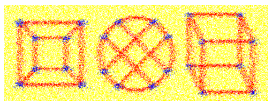
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 14, no. 1, pp. 53–74 (2010)

Efficient, Proximity-Preserving Node Overlap Removal

Emden R. Gansner Yifan Hu

AT&T Labs,
Shannon Laboratory,
180 Park Ave.,
Florham Park, NJ 07932.

Abstract

When drawing graphs whose nodes contain text or graphics, the non-trivial node sizes must be taken into account, either as part of the initial layout or as a post-processing step. The core problem in avoiding or removing overlaps is to retain the structural information inherent in a layout while minimizing the additional area required. This paper presents a new node overlap removal algorithm that does well at retaining a graph's shape while using little additional area and time. As part of the analysis, we consider and evaluate two measures of dissimilarity for two layouts of the same graph.

Submitted:	Reviewed:	Revised:	Accepted:
November 2008	March 2009	May 2009	November 2009
	Final:	Published:	
	November 2009	January 2010	
Article type:		Communicated by:	
Regular paper		I. G. Tollis and M. Patrignani	

E-mail addresses: erg@research.att.com (Emden R. Gansner) yifanhu@research.att.com (Yifan Hu)

1 Introduction

Most existing symmetric graph layout algorithms treat nodes as points. In practice, nodes usually contain labels or graphics that need to be displayed. Naively incorporating this can lead to nodes that overlap, causing information of one node to occlude that of others. If we assume that the original layout conveys significant aggregate information such as clusters, the goal of any layout that avoids overlaps should be to retain the "shape" of the layout based on point nodes.

The simplest and, in some sense, the best solution is to scale up the drawing while preserving the node size until the nodes no longer overlap [30]. This has the advantage of preserving the shape of the layout exactly, but can lead to inconveniently large drawings. In general, overlap removal is typically a trade-off between preserving the shape and limiting the area, with scaling at one extreme.

Many techniques to avoid overlapping nodes have been devised. One approach is to make the node size part of the model of the layout algorithm. It is assumed that whatever structure that would have been exposed using point nodes will still be evident in these more general layouts. For hierarchical layouts, the node size can be naturally incorporated into the algorithm [11, 35]. For symmetric layouts, various authors [3, 19, 28, 37] have extended the spring-electrical model [7, 12] to take into account node sizes, usually as increased repulsive forces. Node overlap removal can also be built into the stress model [26] by specifying the ideal edge length to avoid overlap along the graph edges. Such heuristics, however, cannot guarantee all overlaps will be removed, so they typically rely on overly large repulsive forces, or the type of post-processing step considered below. This problem was addressed by Dwyer et al. [5], who showed how to encode overlap avoidance, as well as many other layout features, as linear constraints in a stress function, and then optimize the stress function using stress majorization [13].

An alternative approach is to remove node overlaps as a post-processing step after the graph is laid out. Here the trade-off between layout size and preserving the graph's shape is more explicit. In addition to many of the algorithms mentioned above that can be adapted for this use, a number of other algorithms have been proposed. For example, the Voronoi cluster busting algorithm [15, 29] works by iteratively forming a Voronoi diagram from the current layout and moving each node to the center of its Voronoi cell until no overlaps remain. The idea is that restricting each node to its corresponding Voronoi cell should preserve the relative positions of the nodes. In practice, because of the number of iterations often required and the use of a rectangular bounding box,[1] the nodes in the final drawing can be homogeneously distributed within a rectangle, the graph bearing little resemblance to the original layout.

Another group of post-processing algorithms is based on setting up pairwise node constraints to remove overlaps and then performing some procedure to

[1] The latter might be improved by using something like α-shapes [8] to provide a more accurate boundary.

generate a feasible solution. The force scan algorithm [31] and its later variants [20, 24] proceed along these lines. More recently, Marriott et al. [30, 31] have presented quadratic programming algorithms which remove node overlaps while, if desired, minimizing node displacement. All of these techniques rely on solving separate horizontal and vertical problems. They behave differently if the layout is rotated by a degree that is not a multiple of $\pi/2$. We have also found that, in practice, this asymmetry often results in a layout with a distorted aspect ratio (e.g., Figure 4, bottom right).

One desideratum proposed [31] for overlap removal is the preservation of *orthogonal ordering*, i.e., the relative ordering of the x and y coordinates of two nodes should be the same in the original layout and the one with overlaps removed. Thus, if a node is above and to the left of another node in the original layout, it is above and to the left in the derived one. The idea is that, in this manner, a user's "mental map" of the graph is better preserved. Many of the algorithms described above either inherently preserve orthogonal ordering, or have simple variations that do. While preserving orthogonal ordering can be important, it alone cannot ensure that the relative proximity relations [23] between nodes are preserved. At other times, it is too restrictive, causing two highly-related nodes to be moved far apart due to some largely unrelated third node. For these reasons, our approach is to concentrate on proximity relations and not deal with orthogonal ordering.

In this paper, we focus on the problem of removing overlaps rather than avoiding them. To this end, we discuss (Section 3) metrics for the similarity between two layouts which we believe better quantify the desired outcome of overlap removal than minimized displacement or such simpler measures as aspect ratio or edge ratio (the ratio of the longest and the shortest edge lengths). We then present (Section 4) a node overlap removal algorithm based on a proximity graph of the nodes in the original layout. Using this graph as a guide, it iteratively moves the nodes, particularly those that overlap, while keeping the relative positions between them as close to those in the original layout as possible. The algorithm is similar to the stress model [26] used for graph layout, except that the stress function involves only a sparse selection of all possible node pairs. Because of this sparse stress function, the algorithm is efficient and is able to handle very large graphs. In Section 5, we evaluate our algorithm and others using the proposed similarity measures. Finally, Section 6 presents a summary and topics for further study.

2 Background

We use $G = (V, E)$ to denote an undirected graph, with V the set of nodes (vertices) and E edges. We use $|V|$ and $|E|$ for the number of vertices and edges, respectively. We let x_i represent the current coordinates of vertex i in Euclidean space. For this paper we are interested in 2D layout, therefore $x_i \in R^2$.

