Mitchell Anderson a1647264

ASSIGNMENT 1 – PSSD

ALGORITHM

At the most basic level this algorithm simulates a system whereby nodes (towns in the case of Binomia) are represented as electrically charged particles which exert a force of repulsion on each other and edges (roads in Binomia) act as springs. This creates a system whereby nodes tend to be attracted closer towards their connected nodes whilst still being far away enough as to not be on top of each other. This algorithm is inspired by the early work of P. Eades and his experimentation with calculating an "Aesthetically Pleasing" graph (i.e. a graph with minimal edge crossings and uniform edge length).[1][2]

Intuitively it makes sense that when drawing a graph (which is really what the town of Binomia is) that one would want to make the roads connecting the towns together as short as practical. The shorter the roads are the less opportunity they have to intersect with each other. This can be achieved as aforementioned by exerting forces on the nodes.

Essentially this algorithm attempts to simulate a system where the edges between nodes act as springs that drag in nodes that are too far apart and repel nodes which are too close together while the nodes themselves act as electrically charged particles which repel each other.

Below is the extremely basic outline of the pseudo code

Given a graph with $G(V, E)$ where V is the set of vertices, E is the set of edges

```
FOR EACH u ∈ V
        FOR EACH v ∈ V
                IF u ≠ v
                        CALC-COULOMB-FORCE(v)
                END IF
        END FOR
        FOR EACH e ∈ E(u,v)
                CALC-HOOKES-FORCE(v)
        END FOR
        FOR EACH v ∈ V
                REPOSITION(v)
        END FOR
END FOR
```

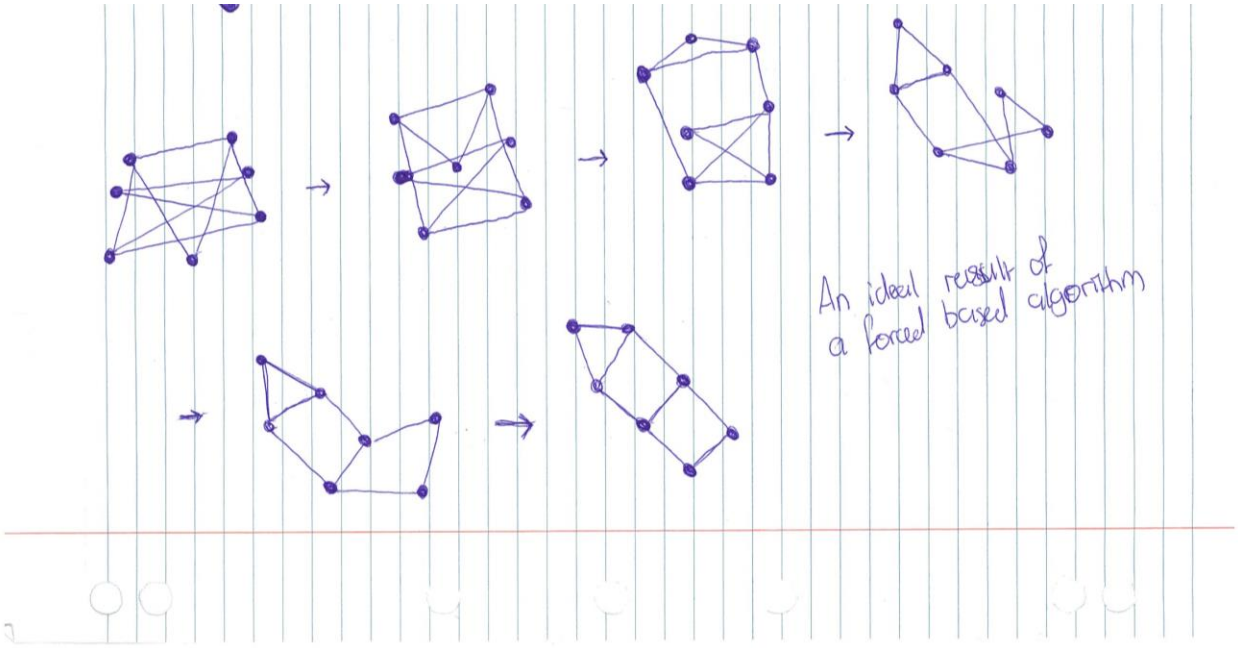
The above code will need to be repeated for a number of iterations until edge crossings are minimized. For this algorithm to be effective it must be ran multiple times with 'good' (meaning ones which minimize edge crossings) force calculation formulas to be chosen.

---

[1] http://arxiv.org/pdf/1201.3011.pdf Stephen G. Kobourov , Spring Embedders and Force Directed Graph drawing Algorithms, University of Arizona, 2012

[2] ftp://ftp.mathe2.uni-bayreuth.de/axel/papers/reingold:graph_drawing_by_force_directed_placement.pdf Thomas M. J. Fruchterman, Edward M. Reingold. Graph Drawing by Force Directed Placement, University of Illinois (1991)

Below is an image of a desirable output from this algorithm



Simple Analysis of Algorithm

There are however many issues still present in this algorithm.

- Does not tend to work well when a graph which cannot be made planar is given as input
- Hard to determine what are appropriate functions to generate force
- Has a slow running time, for each iteration the running time is O(V^2), meaning that for a large graph (large meaning one with many nodes and edges), the running time will be quite slow
- Is not guaranteed to produce desirable results
- Never guarantees a graph with no edge crossings

Although it does present advantages too

- Simple to understand as it models a real physical system (anyone with knowledge of Hooke's/Coulomb's law) can understand the forces acting on the system
- Simple to code
- Reasonable running time for small (small meaning a graph with few edges/nodes) graphs
- Tends to produce reasonable results on all test cases given, generally greater than 80% of crossings are removed even with naive functions used to generate force

The two major obstacles in implementing this function is determining an appropriate way to generate force and determining an appropriate number of iterations to run the algorithm for. This proved to be a quite difficult task (the process is further documented in the journal) but even for naive force generation formulas (i.e. 1/d^2 and delta X or delta Y) the results are quite promising, managing to remove around 85% or higher of the initial crossings.

There are other improvements that could be made to this algorithm. For example when calculating the number of iterations one could attempted to stop when the energy of the entire system has reached below a certain threshold (i.e. The nodes are no longer moving) but this may do little to improve running time for larger graphs. Another improvement may be trying to find a way to 'cut' 'long' edges in order to separate the graph into two (or more) components then individually solve these and then connect them after each component has been solved. This may provide better results but would add complexity in the sense that it may be hard to calculate what 'long' edge is.

Further improvements may require a fundamental change to the algorithm in order to position nodes in such a way that edge crossings are minimized (possible utilizing some mathematical formula). Further research shows that there may be better ways of generating a graph with minimal edge crossings by generating graph layouts which are favorable to minimizing edges.