Layout in Visual Modelling

Gitte Bluekens

University of Antwerp, Belgium

Abstract

This reports is a summary of the section 'Graph Drawing Technique Implementations' of the master thesis of Denis Dubé. [1] The first section will talk about the layered drawing technique. The second section will talk about the Spring-embedder algorithm. The third section covers the Force-transfer layout algorithm, and we end the paper with the tree-like and circle layout algorithm. These algorithms are discussed on how they work and for which models they are best used.

Keywords: Visual modeling

1. Introduction

The usefulness of visual modeling is dependent on how elements of a model are visually arranged. Hence, any tool supporting visual modeling should provide some mechanisms to reduce the burden of drawing models with good layouts. In this paper, we will discuss some automatic layer techniques implemented in the tool ATOM^3. We won't go into the implementation details, but we will discuss how these techniques work. To communicate with ATOM^3 and the layout methods, an abstraction layer was designed. This gives us the advantage that all layout algorithms can be isolated from the internal data structures of the ATOM^3 tool.

ATOM^3. ATOM^3 is an older version of the tool now known as ATOMPM. Where ATOM^3 is an acronym for "A Tool for Multi-formalism Meta-Modeling, ATOMPM is an acronym for "A Tool for Multi-Paradigm Modeling". It is used

for modeling, meta-modeling, and transforming models with graph grammars. ATOM^3 has the ability to transform a model in one formalism to another formalism using visual graph grammars. Once these grammars are created, any traffic model can be automatically transformed.

abstraction layer design. An abstraction layer was built to communicate between ATOM^3 and the automatic layout methods. This layer was constructed to reduce the implementation complexity of the automatic layout models. The functionality of the layer is to extract position information from the vertices and edges from ATOM^3 so that when the layout method is run, new position information can be sent back to the ATOM^3 entities.

2. Layered drawing technique

The layer drawing technique is excellent for visualizing graphs with a hierarchical structure. It partitions vertices into layers and finds good positions for the vertices within those layers. Edges are then drawn in between the vertices. A layer technique consists of two phases. The first phase is the layer assignment. This phase combines the greedy cycle removal algorithm with a layering algorithm. For example, D. Dubé implemented the BFS layering, longest-path layering, and minimum width layering[1]. The second phase is crossing reduction.

2.1. layer assignment

greedy cycle removal. The greedy cycle removal orders the vertices according to depth first search discovery order. The algorithm visits each vertex and if the child vertex has an order less than its parent, it will be given a flag so that they can be drawn in their original direction. The greedy cycle removal provides no performance guarantees.

BFS layering. BFS layering starts by finding the root vertices, on which the breadth first search algorithm is applied. All the discovered vertices are labeled

the next layer. This method is applied recursively until no more vertices remain. This layering technique will place vertices at their graph theoretical distances from their root vertices. BFS layering yields optimal height, but unbounded width.

longest-path layering. The longest-path layering technique starts by layering the leaf vertices at layer 1. Then vertices are added to successive layers if all their children are in layers below them. This results in the opposite of the BFS layering algorithm. Longest-path layering yields optimal height, but unbounded width.

minimum width layering. The minimum width layering is excellent at yielding drawings with good aspect ratios, but it tends to create longer edges that traverse multiple layers. It is based on the longest-path layering algorithm, but adds an additional step that connects all unconnected vertices and places them in the first or last layer.

2.2. Crossing Minimization

Layer-by-layer sweep. Layer-by-layer sweep is very important for both the quality and running time of the crossing reduction phase. We need a deterministic and converging heuristic to be able to fulfill the stopping condition of the sweep. We will be using the barycenter heuristic for this purpose. However, we will not go in more detail about heuristic algorithms. The layer-by-layer sweep stores the current best ordering of the vertices, as compared to the best number of crossings seen thus far. The algorithm halts on one of the following conditions: a hard limit on the number of rounds or iterations of the outermost loop, the algorithm has not reduced the crossings for a certain number of consecutive rounds, or, no crossings remain in the layered graph.

2.3. Analysis

It is difficult to analyze the running time of the crossing minimization phase. Crossing minimization is already a NP-hard problem with just two layers of

vertices. The problem is made even worse by the requirement of proper layering, which introduces a large number of dummy vertices where multi-layer traversing edges occur.

. The overall run-time of the layer assignment phase varies between O(|V|+|E|) and cubic time complexity depending on the layering strategy. If we make the assumption that the number of layers and the size of a layer are roughly half the number of vertices in a graph, then the overall time complexity for the layered drawing technique is quartic.