The aim of graph drawing is to find x_i for all $i \in V$ so that the resulting

drawing gives a good visual representation of the information in the graph. Two popular methods, the spring-electrical model [7, 12], and the stress model [26], both convert the problem of finding an optimal layout to that of finding a minimal energy configuration of a physical system. We shall describe the stress model in more detail since we will use a similar model for the purpose of node overlap removal in Section 4.

The stress model assumes that there are springs connecting all nodes of the graph, with the ideal spring length equal to the graph theoretical distance between nodes. The energy of this spring system is

$$\sum_{i \neq j} w_{\mathrm{ij}} \left(\|x_i - x_j\| - d_{\mathrm{ij}} \right)^2, \tag{1}$$

where d_{ij} is the graph theoretical distance between vertices i and j, and w_{ij} is a weight factor, typically $1/{d_{\mathrm{ij}}}^2$. The layout that minimizes the above stress energy is an optimal layout of the graph. There are several ways to find a solution of the minimization problem. An iterative approach can be employed. Starting from a random layout, the total spring force on each vertex is calculated, and the vertex is moved along the direction of the force for a certain step length. This process is repeated, with the step length decreasing every iteration, until the layout stabilizes. Alternatively, a stress majorization technique can be employed, where the cost function (1) is bounded by a series of quadratic functions from above, and the process of finding an optimum becomes that of solving a series of linear systems [13].

In the stress model, the graph theoretical distance between all pairs of vertices has to be calculated, leading to quadratic complexity in the number of vertices. There have been attempts (e.g., [2, 13]) to simplify the stress function by considering only a sparse portion of the graph. Our experience, however, with real-life graphs is that these techniques may fail to yield good layouts. Therefore, algorithms based on a spring-electrical model employing a multilevel approach and an efficient approximation scheme for long range repulsive forces [18, 22, 36] are still the most efficient choices to lay out large graphs without consideration of the node size.

3 Measuring Layout Similarity

The outcome of an overlap removal algorithm should be measured in two aspects. The first aspect is the overall bounding box area: we want to minimize the area taken by the drawing after overlap removal. The second aspect is the change in relative positions. Here we want the new drawing to be as "close" to the original as possible. It is this aspect that is hard to quantify.

When total node movement is optimized [6, 24], comparison is reduced to measuring the amount of displacement of the vertices in the new layout from those of the original graph. This measurement does not take into account possible shifts, scalings or rotations, nor the importance of maintaining the relative position among vertices.

As far as we are aware, there is no definitive way to measure similarity of two layouts of the same graph. We adopt two approaches. The first approach is based on measuring changes in lengths of edges. The second approach, which is a modification of the metric of Dwyer et al. [6], is based on measuring the displacement of vertices, after discounting shift, scaling and rotation.

Before defining these two measures of similarity, we first introduce the concept of a *proximity graph*. A proximity graph is a graph derived from a set of points in space: points that are "neighbors" to each other in the space form an edge in the proximity graph. There are many ways to create a proximity graph [25]. In this paper, we shall work with the Delaunay triangulation (DT), which is an approximation to the proximity graph, and has the advantage being rigid. Two points are neighbors in DT if and only if there exists a sphere passing through these two points, and no other points lie in the interior of this sphere.

One way to measure the similarity of two layouts is to measure the distance between all pairs of vertices in the original and the new layout. If the two layouts are similar, then these distances should match, subject to scaling. This is known as Frobenius metric in the sensor localization problem [9]. Calculating all pairwise distances is expensive for large graphs, both in CPU time and in the amount of memory. As we want a metric feasible for very large graphs, we instead form a DT of the original graph, then measure the distance between vertices along the edges of the triangulation for the original and new layouts.[2] If x^0 and x denote the original and the new layout, and E_P is the set of edges in the triangulation, we calculate the ratio of the edge length

$$r_{\mathrm{ij}} = \frac{\|x_i - x_j\|}{\|x_i^0 - x_j^0\|},\ \{i,j\} \in E_P,$$

then define a measure of the dissimilarity as the normalized standard deviation

$$\sigma_{\mathrm{dist}}(x^0, x) = \frac{\sqrt{\frac{\sum_{\{i,j\}\in E_P}(r_{\mathrm{ij}} - \bar{r})^2}{|E_P|}}}{\bar{r}},$$

where

$$\bar{r} = \frac{1}{|E_P|} \sum_{\{i,j\}\in E_P} r_{\mathrm{ij}}$$

is the mean ratio.

The reason we measure the edge length ratio along edges of the proximity graph, rather than along edges of the original graph, is that if the original graph is not rigid, then even if two layouts of the same graph have the same edge lengths, they could be completely different. For example, think of the graph of a square, and a new layout of the same graph in the shape of a non-square rhombus. These two layouts may have exactly the same edge lengths,

[2] Another possibility would be to sample $O(|V|)$ out of the $|V|(|V|-1)/2$ possible vertex pairs. Some care is needed in making sure that the resulting graph is rigid.

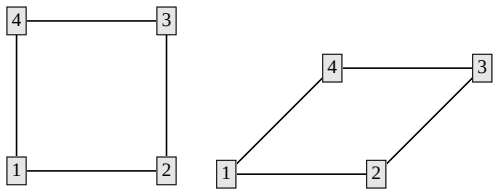

Figure 1: The edge lengths of these two layouts of a non-rigid graph are exactly the same, but the layouts are clearly different.

but are clearly different (see Figure 1). The rigidity of the triangulation avoids this problem.

Notice that $\sigma_{\text{dist}}(x^0, x)$ is not symmetric with regard to which layout comes first. Furthermore, in theory, this non-symmetric version could class a layout and a foldover of it (e.g., a square grid with one half folded over the other) as the same. We can symmetrize it by defining the dissimilarity between layout x and x^0 as $(\sigma_{\text{dist}}(x^0, x) + \sigma_{\text{dist}}(x, x^0))/2$. This also resolves the "foldover problem". The symmetric version may be more appropriate if we are comparing two unrelated layouts. Since, however, we are comparing a layout derived from an existing layout, we feel that the asymmetric version is adequate.

