Université de Bordeaux

Projet d'Etude et de Recherche

Efficient, Proximity-Preserving Node Overlap Removal

Claire Pennarun
Tatiana Rocher

Article by E.R. Gansner and Y. Hu (2009)

January 21, 2014

Contents

Abstract

Abstract..

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 a given information as a graph, generally through a "node-link" layout, letting only 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.

Moreover, as we generally consider that the original layout contains significant information, an other parameter to deal with is to maintain the "global shape" of the initial representation.

This "global shape" can be seen as "preserving the proximity relations between nodes", "preserving the orthogonal ordering of nodes"(see [18]) or "preserving the relative positions of nodes by limiting the vertices displacement"(see [8]), and a choice between these criteria has to be made.

The easiest approach 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, each of

them focusing on a different "global shape" definition.

The first approach is to try to avoid overlaps while generating the layout.

The spring-electrical model presented by Eades [4] and Fruchterman and Reingold [6] considers the edges as springs between nodes, so that the spring forces move the nodes to a minimal energy state of the global system. A repelling force between non-adjacent nodes is added. This model has been adapted by various authors ([10], [17]) to take the node size into account, generally as increased repulsive forces.

The stress model of Kamada and Kawai [13] is based on the assumption that a graph layout is "good" if the distance between two vertices is close to the theoretical graph distance between these vertices, i.e. to the length of their shortest path. It can also be extended to avoid as much as possible overlap along the edges.

These two models (spring-electrical model and stress model) try to avoid all overlaps, but use generally a post-processing algorithm to ensure the total overlap removal.

More details about the force-directed drawing algorithms can be found in [14].

The second possibility is to remove overlaps after the graph is drawn : these are post-processing algorithms.

The Voronoi cluster busting algorithm [8] restrains the possible displacement of a node with the use of Voronoi cells. This restriction aims to help preserving the relative positions of the nodes. In practice, the algorithm often requires a lot of iterations and the global similarity with the initial layout can be low (see for example the figure 1.1).

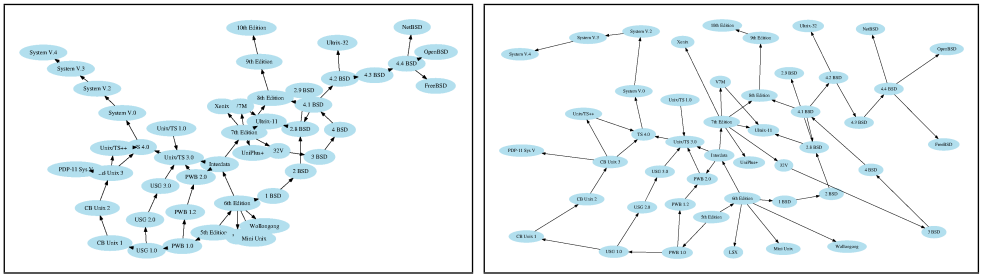

Figure 1.1: (left) The initial layout. (right) The result of the Voronoi-based algorithm

The Satisfy_VPSC algorithm [3], solving the "variable placements with separation constraints" problem, moves iteratively the nodes in the horizontal and in the vertical dimensions. This algorithm aims to minimize the vertices displacement but can generate layouts that are very dissimilar to the initial layout (see figure 1.2).

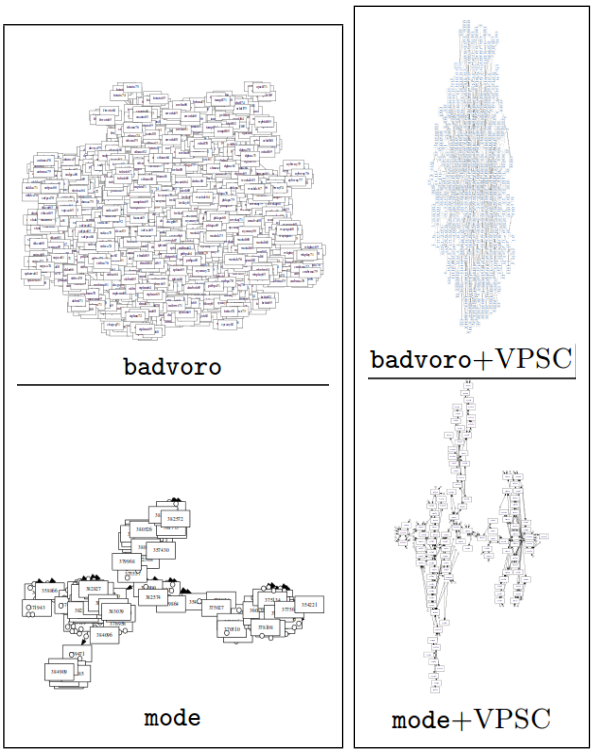

Figure 1.2: (left) The initial layouts. (right) The results of the Solve_VPSC algorithm

