**COMP4060**

**Graph Theory**

**Term Project**

**Purpose:**

* Study graph drawing.
* Learn more about graphing algorithms.
* Learn how to represent information well.
* Learn about the Eades algorithm.

**Ulterior Motives:**

* Practice making GUIs.
* Practice for design interviews.
* Practice software usability techniques.
* Learn better programming techniques and practices.
* Try to manage and organize a larger project.
* Practice using command line.

**GUI components:**

Graph details panel (top of screen):

* This panel shows the details of the graph.
* The details include the graph name, the number of vertices, and the number of edges in the graph.
* In future iterations, I would use these details to display more graph details, or use the number of edges and vertices to help find the number of faces in various polyhedrons, then display them.

Choose graph Panel (top left of screen):

* This panel allows a user to pick what graph they want to display.
* Users choose the graph they want from the dropdown list.
* 8 graphs were processed and chosen for this project

Choose line shape panel (below choose graph panel):

* This panel allows a user to choose what type of line they want shown; either octagonal or straight.
* Octagonal lines do not care about cross over, and merge lines together.
* Straight lines go straight from point to point, and act as springs in the “Eades” algorithm

Choose aesthetics panel (below choose line shape panel):

* This panel allows a user to step through the “Eades” algorithm by pressing the button multiple times. This allows the user to see the “spring” action of the edges.
* This panel also has three different buttons for vertex colour choice. Each one can be pressed to get the colour indicated by the button.

**GRAPHS:**

The graphs in this project were taken from: <https://hog.grinvin.org/ViewGraphInfo.action?id=334>

The graphs are formatted as:

*graphName*

*numberVertices*

*-(vertex number and name) adjacent vertex*

*…*

*“coordinates”*

*x y*

***example:***

*StickPerson*

*10*

*-1 5*

*-2 6*

*-3 6*

*-4 5*

*-5 6 7*

*-7 8 10*

*-8 9*

*-9 10*

*coordinates*

*-200 20*

*-140 -245*

*200 -235*

*175 80*

*10 0*

*-30 -220*

*20 90*

*-80 75*

*0 120*

*100 90*

**EADES ALGORITHM:**

I really liked this paper for understanding the “Eades” algorithm.

<http://emr.cs.iit.edu/~reingold/force-directed.pdf>

How the Eades algorithm can be thought of is:

The vertices are magnets with the same charge.

The vertices want to repulse each other.

The edges between vertices are strings.

The edges don’t want to stretch too far, and will try to pull the vertices toward each other.

Together the force of the vertices and the force of the edges will balance, causing fewer and fewer changes in the depiction in the graph.

Edges can be attracting or repelling forces.

If the edge is stretched too much, it brings vertices together.

If the edge is compressed too much, it pushes vertices apart.

Vertices really just try to repel each other.

There were a few examples in this paper of the attracting force and repulsing force that could be used.

This summer, I plan on playing with this project more and trying the different versions of the attracting and repelling calculations that I was able to find.

I’m also having problems working with a “universal” cooling function.

Through the number of iterations, the springy-ness of the edges should lessen as we don’t want a perpetual-motion machine, at least for graph drawing anyway.

Over time the cooling function should downplay the force effect of the edges.

In the paper it is suggested that the correct cooling method be found through trial and error, of which I have more to do.

At this time, I use (nearly) static edge length, which means that the algorithm is not as responsive as I would like for randomly placed vertices.

To combat this, I took vaguely wrong graph formations and used the spring action to move the vertices throughout the screen.

In future iterations, I would try to expand this out to 3D as well, as it is suggested in the paper that this is a possibility as edges still act as springs.

I would really like to merge this project with the polyhedron project from COMP4420.

**FUTURE WORK:**

Expanding this algorithm to 3D.

Let edges be variable length in “final solution”.

Add more graphs.

Try running with graphs without starting coordinates.

Allow users to click on a vertex to get vertex information.

Allow users to click on an edge to get edge information.

Improve the GUI

Allow for choice of graph treatment type (tree, complete, etc.)

Allow for more aesthetic choices such as symmetry, and so on.