An alternative measure of similarity is to calculate the displacement of vertices of the new layout from the original layout [6]. Clearly a new layout derived from a shift, scaling and rotation should be considered identical. Therefore we modify the straight displacement calculation by discounting the aforementioned transformations. This is achieved by finding the optimal scaling, shift and rotation that minimize the displacement. The optimal displacement is then a measure of dissimilarity.

We denote by scalars r and θ the scaling and rotation,

$$T = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2}$$

the rotation matrix, $p \in \mathbb{R}^2$ the translation, and define the displacement dissimilarity as

$$\sigma_{\text{disp}}(x^0, x) = \min_{p \in \mathbb{R}^2, \theta, r \in \mathbb{R}} \sum_{i \in V} \|rTx_i + p - x_i^0\|^2, \tag{3}$$

This is a known problem in the Procrustes analysis [1, 16] and the solution (the Procrustes statistic) is

$$\sigma_{\text{disp}}(x^0, x) = \text{tr}(X^0 X^{0^T}) - (\text{tr}((X^T X^0 X^{0^T} X)^{\frac{1}{2}})^2 \text{tr}(X^T X), \tag{4}$$

where $\text{tr}(A)$ is the trace of a matrix A, X is a matrix with columns $x_i - \bar{x}$, X^0 is a matrix with columns $x_i^0 - \bar{x}^0$, and $\bar{x}$ and $\bar{x}^0$ are the centers of gravity of the new and original layout.

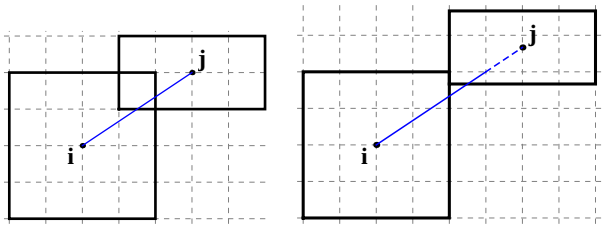

Figure 2: Nodes i and j overlap (left). The overlap factor, according to (5), is $min((2+2)/3, (1+2)/2) = 1.33$. Expanding the edge $i-j$ by 33% removes the overlap (right).

In the above we do not consider shearing, since we believe a layout derived from shearing of the original should not be considered identical to the latter.

The quality of an overlap removal algorithm is a combination of how similar the new layout is to the original, and how small an area it occupies. The simplest overlap removal algorithm is that of scaling the layout until all overlaps are removed. This has a dissimilarity of 0, but usually occupies a very large area. The alternative extreme is to pack the nodes as close to each other as possible while ignoring the original layout. This will have the smallest area, but a large dissimilarity. A good solution should be a compromise between these two extremes. We now describe one candidate.

4 A Proximity Stress Model for Node Overlap Removal

Our goal is to remove overlaps while preserving the shape of the initial layout by maintaining the proximity relations [23] among the nodes. To do this, we first set up a rigid "scaffolding" structure so that while vertices can move around, their relative positions are maintained. This scaffolding is constructed using an approximate proximity graph, in the form of a Delaunay triangulation (DT).

Once we form a DT, we check every edge in it and see if there are any node overlaps along that edge. Let w_i and h_i denote the half width and height of the node i, and $x_i^0(1)$ and $x_i^0(2)$ the current X and Y coordinates of this node. If i and j form an edge in the DT, we calculate the *overlap factor* of these two nodes

$$t_{\mathrm{ij}} = \max\left(\min\left(\frac{w_i + w_j}{|x_i^0(1) - x_j^0(1)|}, \frac{h_i + h_j}{|x_i^0(2) - x_j^0(2)|}\right), 1\right). \tag{5}$$

For nodes that do not overlap, $t_{\mathrm{ij}} = 1$. For nodes that do overlap, such overlaps can be removed if we expand the edge by this factor, see Figure 2. Therefore we want to generate a layout such that an edge in the proximity graph has the

ideal edge length close to $t_{\rm ij}\|x_i^0 - x_j^0\|$. In other words, we want to minimize the following stress function

$$\sum_{(i,j)\in E_P} w_{\rm ij}\,(\|x_i - x_j\| - d_{\rm ij}\,)^{\,2}. \tag{6}$$

Here $d_{\rm ij} = s_{\rm ij}\|x_i^0 - x_j^0\|$ is the ideal distance for the edge $\{i,j\}$, $s_{\rm ij}$ is a scaling factor related to the overlap factor $t_{\rm ij}$ (see (7)), $w_{\rm ij} = 1/\|d_{\rm ij}\|^2$ is a weighting factor, and E_P is the set of edges of the proximity graph. We call (6) the *proximity stress model* in obvious analogy.

Because DT is a planar graph, which has no more than $3|V| - 6$ edges, the above stress function has no more than $3|V| - 6$ terms. Furthermore, because DT is rigid, it provides a good scaffolding that constrains the relative position of the vertices and helps to preserve the global structure of the original layout.

It is important that we do not attempt to remove overlaps in one iteration by using the above model with $s_{\rm ij} = t_{\rm ij}$. Imagine the situation of a regular mesh graph, with one node i of particularly large size that overlaps badly with its nearby nodes, but the other nodes do not overlap with each other. Suppose nodes i and j form an edge in the proximity graph, and they overlap. If we try to make the length of the edge equal $t_{\rm ij}\|x_i^0 - x_j^0\|$, we will find that $t_{\rm ij}$ is a number much larger than 1, and the optimum solution to the stress model is to keep all the other vertices at or close to their current positions, but move the large node i outside of the mesh, at a position that does not cause overlap. This is not desirable because it destroys the original layout. Therefore we damp the overlap factor by setting

$$s_{\rm ij} = \min(t_{\rm ij}, s_{\rm max}) \tag{7}$$

and try to remove overlap a little at a time. Here $s_{\rm max} > 1$ is a number limiting the amount of overlap we are allowed to remove in one iteration. We found that $s_{max} = 1.5$ works well.