Some work on word cloud generators like Wordle (`http://www.wordle.net/`) allowed to develop new algorithms like Mani-Wordle [15] and RWordle [19] based on a spiral scheme for the random placement of text labels in order to overcome overlaps.

Our project consists in understanding the algorithm PRISM proposed by Gansner and Hu in [7] and in analyzing the feasability of its implementation as a plugin for the Tulip software [1].

Chapter 2

PRISM algorithm

2.1 Formal notations

We use the following notations : $G = (V, E)$ denotes the current graph, with V the set of vertices (or nodes), and E the set of edges. The number of vertices and of edges are denoted respectively $|V|$ (or n) and $|E|$.

An edge between two vertices i and j is denoted (i, j).

The position of a node i in a layout is represented as a set of 2D coordinates $p_i = (x_i, y_i)$. The initial layout position of i is denoted as $p_i^0 = (x_i^0, y_i^0)$.

We consider for the PRISM 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.

2.2 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 overlapping nodes and removes these overlaps as well.

2.2.1 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 (a survey on proximity graphs can be found in [12]).

The *Delaunay triangulation* (DT) (named after the work of Delaunay [2]) 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 on the inside. This particular triangulation also maximises the minimum angle of the triangles found.

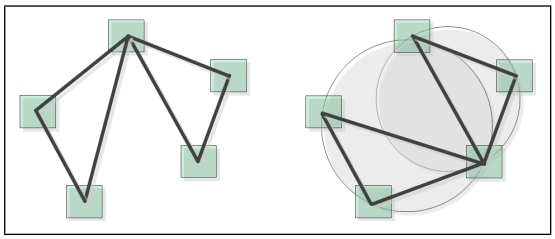

Figure 2.1: (left) A graph G. (right) The Delaunay triangulation of G

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) (see [12]).

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 :

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

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} (see figure 2.2).

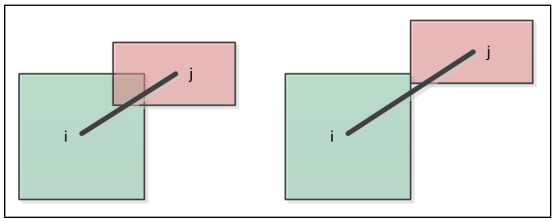

Figure 2.2: Overlap factor

Thus, the "ideal length" of an edge of the Delaunay triangulation is $l_{ij} = f_{ij}||p_i^0 - p_j^0||$, where p_i^0 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 :

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

where l_{ij} is the overlap factor and $w_{ij} = 1/(l_{ij})^2$ is a classic weighting factor, used to equalize the contributions to the total sum from the different edges.

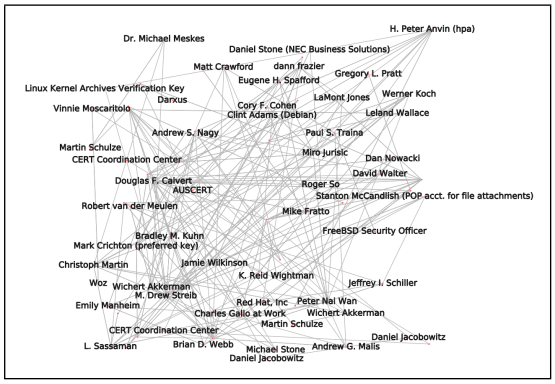

Figure 2.3: A bad case

But there are some situations where keeping l_{ij} is not a good idea (see figure 2.3).

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

This type of sum is called a "stress function", in analogy with the well studied *stress model* introduced by Kruskal in [16] and applied to graph drawing by Kamada and Kawai in [13]. 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 and the proximity graph on which we are calculating, as a triangulation, is a rigid graph.

