Technical Report

Efficient, Proximity-Preserving Node Overlap Removal

Claire Pennarun
Tatiana Rocher

Bordeaux 1
Projet d'Etude et de Recherche

January 17, 2014

Contents

Chapter 1

Subject presentation and state of the art

A graph is a data structure encoding information with the use of nodes and edges (which are binary relations between nodes).

Graph drawing aims to represent given information as a graph, generally through a "node-link" layout, letting nodes and edges be displayed.

Most of the layout algorithms consider nodes as points, but some need to let appear additional information as labels. For example, London subway maps would be useless without the indication of the stations on the lines.

This could lead to an overlap of some nodes. That must be avoided, as it clearly confuses the understanding of the graph.

Many approachs are generally considered ; the easiest to apply is to "scale" the layout until no overlaps occur. This method has the advantage to preserve the global shape of the layout, but the area of the graph can become very inconvenient. That is why a compromise between the preservation of the shape of the graph and a minimization of the total area has to be found.

Different algorithms have been devised to answer the problem.

Our project consists in understanding the algorithm PRISM proposed by Gansner and Hu in [**?**] and in analyzing the feasability of its implementation as a plugin for the Tulip software.

Chapter 2

PRISM algorithm

2.1 Description of the algorithm

The PRISM algorithm focuses on two main constraints for the final layout of the graph. First, the area taken by the layout must be minimal. The second constraint is to preserve the global "shape" of the original layout by maintaining all proximity relations between the nodes.

The PRISM algorithm runs in two main steps ; in a first step, it removes iteratively the overlaps between near nodes of the given graph G. Then it finds the non-near overlaping nodes and removes these overlaps as well.

We consider for this algorithm that a node i has a certain width w_i and height h_i, thus forming a rectangle containing the label, likely to cause overlaps.

2.1.1 Overlap removal between near nodes

Use of a proximity graph - Delaunay Triangulation

To find easily the overlaps between near nodes of the graph G, it will efficient to work on a proximity graph of G. Such a graph will also guarantee the preservation of the proximity relations during the different stages of the algorithm.

A *proximity graph* is a graph in which two vertices are connected by an edge if (and only if) they satisfy a given geometrical property.

The *Delaunay triangulation* (DT) of a graph G is a triangulation of the

graph such that none of the circumscribed circles of the triangles in DT(G) contains a vertex. This particular triangulation also maximises the minimum angle of the triangles found.

The Delaunay triangulation of G, as a triangulation, is also a planar graph, and has thus at most $3n - 6$ edges (if $|V(G)| = n$), which is a very practical parameter for the algorithm.

The nearest neighbors of a vertex $v \in V(G)$ tend to form triangles with v and in particular, the closest neighbor of v has an edge with v in DT(G), as the nearest neighbor graph of G is a subgraph of DT(G).

In the PRISM algorithm, we consider that the near nodes in G are connected by an edge in the Delaunay triangulation of G.

Thus, the algorithm's first goal is to remove overlaps along the edges of the Delaunay triangulation of G.

Ideal edge length

The idea is to find the "ideal length" of the Delaunay triangulation edges : the "ideal length" of an edge is such that the two edge ends have no overlap.

In order to do that, we calculate an *overlap factor* f_{ij} for each edge (i, j) $(i, j \in V(G))$ of the Delaunay triangulation of G :

$$f_{ij} = \max(\min(\frac{w_i/2 + w_j/2}{x_i - x_j}, \frac{h_i/2 + h_j/2}{y_i - y_j}), 1)$$

where (x_i, y_i) are the coordinates of vertex i, w_i its width and h_i its height.

If two nodes i and j have no overlap, then $f_{ij} = 1$. If i and j do overlap, then that overlap can be removed by expanding the edge (i, j) by the overlap factor found f_{ij}.

Thus, the "ideal length" of an edge of the Delaunay triangulation is $l_{ij} = f_{ij}||p_i - p_j||$, where p_i is the initial set of coordinates of a node i.

We now want to find coordinates for the nodes of the initial graph such that the edges length in $DT(G)$ are close to their ideal length.

Proximity stress model

Finding this new set of coordinates means minimizing the following sum :

$$\sum_{i,j \in E(DT(G))} w_{ij}(||p_i - p_j|| - l_{ij})^2$$

where l_{ij} is the overlap factor and w_{ij} is a classic weighting factor, used to equalize the contributions to the total sum from the different edges.

But there are some situations where keeping l_{ij} is not a good idea (see figure)

We thus want to avoid removing the "big" overlaps in one iteration only. So we have to replace l_{ij} by $\min(l_{ij}, s)$, where $s > 1$ will be a limiting factor. The authors found that $s = 1.5$ worked well.

We now want to minimize :

$$\sum_{i,j \in E(DT(G))} w_{ij}(||p_i - p_j|| - \min(l_{ij}, s))^2$$

This type of sum is called a "stress function", in analogy with the well studied *stress model*. The above sum is called the *proximity stress model*.

The minimization of the sum gives new positions for the nodes of G.

Iteration and termination

This first phase provides us with a new layout of the graph G, in which the nodes positions are given according to the previous minimization of the proximity stress model.

This layout may still contain overlaps. We must thus iterate the construction of the Delaunay triangulation, the computation of the overlap factors, the minimization of the proximity stress model and the move of the nodes until no more overlaps occur along the edges of the Delaunay triangulation of the graph.

The process of the first stage makes clear that no overlaps can appear : the distance between nodes can only be increasing.

But the authors do not explain formally the reason of the termination of this phase : the stress function could always be smaller but never reach a local minimum.

The author's implementation of PRISM contains a threshold : if the gain in the minimization of the stress function is smaller, the first phase of the algorithm ends.

2.1.2 Overlap removal between non-near nodes

Why do we need a second stage ?

Scan-line algorithm

Algorithm 1: PRISM

Input: p_i^0 : coordinates of each vertex
width w_i and height h_i of each vertex ($i = 1, 2, ..., |V|$)

repeat
  G_{DT} : proximity graph of G by Delaunay triangulation
  for *all edges of* G_{DT} **do**
    Compute the overlap factor
  $\{p_i\}$: solution of the proximity stress model
  $p_i^0 = p_i$
until no more overlaps along edges of G_{DT};
repeat
  G_{DT} : proximity graph of G by Delaunay triangulation
  Find overlaps in G through a scan-line algorithm
  Add the overlapping edges to G_{DT}
  for *all edges of* G_{DT} **do**
    Compute the overlap factor
  $\{p_i\}$: solution of the proximity stress model
  $p_i^0 = p_i$
until no more overlaps found by the scan-line algorithm;

2.1.3 Dissimilarity metrics

Area

Edge length ratio

Vertices displacement

2.2 Complexity

Chapter 3

Implementation within Tulip

3.1 Tulip framework

Tulip presentation
- Tulip node structure (problems with label size)
- Solution : forcing node size and labels

3.2 Resolution of the stress model

Stress majorization

Not possible natively in Tulip
- Too many dependencies in GraphViz
- Use of Eigen possible ?

Newton-Raphson method - Kamada & Kawai

Implementation done ?

3.3 Scan-line algorithm

Implementation details

Chapter 4

Tests and results

4.1 Use of GraphViz

Chapter 5

Conclusion