After minimizing (6), we arrive at a layout that may still have node overlaps. We then regenerate the proximity graph using DT and calculate the overlap factor along the edges of this graph, and redo the minimization. This forms an iterative process that ends when there are no more overlaps along the edges of the proximity graph.

For many graphs, the above algorithm yields a drawing that is free of node overlaps. For some graphs, however, especially those with nodes having extreme aspect ratios, node overlaps may still occur. Such overlaps happen for pairs of nodes that are not near each other, and thus do not constitute edges of the proximity graph. Figure 3(a) shows the drawing of a graph after minimizing (6) iteratively, so that no more node overlap is found along the edges of the Delaunay triangulation. Clearly, node 2 and node 4 still overlap. If we plot the Delaunay triangulation (Figure 3(b)), it is seen that nodes 2 and 4 are not neighbors in the proximity graph, which explains the overlap.

To overcome this situation, once the above iterative process has converged so that no more overlaps are detected over the DT edges, we apply a scan-

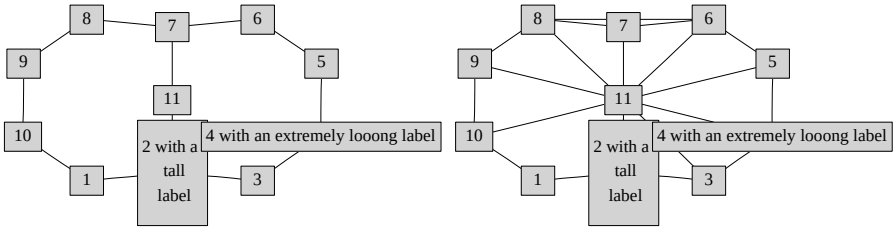

Figure 3: (a): A graph layout where nodes 2 and 4 overlap. (b): the proximity graph (Delaunay triangulation) of the current layout. No two nodes linked by an edge of the proximity graph overlap.

line algorithm [6] to find all overlaps, and augment the proximity graph with additional edges, where each edge consists of a pair of nodes that overlap. We then re-solve (6). This process is repeated until the scan-line algorithm finds no more overlaps. We call our algorithm PRISM (PRoxImity Stress Model). Algorithm 1 gives a detailed description of this algorithm.

Algorithm 1 Proximity stress model based overlap removal algorithm (PRISM)

Input: coordinates for each vertex, x_i^0, and bounding box width and height $\{w_i, h_i\},\ i = 1, 2, \ldots, |V|$.
repeat
  Form a proximity graph G_P of x^0 by Delaunay triangulation.
  Find the overlap factors (5) along all edges in G_P.
  Solve the proximity stress model (6) for x. Set $x^0 = x$.
until (no more overlaps along edges of G_P)
repeat
  Form a proximity graph G_P of x^0 by Delaunay triangulation.
  Find all node overlaps using a scan-line algorithm. Augment G_P with edges from node pairs that overlap.
  Find the overlap factor (5) along all edges of G_P.
  Solve the proximity stress model (6) for x. Set $x^0 = x$.
until (no more overlaps)

We now discuss some of the main computational steps in the above algorithm. Delaunay triangulation can be computed in $O(|V| \log |V|)$ time [10, 17, 27]. We used the mesh generator Triangle [33, 34] for this purpose.

The scan-line algorithm can be implemented to find all the overlaps in $O(l|V|(\log |V| + l))$ time [6], where l is the number of overlaps. Because we only apply the scan-line algorithm after no more node overlaps are found along edges of the proximity graph, l is usually a very small number, hence this step can be considered as taking time $O(|V| \log |V|)$.

The proximity stress model (6), like the spring model (1), can be solved

using the stress majorization technique [13] which is known to be a robust process for finding the minimum of (1). The technique works by bounding (6) with a series of quadratic functions from above, and the process of finding an optimum becomes that of finding the optimum of the series of quadratic functions, which involves solving linear systems with a weighted Laplacian of the proximity graph. We solve each linear system using a preconditioned conjugate gradient algorithm. Because we use DT as our proximity graph and it has no more than $3|V| - 6$ edges, each iteration of the conjugate gradient algorithm takes a time of $O(|V|)$.

Overall, therefore, Algorithm 1 takes $O(t(mk|V| + |V| \log |V|))$ time, where t is the total number of iterations in the two main loops in Algorithm 1, m is the average number of stress majorization iterations, and k the average number of iterations for the conjugate gradient algorithm.

In practice, we found that the majority of CPU time is spent in repeatedly solving the linear systems (which takes a total time of $O(tmk|V|)$). We terminate the conjugate gradient algorithm if the relative 2-norm residual for the linear system involved in the stress majorization process is less than 0.01. A tighter tolerance is not necessary because the solution of each linear system constitutes an intermediate step of the stress majorization. Furthermore, solution of the proximity stress model (6) is an intermediate step itself in Algorithm 1, so we do not need to solve (6) accurately either. Hence we set a limit of m_{max} iterations. By experimentation, we found that a smaller value m_{max} gives a faster algorithm, and that, in terms of quality, a smaller m_{max} is often just as good as, if not better than, a larger value of m_{max}. Therefore, we set $m_{max} = 1$.

5 Numerical Results

To evaluate the PRISM algorithm and other overlap removal algorithms, we apply them as a post-processing step to a selection of graphs from the `Graphviz` [14] test suite. This suite, part of the `Graphviz` source distribution, contains many graphs from users. As such, these are good examples of the kind of graphs actually being drawn.

Our baseline algorithm is Scalable Force Directed Placement (SFDP) [22], a multilevel, spring-electrical algorithm. Using the layout of SFDP, we then apply one of the overlap removal algorithms to get a new layout that has no node overlaps, and compare the new layout with the original in terms of dissimilarity and area.

In Table 1, we list the 14 test graphs, the number of vertices and edges, as well as CPU time[3] for PRISM and three other overlap removal algorithms. The graphs are selected randomly with the criteria that a graph chosen should be connected, and is of relatively large size. We compared PRISM with an implementation of the solve_VPSC algorithm [6] provided by its authors. We also evaluated the companion algorithm satisfy_VPSC. This offered, at best, a 2%

[3]All timings were derived on a 4 processor, 3.2 GHz Intel Xeon CPU, with 8.16 GB of memory, running Linux.