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. Moreover, the number of iterations needed during the first phase is not explicited nor bounded in the article.

The authors' 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.

2.2.2 Overlap removal between non-near nodes

The first step removes the overlap between ends of edges of the Delaunay triangulation of the graph. But some overlaps can be caused by nodes not being near, and thus not generating an edge in the proximity graph. These overlaps can not have been removed by the first stage of the PRISM algorithm.

To find these still overlapping nodes, we have to use a scan-line algorithm.

Scan-line algorithm

A scan-line algorithm is an algorithm which will consider all the points of a layout.

We can not use a algorithm which uses the graph properties because the vertices do not know the positions of others vertices, so we can not find the overlaps by using the vertices properties.

For all the ordinate's points we consider all the points in the abscissa. If there is an overlap at a point, we add the edge (between the two vertices overlapping) in the Delaunay triangulation.

It is interesting to note that one of the opponent algorithms, Satisfy_VPSC [3], mainly uses a scan-line algorithm to remove overlaps.

Overlap removal

The second stage uses the same processus as the first one, only adding the overlapping edges found by the scan-line algorithm to the Delaunay triangulation before the calculation of the overlap factors and the resolution of the proximity stress model.

This stage ends when no more overlaps are found by the 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;

Figure 2.4: The PRISM algorithm

2.2.3 Dissimilarity metrics

To be able to compare different graph layouts, the authors propose three dissimilarity metrics : the area taken by the layout, a metric based on the edge length ratio in the proximity graphs of the initial and final layouts, and a metric measuring the vertices displacement between the initial and the final layouts.

Area

As said before, it is easy to remove all the overlaps by extending all the edges of the initial layout. But the final layout can be extremely large and thus unreadable, so we want to keep an area as small as possible.

Edge length ratio

We first calculate the ratio between the edge lengths of the proximity graphs of the original layout and the final one. The metric is then defined as the normalized standard deviation to the mean ratio found.

This metric has to be as small as possible to minimize the changes made to the edges length during the algorithm.

The edge length ratio has to be calculated on a rigid graph (as the Delaunay triangulations) to be meaningful : two layouts of the same graph can be completely different if the graph is not rigid. (exemple ?)

Vertices displacement

The third measure of dissimilarity is the calculation of the displacement of vertices between the original layout and the final one. We consider that layouts resulting from a scaling, a shift or a rotation are identical. Thus, computing the displacement of vertices means finding the optimal scaling, shift and rotation minimizing the displacement. (formule ?)

2.3 Complexity

→ comparaison avec VPSC

Chapter 3

Implementation within Tulip

3.1 The Tulip framework

The Tulip framework [1], developed mainly by the Data Visualization team of the LaBRI (Bordeaux), is a data visualization software dedicated to the analysis and visualization of relational data. It focuses on the manipulation of graphs. Our goal was to implement the PRISM algorithm as a plugin for Tulip, reusing some of the tools already present in the software, such as the calculation of the Delaunay triangulation of a graph.

Tulip uses particular structures to deal with graph manipulations, and one of these structure is very important for the PRISM algorithm : the `node` structure.

The PRISM algorithm considers that the label is part of the node, and that the size of the node is determined by the size of the underlying label.

But in Tulip, the labels are seen as a "property" of the graph, and are decorrelated from the nodes themselves. They generally are dynamic sized, and displayed in order to avoid overlaps. We thus do not have access to the size of labels.

The solution was to increase the size of the nodes and to force the labels inside them (see figure 3.1). Thus, we could considerate that the "labels" were overlapping, even if it was artificially.

