Advanced Algorithms Project, Part 1

Nathan Stouffer Kevin Browder Seth Basseti

due: 22 October 2020

1 Problem-Statement

According to Yifan Hu "the graph layout problem is one of finding a set of coordinates, $x = \{x_i | i \in V\}$, with $x_i \in R^2$ or $x_i \in R^3$, for 2D or 3D layout, respectively, such that when the graph G is drawn with vertices placed at these coordinates, the drawing is visually appealing." [2] This problem is difficult because "visually appealing" is a subjective criteria. Different people have different interpretations of visually appealing.

There is also no universal quantitative way to measure visually appealing which makes this a difficult problem for computer to solve but there are some metrics that can help define visually appealing a little bit better. One of these is minimal edge crossing. This is used because graphs get much harder to read and understand to the human eye when there are a whole bunch of edges crossing and going to different nodes. You need to look at the visualization a lot harder before you are able to get anything from the graph. Another method is taking the forces of nature. Humans are used to looking and processing visuals produced by the laws of nature so applying the physical laws to a graph is a natural step. This can be done by simulating forces between nodes based on their position and if they are connected with an edge. This can result in a very visually appealing graph and what we will exploring in this project with force directed graphs.

2 Algorithm Description

The algorithm comes in three parts. First there is the physical modeling behind each algorithm, then there is the iterative algorithm to compute the positions, and finally there is a multilevel approach that speeds up the algorithm.

2.1 Physical Modeling

One heuristic to reach a "pretty" rendering of a graph is to introduce physical force laws to a graph and iterate until the forces no longer provide updates to the positions of the vertices. Characterized another way, we define an energy surface and move around in the state space (the positions of all the vertices) until we find a minimum configuration. If we have n vertices and we embed the graph in $\mathbb{R}^2$, then the state space is $\mathbb{R}^{2n}$ (or $\mathbb{R}^{3n}$ for graphs depicted in three space). Since n can be quite large, we are not guaranteed to find a global minimum on the energy surface, but a local minimum may suffice.

Yifan Hu covers two physically-based methods for defining an energy surface. We found the more interesting of the two to be a spring-electric model introduced by Fruchterman and Reingold [1]. There are two forces defined in this model. The first is an attracting spring force between neighbors. Given two neighboring vertices x_i, x_j, the attracting spring force is given by

$$g_s(i,j) = \|x_i - x_j\|/K$$

where K and C are tuned parameters. Note that $g_s(i,j) = 0$ for non-neighboring vertices x_i, x_j. Hu notes that K and C just scales the energy surface, hence each minima on the energy surface is also just scaled [2]. The attracting force brings neighboring vertices together, however, this is balanced by a repelling force between all vertices in the graph (called the electrical force). Given any pair of distinct vertices x_i, x_j in the graph, we can compute the repelling force as

$$g_e(i,j) = -CK^2/\|x_i - x_j\|$$

Then the total force between any pair of vertices is $f(i,j) = g_s(i,j) + g_e(i,j)$. The total force on a single vertex x_i can be computed as

$$f(i) = \sum_{i \leftrightarrow j} \frac{\|x_i - x_j\|}{K}(x_j - x_i) + \sum_{i \neq j} \frac{-CK^2}{\|x_i - x_j\|^2}(x_j - x_i)$$

Using this formula, we can construct an iterative algorithm the updates the positions of the vertices until an equilibria is reached.

2.2 Iterative Algorithm

Since we can compute the local forces on each vertex, we can initialize each vertex of the graph at a random point and the then iteratively update the positions of each vertex according to the applied force. We should note that Hu uses a momentum-esque term to escape local minimums. The algorithm computes the direction of the force as a unit vector and the magnitude is determined by the step size. Step size starts off at a relatively high value and dynamically decreases each iteration by $step_{i+1} = 0.9 * step_i$. This will help the algorithm escape local minimums.

Given two verticies and their locations, we compute the unit vector between them as $\hat{\mathbf{u}} = \frac{(x_j - x_i)}{\|x_j - x_i\|}$. Note that x_0 denotes the set of vertex coordinates.

ForceDirectedAlgorithm(G, x, tol)

```
1.   converged = FALSE
2.   step = initial step length
3.   while (converged = FALSE)
4.      x_0 = x
5.      for i ∈ V
6.         f = 0
7.         for each (i ↔ j)
8.            f = f + g_s(i,j)û
9.         for (j ≠ i, j ∈ V)
10.           f = f + g_e(i,j)û
11.        x_i = x_i + step * (f/||f||)
12.        step = 0.9*step
13.        if (||x − x_0|| < tol)
14.           converged = TRUE
14.     return x
```

Each iteration of the while loop runs in $O(|V|^2)$ time. Using the step update rule of multiplying by 0.9 will eventually force the while loop to terminate since the change in the state will decrease as the number of iterations increases. Hu also mentions a practical optimization for this algorithm. For sufficiently large distances, the combined electric forces of a number of vertices can be approximated with a "supervertex" at a center distance. The approximation is based on the physical laws used for electric forces. Hu achieves this by computing an octtree at line 4 using the initial state and using the center of each square/cube (depending on the dimension) as the center point. This reduces the run time of each iteration of the while loop to $O(|V| \log(|V|))$.

2.3 Multilevel Approach

A further optimization of this algorithm is to simplify the underlying graph. Hu calls this the Multilevel approach. Essentially, an extremely large graph G is simplified into a smaller graph G' such that G can be reconstructed from G'. Then we can run the above algorithm on G' (which will run much faster) and reconstruct G afterwards. This will not necessarily produce the same graph as running the algorithm on G but the resulting graph still looks nice (according to Hu).

3 Plus One

For our plus one component of this project, we will create a visual presentation of the force-directed graph algorithm. To do this, we will implement the algorithm presented in Hu's paper. Utilizing this algorithm and additional Python packages for visualizations, we will pick 4-5 planar graphs and display the algorithm through its iterations. We chose matplotlib as a Python plotting package to display the graphs throughout iterations. We will begin by plotting each point with matplotlib and iterate through the edges drawing lines with matplotlib to create a representation of the graph. Then, using a gif library, each graph can be saved as a frame of the animation. After saving an individual frame to the animation, we will update the vertex positions using the "adaptive cooling" algorithm shown in Hu's paper. Each animation will begin with the vertices in a random layout and display the vertices changing positions throughout each iteration of this algorithm as it optimizes the "energy" of the system. The animations will then be included in our presentation as a way of displaying, visually, how this algorithm performs. By having a method of visualizing this algorithm's process, we can use it as an aid to thoroughly explain the actual function and design of the algorithm.

References

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

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