doi.org/10.1007%2F3-540-44969-8), ISBN 978-3-540-42062-0

External links

- Video of Spring Algorithm (http://www.cs.usyd.edu.au/%7Eaquigley/avi/spring.avi)
- Live visualisation in flash + source code and description (http://blog.ivank.net/force-based-graph-drawing-in-as3.html)
- Daniel Tunkelang's dissertation (http://reports-archive.adm.cs.cmu.edu/anon/1998/abstracts/98-189.html) (with source code (http://www.cs.cmu.edu/~quixote/JiggleSource.zip) and demonstration applet (http://www.cs.cmu.edu/~quixote/gd.html)) on force-directed graph layout
- Hyperassociative Map Algorithm (http://wiki.syncleus.com/index.php/DANN:Hyperassociative_Map)
- Interactive and real-time force-directed graphing algorithms used in an online database modeling tool (http://www.anchormodeling.com/modeler)