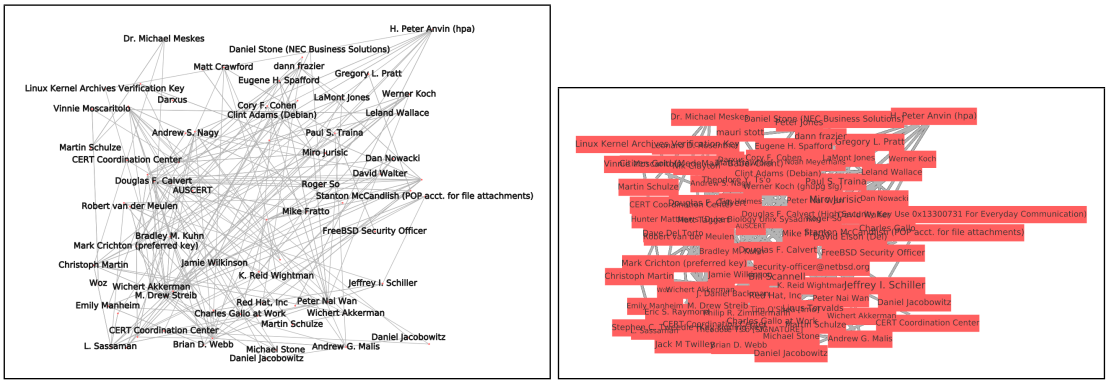

Figure 3.1: Adjustment of the nodes size in Tulip

3.2 Resolution of the stress model

The resolution of the proximity stress model can be done in many different ways. The authors, through their implementation in GraphViz [5], a set of graph drawing tools, chose to solve it via a stress majorization technique (described in details in [9]).

Stress majorization

We first considered to implement the resolution of the proximity stress model in the same way as the authors by employing a stress majorization technique.

This procedure consists in bounding the function we want to optimize by a series of quadratic functions from above. Solving the stress model means then finding an optimum, which is done by solving a series of linear systems.

This resolution implies the manipulation of weighted Laplacian of matrices and the use of a conjugate gradient algorithm with preconditions. These features are not present in the Tulip framework and were too complex to implement in the given time for the project.

Another possibility was to retrieve some implementation from the GraphViz suite and use it as an external plugin or module. But an analysis of the dependencies of the requested function (the `StressMajorizationSmoother_smooth()` function in the `post_process.c` file) were to important to do so.

We used a python script to retrieve the number of lines of the dependance files of `post_process.c`. The script is given as an annex (mettre ref). We

used the `-MM` option of `gcc` to generate the headers (.h files) necessary to the execution of the file, then searched for the .c files having the same name and count the number of lines. As a result, 16 of the 52 needed headers have a corresponding .c having the same name, and these 16 files have a total of 14033 lines. This was considered as a too large amount of code to use as an external plugin or module (and these are only the first level dependancies !).

We then focused on finding an other resolution method for the proximity stress model and we decided to adapt an algorithm based on a Newton-Raphson method.

Newton-Raphson method - Kamada & Kawai

Kamada and Kawai [13] proposed an algorithm computing a local minimum of the energy E of a system. We adapted and implemented it by using

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

The purpose is to calculate new positions $p_i = (x_i, y_i)$ for $i = 1...n$, locally minimizing the global stress function E.

We thus want to find x_i, y_i such that $\delta E/\delta x_i = \delta E/\delta y_i = 0$ for $i = 1...n$. That means solving simultaneously $2n$ (non-linear) equations.

The Kamada and Kawai algorithm consists in moving only one node i at a time to reach $\delta E/\delta x_i = \delta E/\delta y_i = 0$, "freezing" the other nodes during the computation.

We adapted the computation of $\delta E/\delta x_i$ and $\delta E/\delta y_i$ to our configuration : we considered the following values :

$$\begin{aligned} \frac{\delta^2 E}{\delta x_i} &= \sum_{(i,m) \in E(DT)} w_{im}((x_m - x_i) - \frac{l_{im}(x_m - x_i)}{||p_m - p_i||} \\ \frac{\delta^2 E}{\delta y_i} &= \sum_{(i,m) \in E(DT)} w_{im}((y_m - y_i) - \frac{l_{im}(y_m - y_i)}{||p_m - p_i||} \end{aligned}$$

where DT is the Delaunay triangulation of the graph, l_{im} is the ideal length of the (i, m) edge, $w_{im} = 1/(l_{im})^2$ is a weighting factor, and $||p_m - p_i||$ is the euclidian distance between the nodes m and i.

