Drawing graphs by numerical solution of a system of second order ordinary differential equations

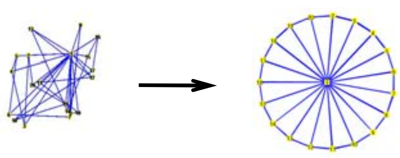
Mohammad Ali Ebrahimi, Michael Monagan.
Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada

Introduction

In this poster we present the spring algorithm for drawing directed and undirected graphs of vertices and edges in 2D or 3D. Starting from random initial position for the vertices, the algorithm finds a local minimal total energy of the graph. While graph drawing is a complicated problem, this algorithm requires no special knowledge about the structure of the graph, The Maple implementation of this algorithm will be used by the Graph Theory Package.

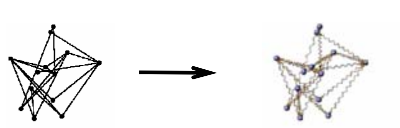
Problem

• Find an aesthetic layout of the graph that clearly conveys its structure.
• Assign a location for each node and a route for each edge, so that the resulting drawing is "nice".



Methods

• Replace edges with springs (zero rest length).
• Replace vertices with electrically charged particles. These particles repel each other.
• Add damping term.

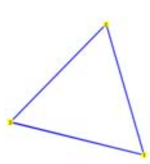


$$\frac{d^2}{dt^2}X_i(t) = \left(\sum_{j=N(i)} (X_i(t) - X_j(t))\right) - \left(\frac{d}{dt}X_i(t)\right) + \left(\sum_{j=V(G)} \frac{|X_i(t) - X_j(t)|}{(X_i(t) - X_j(t))^3}\right)$$

This is a differential equation for the node i

Example

The system for the graph below is:



$$\frac{d^2}{dt^2}x_1(t) = K(2x_1(t) - x_2(t) - x_3(t)) - C\left(\frac{d}{dt}x_1(t)\right) + A\left(\frac{x_1(t)-x_2(t)}{(x_1(t)-x_2(t))^2+(y_1(t)-y_2(t))^2} + \frac{x_1(t)-x_3(t)}{(x_1(t)-x_3(t))^2+(y_1(t)-y_3(t))^2}\right)$$

$$\frac{d^2}{dt^2}y_1(t) = K(2y_1(t) - y_2(t) - y_3(t)) - C\left(\frac{d}{dt}y_1(t)\right) + A\left(\frac{y_1(t)-y_2(t)}{(x_1(t)-x_2(t))^2+(y_1(t)-y_2(t))^2} + \frac{y_1(t)-y_3(t)}{(x_1(t)-x_3(t))^2+(y_1(t)-y_3(t))^2}\right)$$

$$\frac{d^2}{dt^2}x_2(t) = K(2x_2(t) - x_1(t) - x_3(t)) - C\left(\frac{d}{dt}x_2(t)\right) + A\left(\frac{x_2(t)-x_1(t)}{(x_2(t)-x_1(t))^2+(y_2(t)-y_1(t))^2} + \frac{x_2(t)-x_3(t)}{(x_2(t)-x_3(t))^2+(y_2(t)-y_3(t))^2}\right)$$

$$\frac{d^2}{dt^2}y_2(t) = K(2y_2(t) - y_1(t) - y_3(t)) - C\left(\frac{d}{dt}y_2(t)\right) + A\left(\frac{y_2(t)-y_1(t)}{(x_2(t)-x_1(t))^2+(y_2(t)-y_1(t))^2} + \frac{y_2(t)-y_3(t)}{(x_2(t)-x_3(t))^2+(y_2(t)-y_3(t))^2}\right)$$

$$\frac{d^2}{dt^2}x_3(t) = K(2x_3(t) - x_1(t) - x_2(t)) - C\left(\frac{d}{dt}x_3(t)\right) + A\left(\frac{x_3(t)-x_1(t)}{(x_3(t)-x_1(t))^2+(y_3(t)-y_1(t))^2} + \frac{x_3(t)-x_2(t)}{(x_3(t)-x_2(t))^2+(y_3(t)-y_2(t))^2}\right)$$

$$\frac{d^2}{dt^2}y_3(t) = K(2y_3(t) - y_1(t) - y_2(t)) - C\left(\frac{d}{dt}y_3(t)\right) + A\left(\frac{y_3(t)-y_1(t)}{(x_3(t)-x_1(t))^2+(y_3(t)-y_1(t))^2} + \frac{y_3(t)-y_2(t)}{(x_3(t)-x_2(t))^2+(y_3(t)-y_2(t))^2}\right)$$