Table 1: Comparing the CPU time (in seconds) of several overlap removal algorithms. Initially the layout is scaled to an average edge length of 1 inch.

Graph	$\|V\|$	$\|E\|$	PRISM	VPSC	VORO	ODNLS
b100	1463	5806	1.44	14.85	350.7	258.9
b102	302	611	0.14	0.10	4.36	5.7
b124	79	281	0.03	0.01	0.02	0.5
b143	135	366	0.04	0.01	0.47	1.3
badvoro	1235	1616	0.54	71.15	351.51	73.6
mode	213	269	0.09	0.09	2.15	2.1
ngk10_4	50	100	0.01	0.00	0.02	0.14
NaN	76	121	0.01	0.01	0.11	0.27
dpd	36	108	0.01	0.01	0.02	0.1
root	1054	1083	0.89	7.81	398.49	46.9
rowe	43	68	0.00	0.00	0.04	0.1
size	47	55	0.01	0.00	0.06	0.09
unix	41	49	0.01	0.00	0.04	0.07
xx	302	611	0.13	0.10	8.19	5.67

reduction in time, while producing almost identical results concerning displacement, dissimilarity and area. For this reason, in the sequel, we only consider the solve_VPSC algorithm, hereafter denoted as VPSC.[4] We also evaluated the Voronoi cluster busting algorithm [15, 29], denoted by VORO, as well as the ODNLS algorithm of Li et al. [28], which relies on varied edge lengths in a spring embedder. We note, in passing, that we also tested scaling using the algorithm of Marriott et al. [30]. As expected, it outperformed all other algorithms in terms of speed and dissimilarity, but at an unacceptably high cost in area. For example, on the b143 graph, the time and dissimilarity were essentially 0 but the area was 82.2. Overall, scaling produced areas at least 4.5 times larger than PRISM and, more typically, at least an order of magnitude larger. We therefore remove scaling from further consideration.

The initial layout by SFDP is scaled so that the average edge length is 1 inch. From the table, it is seen that PRISM is usually faster, particularly for large graphs on which it scales much better. The others are slow for large graphs, with VORO the slowest.

Table 2 compares the dissimilarities and drawing area of the four overlap removal algorithms. The smaller the dissimilarities and area, the better.

The ODNLS algorithm performs best in terms of smaller dissimilarity, followed by PRISM, VPSC and VORO. In terms of area, PRISM and VPSC are pretty close, and both are better than ODNLS and VORO, which can give ex-

[4] Both versions of the VPSC algorithm can be extended to support the preservation of orthogonal ordering. We did not have an opportunity to check these versions.

tremely large drawings. Indeed, in terms of area, scaling outperformed ODNLS and VORO in 20%-30% of the examples.

Comparing PRISM with VPSC, Table 2 shows that PRISM gives smaller dissimilarities most of the time. The two dissimilarity measures, σ_{dist} and σ_{disp}, are generally correlated, except for `ngk10_4` and `root`. Based on σ_{dist}, VPSC is better for these two graphs, while based on σ_{disp}, PRISM is better. The first row in Figure 4 shows the original layout of `ngk10_4`, as well as the result after applying PRISM and VPSC. Through visual inspection, we can see that PRISM preserved the proximity relations of the original layout well. VPSC "packed" the labels more tightly, but it tends to line up vertices horizontally and vertically, and also produces a layout with aspect ratio quite different from the original graph. It seems that σ_{dist} is not as sensitive in detecting differences in aspect ratio. This is evident in drawings of the `root` graph (Figure 4, second row): VPSC clearly produced a drawing that is overly stretched in the vertical direction, but its σ_{dist} is actually smaller! Consequently, we conclude that σ_{disp} may be a better dissimilarity measure.

The fact that VPSC can produce very tall and thin, or very short and wide, layouts is not surprising, and has been observed often in practice. VPSC works in the vertical and horizontal directions alternatively, each time trying to remove overlaps while minimizing displacement. As a result, when starting from a layout with severe node overlaps, it may move vertices significantly along one direction to resolve the overlaps, creating drawings with extreme aspect ratios. In fact, for 9 out of 14 test graphs, VPSC produces layouts with extreme aspect ratios. PRISM does not suffer from this problem.

When starting from a layout that is scaled sufficiently so that relative fewer nodes overlap, VPSC's performance can be improved. Table 3 compares the four overlap removal algorithms, starting from layouts that are scaled to give an average edge length that equals 4 times the average node size. Here the size of a node is calculated as the average of its width and height.

From the table, we can see that in terms of dissimilarity, PRISM and VPSC are now similar, closer to the better performing ODNLS. In terms of drawing area, PRISM is better than VPSC, with VORO and ODNLS much larger. When visually inspected, VPSC again suffers from extreme aspect ratio issue on at least 5 out of the 14 graphs (`b100, b143, badvoro, mode, root`). Figure 5 shows the layout of `badvoro` (first row), on which VPSC performed badly based on the two similarity measures. In the same figure, we also show `b124` (second row), on which PRISM is rated worse than VPSC based on the same measures. On `badvoro` it is clear that VPSC performed badly, as the similarity measures suggest. On the other hand, if we look at `b124`, VPSC perhaps performed better than PRISM, but not as clearly as the similarity measures suggest. Overall, visual inspection of the drawings of these 14 graphs, as well as drawings for graphs in the complete `Graphviz` test suite (a total of 204 graphs in March 2008), shows that PRISM performs very well, and is overall better and faster than VPSC and VORO. The ODNLS algorithm preserves similarity somewhat better than PRISM, but at much higher costs in term of speed and area.

Considering a larger collection of graphs, Table 4 compares PRISM with

Table 2: Comparing the dissimilarities and area of overlap removal algorithms. Results shown are σ_{dist}, σ_{disp} and area. Area is measured with a unit of 10^6 square points. Initially the layout is scaled to an average length of 1 inch.