At each iteration, the node moved is the one having the largest value of Δ_i (i.e. the "further" one of the condition to reach):

$$\Delta_i = \sqrt{\left(\frac{\delta E}{\delta x_i}\right)^2 + \left(\frac{\delta E}{\delta y_i}\right)^2}$$

We thus only have to solve a two-equations system at each step (the two unknown values are δx and δy):

$$\begin{aligned} \frac{\delta^2 E}{\delta x_i^2}(p_i) \times \delta x + \frac{\delta^2 E}{\delta x_i \delta y_i}(p_i) \times \delta y &= -\frac{\delta E}{\delta x_i}(p_i) \\ \frac{\delta^2 E}{\delta y_i \delta x_i}(p_i) \times \delta x + \frac{\delta^2 E}{\delta y_i^2}(p_i) \times \delta y &= -\frac{\delta E}{\delta y_i}(p_i) \end{aligned}$$

The $(\delta x, \delta y)$ found are added to the current position of the node i, and this is done iteratively until $\Delta_i < \varepsilon$, where ε is a threshold determined by the user.

The algorithm stops when all considered nodes have reached $\Delta_i < \varepsilon$. As $\varepsilon \neq 0$, it happens that all nodes have not been removed.

3.3 Scan-line algorithm

Implementation details

3.4 Tests and results

Tests and time of the Kamada and Kawai solution

Chapter 4

Conclusion

Since PRISM :
    ePRISM : PRISM on overlap of edges [11]

Appendix A

Python script

```
#! bin/python
import os
import os.path
s = os.popen("gcc -MM post_process.c -I ../common/ -I ../gvc/ -I
../pathplan/ -I ../cdt/ -I ../sparse/ -I ../neatogen/ -I ../sfdpgen/
-I ../cgraph/").read()
s = s.split(' ',2)[2]
s = s.replace(".h",".c")
totalLineAmount = 0
nbNotFoundFiles = 0
nbFoundFiles = 0
for f in s.split(' '):
    if os.path.isfile(f):
        cm = os.popen("wc -l " + f).read()
        cm = cm.split(' ')[0]
        totalLineAmount += int(cm)
nbFoundFiles += 1
    else:
        nbNotFoundFiles += 1
print s, totalLineAmount, nbNotFoundFiles, nbFoundFiles
```

Bibliography

[1] D. Auber, D. Archambault, R. Bourqui, A. Lambert, M. Mathiaut, P. Mary, M. Delest, J. Dubois, and G. Melançon. The Tulip 3 Framework: A Scalable Software Library for Information Visualization Applications Based on Relational Data. Rapport de recherche RR-7860, INRIA, January 2012.

[2] Boris N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, (6):793–800, 1934.

[3] Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast Node Overlap Removal. In *GD2005: Proceedings of the 13th International Symposium of Graph Drawing 2005*, volume 3843 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2006.

[4] P. A. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[5] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, and Gordon Woodhull. Graphviz— Open Source Graph Drawing Tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, chapter 57, pages 594–597. Springer Berlin / Heidelberg, Berlin, Heidelberg, February 2002.

[6] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[7] Emden R. Gansner and Yifan Hu. Efficient node overlap removal using a proximity stress model. In *Graph Drawing*, pages 206–217, 2008.

[8] Emden R. Gansner and Stephen C. North. Improved Force-Directed Layouts. In *GD 1998: Proceedings of the 6th International Symposium of Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 364–373, Heidelberg, 1998. Springer.

[9] EmdenR Gansner, Yehuda Koren, and Stephen North. Graph Drawing by Stress Majorization. In János Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 239–250. Springer Berlin Heidelberg, 2005.

[10] David Harel and Yehuda Koren. *A Fast Multi-scale Method for Drawing Large Graphs*, volume 1984. January 2001.

[11] Yifan Hu. Visualizing graphs with node and edge labels. *CoRR*, abs/0911.0626, 2009.

[12] Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. In *Proc. IEEE*, pages 1502–1517, 1992.

[13] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[14] S. G. Kobourov. *Handbook of Graph Drawing and Visualization*, chapter Force-Directed Drawing Algorithms, pages p. 383–408. CRC Press, 2013.

[15] Kyle Koh, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. ManiWordle: Providing Flexible Control over Wordle. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1190–1197, November 2010.

[16] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, March 1964.

[17] Wanchun Li, Peter Eades, and Nikola Nikolov. Using spring algorithms to remove node overlapping. In *APVis 2005: Proceedings of the 2005 Asia-Pacific symposium on Information visualisation*, pages 131–140, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[18] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[19] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen. Rolled-out wordles: A heuristic method for overlap removal of 2d data representatives. *Computer Graphics Forum*, 31(3pt3):1135–1144, 2012.