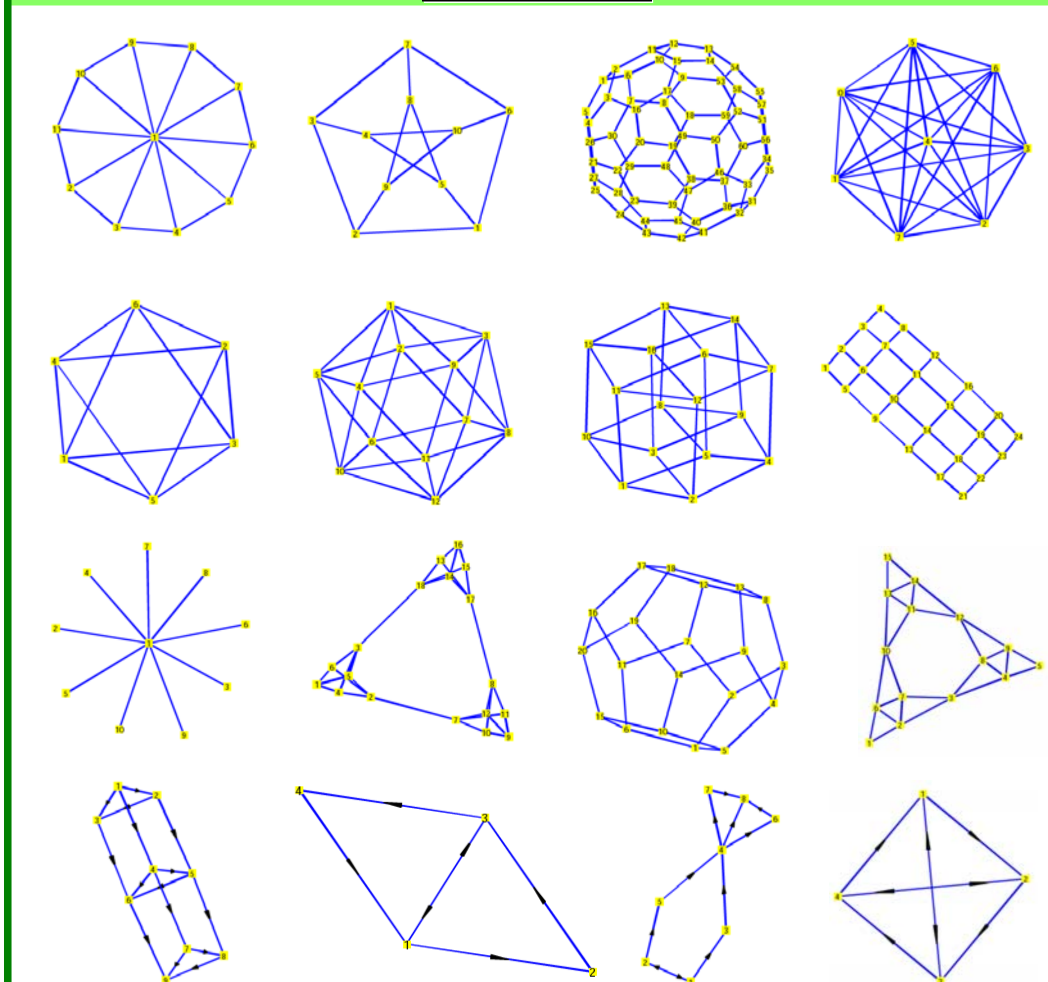
$$\{D(x_2)(0) = 0, D(y_1)(0) = 0, D(x_3)(0) = 0, D(y_3)(0) = 0, D(y_2)(0) = 0, D(x_1)(0) = 0\}$$

$$\{D(y_1)(0) = 0, x_3(0) = rand, y_3(0) = rand, x_1(0) = rand, x_2(0) = rand, y_1(0) = rand, y_2(0) = rand\}$$