Graph	PRISM			VPSC		
	σ_{dist}	σ_{disp}	area	σ_{dist}	σ_{disp}	area
b100	0.74	0.38	14.05	0.76	0.72	18.91
b102	0.44	0.25	2.45	0.58	0.8	2.71
b124	0.65	0.37	1.04	0.78	0.73	0.91
b143	0.59	0.35	1.5	0.78	0.83	2.16
badvoro	0.34	0.15	12.58	0.61	0.75	13.85
mode	0.59	0.37	0.79	1.02	0.77	1.29
ngk10_4	0.41	0.16	0.33	0.39	0.3	0.25
NaN	0.4	0.2	0.72	0.54	0.65	0.71
dpd	0.34	0.18	0.25	0.51	0.4	0.18
root	0.71	0.3	16.99	0.6	0.75	17.68
rowe	0.33	0.14	0.22	0.44	0.31	0.19
size	0.37	0.2	0.47	0.77	0.74	0.4
unix	0.39	0.23	0.39	0.51	0.67	0.36
xx	0.42	0.25	3.96	0.57	0.82	3.9
Graph	VORO			ODNLS		
	σ_{dist}	σ_{disp}	area	σ_{dist}	σ_{disp}	area
b100	-	-	-	0.33	0.20	1.02E3
b102	0.8	0.3	31.79	0.30	0.16	53.13
b124	0.86	0.39	13.42	0.33	0.19	14.79
b143	0.99	0.45	22.91	0.49	0.34	23.79
badvoro	2.29	0.65	3.01E3	0.31	0.26	318.66
mode	0.97	0.54	10.84	0.38	0.27	49.45
ngk10_4	0.48	0.26	0.52	0.22	0.13	2.30
NaN	0.56	0.28	5.04	0.26	0.15	5.10
dpd	0.48	0.32	0.45	0.37	0.29	1.30
root	4.09	0.94	6.93E9	0.29	0.22	950.01
rowe	0.49	0.26	0.95	0.27	0.12	2.10
size	0.62	0.35	1.27	0.32	0.20	4.14
unix	0.6	0.35	0.85	0.26	0.13	2.35
xx	0.97	0.34	58.83	0.29	0.14	74.00

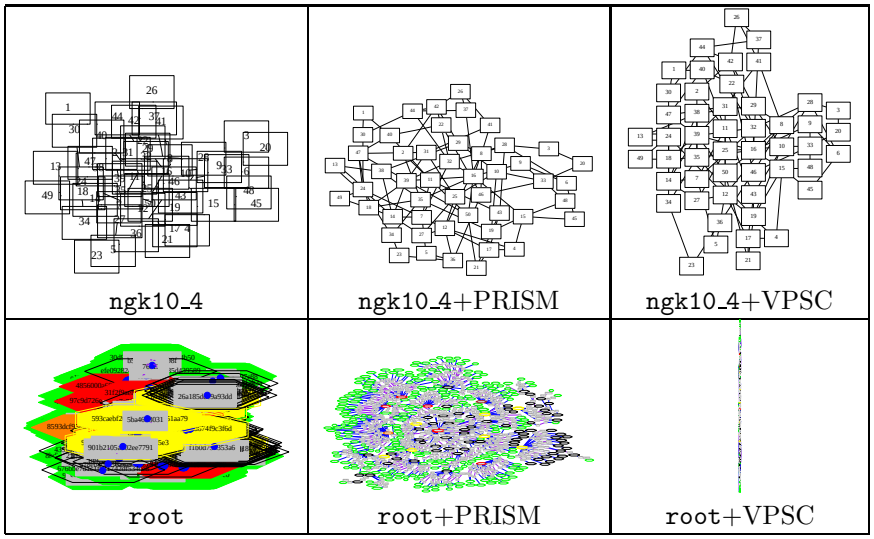

Figure 4: Divergence of dissimilarity measures: for both graphs, σ_{dist} estimates that VPSC gives layout closer to the original, while σ_{disp} predicts the opposite.

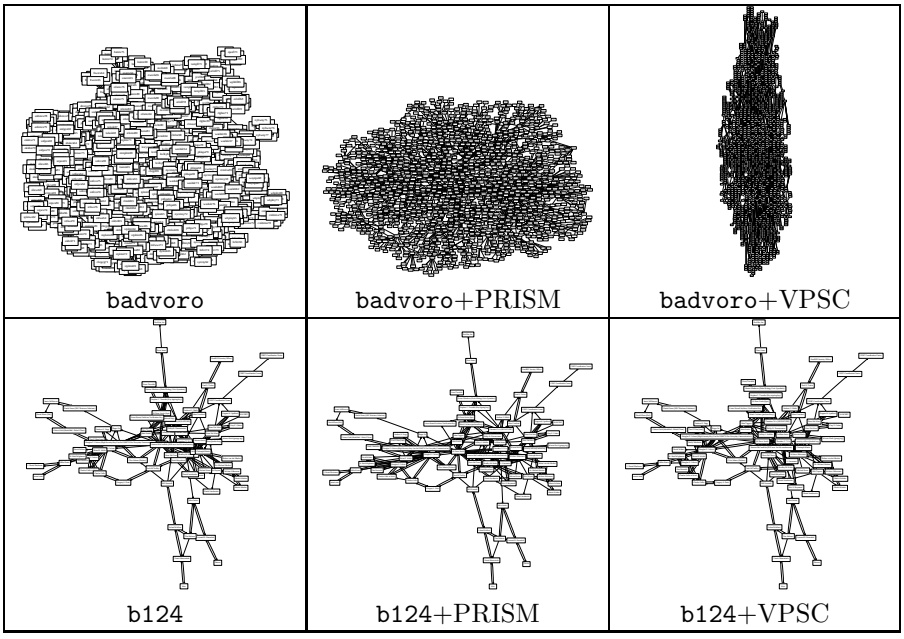

Figure 5: Comparing PRISM and VPSC on two graphs. Original layouts are scaled to have an average edge length that equals 4 times the label size.

Table 3: Comparing the dissimilarities and area of overlap removal algorithms. Results shown are σ_{dist}, σ_{disp} and area. Area is measured with a unit of 10^6 square points. Initially the layout is scaled to an average edge length that equals 4 times the average label size.