3. Spring-embedder algorithm

In the spring-embedder algorithm, edges are simulated as springs and vertices as rings to which the springs are attached to. It is fairly simple to implement. To improve the convergence speed and quality of the final drawing, a pre-processing step of circle layout or a random layout algorithm is recommended. The initialization step then consists of acquiring the center coordinates of the vertices, setting 2D force vectors to zero, and setting repulsion charges to prevent vertices from overlapping. They are set to the diagonal length of each vertex. After the initialization, the forces acting on the vertices are repeatedly calculated. This number of iterations is fixed to a default of 100 iterations. This means that the algorithm stops after 100 iterations, or if a convergence threshold is triggered. The running time of the spring-embedder algorithm is quadratic with the number of vertices.

Repulsion algorithm. The repulsion algorithm is responsible for avoiding the vertex overlaps. This is done by generating large repulsive forces whenever two vertices overlap. Initially, this algorithm calculates the Manhattan and Euclidean distances between the pair of vertices. Then, a scalar force is calculated, proportional to the charges of the vertices and inversely proportional to the square of the distance separating the vertices. This force is finally multiplied by the 2D Manhattan distance vector.

Attractive algorithm. The attractive algorithm first tries to find the Manhattan and Euclidean distances between the pair of vertices connected to a chosen edge. Next, the spring force is calculated using the physical equation for springs, since the algorithm treats edges as physical springs. Finally, the computed spring force is multiplied by the 2D Manhattan distance vector and added to the force vector of one vertex, and subtracted from the other.

Gravity algorithm. The gravity algorithm imparts upon each vertex a velocity towards the gravitational field source. This is determined to be the barycenter of all the vertices. Then, the force vector imparted on each vertex is calculated as the unit vector between the vertex and the barycenter and the strength of the gravity field. This algorithm is used to increase the area usage efficiently.

3.1. Analysis

The spring-embedder algorithm runs with a constant iteration amount. The repulsion algorithm dominates the time complexity for each simulation iteration, requiring $O(|V|^2)$ time. Since the attractive algorithm only uses $O(|E|)$ time and the gravity algorithm only uses $O(|V|)$ time, the overall time complexity for the spring-embedder algorithm is $O(|V|^2)$.

4. Force-transfer layout algorithm

The force-transfer drawing technique consists of a an initialization phase and a simulation phase. The initialization phase sets the forces acting on each vertex to zero and the position of the vertex to its center coordinate. In the simulation phase, each vertex exerts forces on overlapping neighboring vertices. This is done by calculating the Manhattan and Euclidean distances and the unit vector distance between the pair of input vertices. Then, a scalar force magnitude is computed. The direction of the force is determined by the greatest separating distance between the vertices, so the vertices are moved as little as possible. The simulation terminates once the forces have pushed all vertices apart such

that no overlap remains. The simulation can also be ended by a fixed number of iterations.

4.1. Analysis

The first loop of the force-transfer algorithm is bounded by O(|V|), the inner loop is bounded by O(50*$|V|^2$) and the last loop is bounded by O(50*|V|). This means that the overall time complexity of the force-transfer algorithm is O($|V|^2$).

5. Tree-like and circle layout algorithm

The tree-like layout algorithm gives good results on graph structures that are really trees. The circle layout algorithm is best used on subgraphs or small graphs. It makes an excellent preprocessing step for a force directed method such as spring-embedder.

5.1. Tree-like algorithm

The first step of the tree-like algorithm is to find all the root vertices in the graph. A recursive process is then started to assign the children of each root vertex coordinates before the root itself. Roots can be vertices with no parents, or the result of breaking up a cycle. The final step of the algorithm is to assign vertices with no children coordinates immediately, and vertices with children make a recursive call that assigns coordinates to the children before being assigned coordinates themselves.

5.2. Circle layout algorithm

In the circle layout algorithm, all vertices are first sorted topologically. The next step is then to calculate the perimeter of the circle. Finally, an interval fraction is calculated between 0 and 1. This interval will become the radian angle used to calculate the vertex positions on the circle.

5.3. Analysis

Since all steps of the tree-like algorithm and the circle algorithm are done in linear time, both algorithms have a linear overall run-time.

6. Conclusion

The time performance of the different algorithms allows us to compare them to each other.

. The results for the circle and tree-like layout algorithms indicate that even for large graphs, only a fraction of a second is necessary to compute the entire output. The force-transform algorithm yields quadratic asymptotic behavior, but depends on the initial layout. Even though the spring-embedder algorithm is quadratic, the time performance of this algorithm is linear. The results for the layered drawing technique indicate poor asymptotic behavior. This can be defined by the fact that the Python language was used for the implementation.

References

[1] D. Dubé, Graph layout for domain-specific modeling (2006).