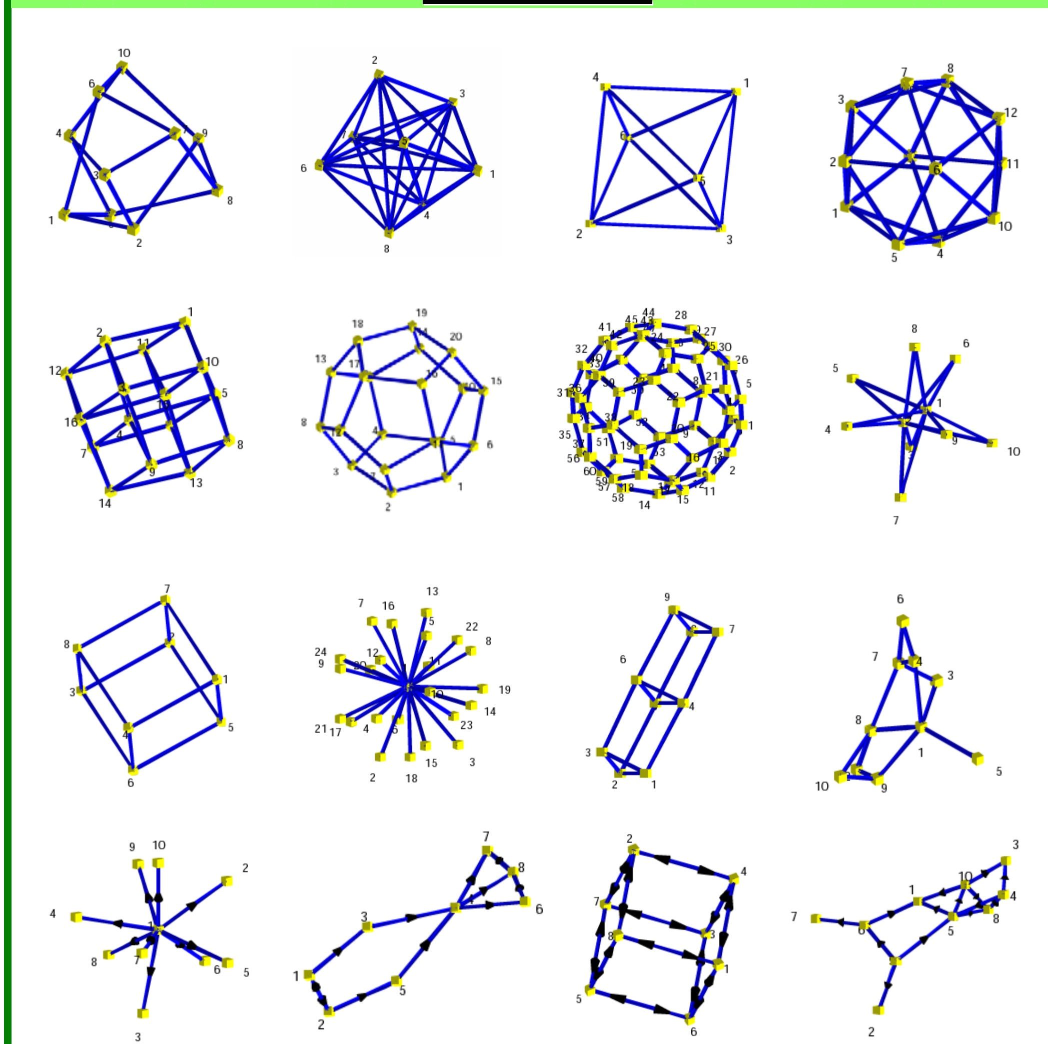
We solve this system by the *Fehlberg fourth-fifth order Runge-Kutta* method (RKF45) with degree four interpolant. This can be done in Maple by the dsolve[numeric] command.

Results

Graphs in 2D



Graphs in 3D



Constants

In order to reconstruct good graphs from the randomly positioned initial vertices it is important that the spring model is not over damped nor under damped. Since we are solving the system of differential equations numerically, in case of over damped or under damped, reaching to the local minimum is almost impossible. To correct this we choose the damping constant, spring constant, and repelling constants carefully. After experimenting with different graphs we have concluded that a dynamic model with respect to different graphs is required.

Damping constant

we suggest that the value of damping constant is proportional to the degree of each node. For example if we have a node with three springs connected to it then there is more damping required for that node.

Spring and repulsion Constants

We suggest that the value of the spring constant and also repulsion constant are directly related. For example if we increase the strength of the spring then we need to have stronger repulsion force. However the relation between the spring and repulsion constant is not linear. We suggest the ratio of

$$\frac{A}{B} = \sqrt{n}$$

where *A* is the repulsion constant, *B* is the spring constant, and *n* is the number of nodes in the graph. On the other hand if we have a node with a high degree then the repulsion force should be small on this node and consequently less force for our spring.
We can summarize all these in the following.
Spring constant:

$$B = \frac{\sqrt{n}}{\deg(n)}$$

Damping constant:

$$\frac{1}{\deg(n_i)}$$

Repulsion constant:

$$A = \frac{1}{\deg(n_i)}$$

where $\deg(n_i)$ is the degree of node *i*.

Options

We have implemented this idea in Maple. In this implementation we have designed the following options:

•**animation**: This options will show an animation of the vertices and edges moving toward the local solution. This way the user can follow how the graph evolves, seeing it unfold from a tangled mess into a good-looking configuration.

•**dimension**: With this option the user can choose to see the graph in 2D or 3D.

•**direction**: With this option the user can choose to see the direction of edges.

Remarks

• We observed that the result of this algorithm is better with random initial positions.
• Since the initial positions are chosen randomly the final positions might be different.
• The major disadvantage is that it is very slow. **order**n^2 where *n* is the number of nodes in 2D.

Acknowledgments

we would like to thank Allan Wittkopf and Simon Lo for their help, suggestions, and great ideas.