Graph	PRISM			VPSC		
	σ_{dist}	σ_{disp}	area	σ_{dist}	σ_{disp}	area
b100	0.74	0.36	14.59	0.95	0.62	25.33
b102	0.41	0.18	2.62	0.44	0.33	2.77
b124	0.52	0.12	3.39	0.28	0.06	3.04
b143	0.5	0.22	2.59	0.67	0.41	4.26
badvoro	0.35	0.15	11.85	0.65	0.58	11.71
mode	0.57	0.35	0.78	0.84	0.59	1.45
ngk10_4	0.25	0.05	0.54	0.16	0.03	0.53
NaN	0.25	0.06	1.15	0.15	0.04	1.15
dpd	0.14	0.03	0.54	0.1	0.03	0.53
root	0.73	0.28	16.02	0.73	0.7	25.93
rowe	0.16	0.04	0.38	0.11	0.03	0.38
size	0.35	0.11	0.54	0.28	0.08	0.57
unix	0.25	0.07	0.6	0.13	0.04	0.59
xx	0.39	0.18	4.23	0.45	0.34	4.54
Graph	VORO			ODNLS		
	σ_{dist}	σ_{disp}	area	σ_{dist}	σ_{disp}	area
b100	-	-	-	0.49	0.24	1.09E3
b102	1.18	0.47	131.55	0.33	0.15	41.01
b124	0.69	0.36	19.09	0.43	0.26	14.64
b143	0.84	0.39	19.35	0.57	0.36	29.39
badvoro	3.8	0.9	4.45E5	0.34	0.10	166.53
mode	1.09	0.58	20.02	0.44	0.28	45.21
ngk10_4	0.46	0.2	0.79	0.23	0.12	2.21
NaN	0.49	0.22	2.28	0.33	0.18	4.86
dpd	0.43	0.23	0.6	0.38	0.29	1.45
root	4.14	0.94	3.68E8	0.39	0.14	1.52E3
rowe	0.41	0.2	0.68	0.25	0.11	2.01
size	0.62	0.34	2.07	0.28	0.16	3.35
unix	0.55	0.24	0.64	0.20	0.09	2.01
xx	1.13	0.44	185.7	0.30	0.11	45.18

Table 4: Comparing dissimilarity and area of overlap removal algorithms on the Rome suite of test graphs. # stands for the number of graphs within the size range specified.

sizes	#	PRISM		VPSC		VORO		ODNLS	
		σ_{disp}	area	σ_{disp}	area	σ_{disp}	area	σ_{disp}	area
10–19	1407	0.009	0.16	0.09	0.061	0.14	0.069	0.07	0.34
20–29	839	0.02	0.27	0.11	0.12	0.14	0.15	0.078	0.92
30–39	2036	0.03	0.36	0.13	0.19	0.13	0.26	0.078	1.85
40–49	1800	0.035	0.43	0.14	0.25	0.12	0.36	0.074	2.71
50–59	1045	0.042	0.51	0.15	0.32	0.11	0.53	0.074	4.53
60–69	1172	0.046	0.58	0.17	0.39	0.11	0.70	0.074	6.18
70–79	1008	0.051	0.64	0.17	0.46	0.11	0.94	0.073	8.44
80–89	788	0.05	0.71	0.17	0.52	0.10	1.20	0.07	9.99
90–99	1296	0.054	0.78	0.18	0.59	0.10	1.49	0.069	13.26
100–109	140	0.055	0.82	0.18	0.61	0.10	1.7	0.068	14.14

VPSC, VORO and ODNLS on the Rome test suite of graphs [4]. This suite has a total of 11534 graphs[5] of relatively small size. Due to space limitation, we only give the similarity measure σ_{disp} and the area, and we average the results over graphs of similar sizes. Again, PRISM achieves the best compromise between being close to the original drawing, and having a smaller drawing area.

We note that while there is no theoretical result guaranteeing that PRISM algorithm converges to an overlap free layout in a finite number of iterations, in practice, out of thousands of graphs tested, some as large as tens of thousands of vertices, PRISM always converges within a few hundred total number of iterations in the two main loops in Algorithm 1. In our implementation we set a limit of 1000 iterations, even though this limit has never been observed to be reached. Table 5 gives the number of iterations taken for the 14 test cases in Tables 2-3. As can be seen, for these graphs, the maximum number of iterations is 122.

As a demonstration of the scalability of PRISM, we consider its application to a large graph. This is a tree from the Mathematics Genealogy Project [32]. Each node is a mathematician, and an edge from node i to node j means that j is the first supervisor of i. The graph is disconnected and consists of thousands of components. Here we consider the second largest component with 11766 vertices. This graph took 31 seconds to lay out using SFDP, and 15 seconds post-processing using PRISM for overlap removal. PRISM converges in 81 iterations. Important mathematicians (those with the most offspring) and important edges (those that lead to the largest subtrees) are highlighted with larger nodes

[5]Three graphs were dropped from the test because they were disconnected.

Table 5: Number of iterations taken in the two main loops in Algorithm 1. A: initially the layout is scaled to an average edge length of 1 inch. B: initially the layout is scaled to an average edge length that equals 4 times the average label size.

Graph	$\|V\|$	$\|E\|$	A	B
b100	1463	5806	122	75
b102	302	611	66	44
b124	79	281	43	19
b143	135	366	46	26
badvoro	1235	1616	53	32
mode	213	269	43	26
ngk10_4	50	100	29	8
NaN	76	121	34	9
dpd	36	108	26	6
root	1054	1083	103	119
rowe	43	68	28	7
size	47	55	29	14
unix	41	49	32	8
xx	302	611	53	46

and thicker edges. Figure 6(top) gives the overall layout, which shows that PRISM preserved the tree structure of the layout very well after node overlap removal. Figure 6(bottom) gives a close up view of the details of a small area in the center-left part of Figure 6(top) with many famous mathematicians of early generations. Additional drawings of this and other components of the Mathematics Genealogy Project graph, including that of the largest component, are available [21].

6 Conclusions and Future Work

A number of algorithms have been proposed for removing node overlaps in undirected graph drawings. For graphs that are relatively large with nontrivial connectivities, these algorithms often fail to produce satisfactory results, either because the resulting drawing is too large (e.g., scaling, VORO, ODNLS), or the drawing becomes highly skewed (e.g., VPSC). In addition, many of them do not scale well with the size of the graph in terms of computational costs. The main contributions of this paper is a new algorithm for removing overlaps that is both highly effective and efficient. The algorithm is shown to produce layouts that preserve the proximity relations between vertices, and scales well with the size of the graph. It has been applied to graphs of tens of thousands of vertices, and is able to give aesthetic, overlap-free drawings with compact area

in seconds, which is not feasible with any algorithm known to us.

It is possible that algorithms such as VPSC, which rely on separate passes in the X and Y directions, might be improved by randomizing which overlaps are removed in which pass or by gradually removing overlaps using many alternating X and Y passes. This would, however, further increase their computational cost, which is already much higher than the algorithm proposed in this paper.

For future work, we would like to extend the overlap removal algorithm to deal with edge-node overlaps. We would also like to explore the possibility of using the proximity stress model for packing disconnected components.

Acknowledgments

We would like to thank Tim Dwyer and Wanchun Li for making their implementations of VPSC and ODNLS, respectively, available to us. We would like to thank Yehuda Koren and Stephen North for helpful discussions, and Stephen Kobourov for bringing our attention to the term "Frobenius metric". Finally, the reviewers' comments were very helpful, especially in making the exposition clearer and more accurate.

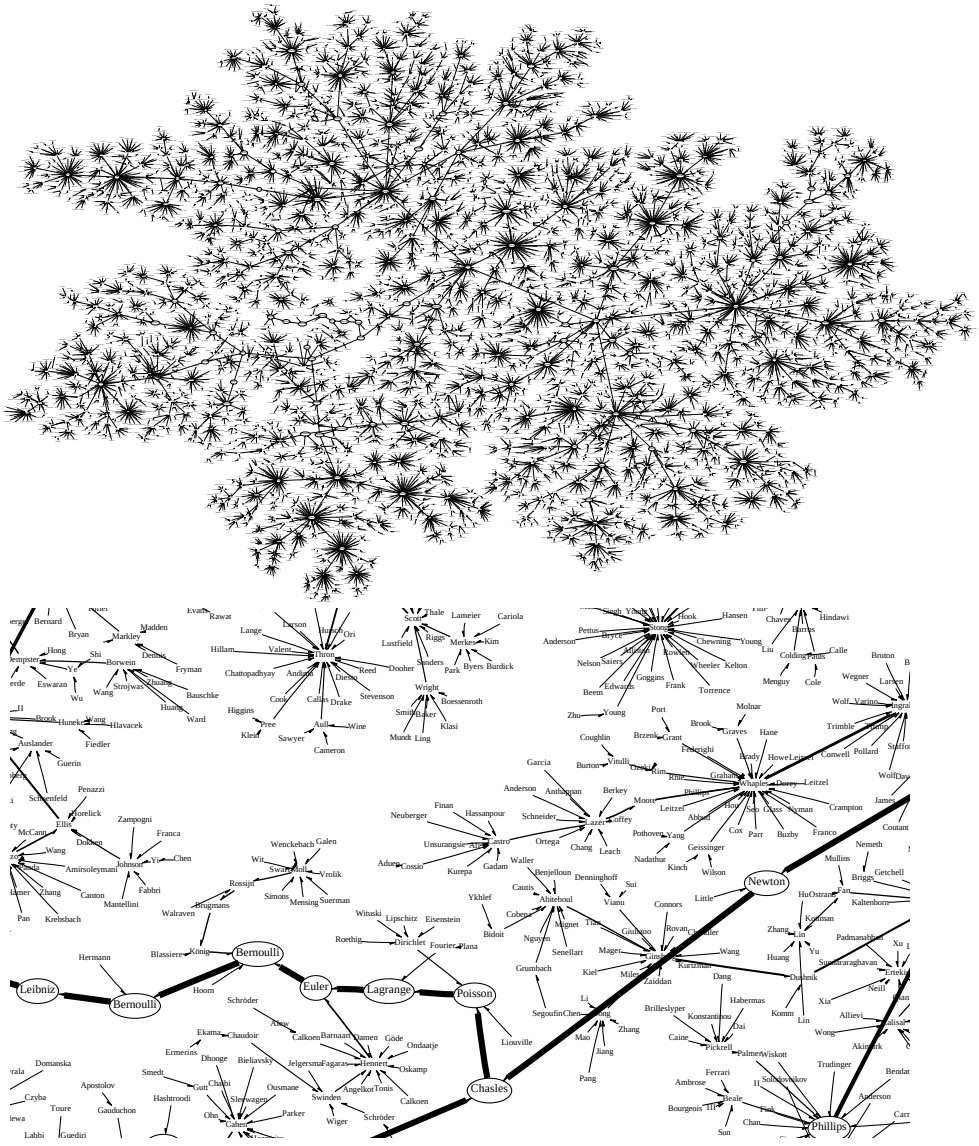

Figure 6: The second largest component from the Mathematics Genealogy Project. Top: overall layout with node overlap removed. Bottom: close up view of a small area of the center-left part of above.

References

[1] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 1997.

[2] U. Brandes and C. Pich. An experimental study on distance based graph drawing. In *GD'08: Proceedings of the Symposium on Graph Drawing*, 2008 (to appear).

[3] J.-H. Chuang, C.-C. Lin, and H.-C. Yen. Drawing graphs with nonuniform nodes using potential fields. In *Proc. 11th Intl. Symp. on Graph Drawing*, volume 2012 of *LNCS*, pages 460–465. Springer-Verlag, 2003.

[4] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *CGTA: Computational Geometry: Theory and Applications*, 7:303–325, 1997.

[5] T. Dwyer, Y. Koren, and K. Marriott. Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):821–828, 2006.

[6] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *Proc. 13th Intl. Symp. Graph Drawing (GD '05)*, volume 3843 of *LNCS*, pages 153–164. Springer, 2006.

[7] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[8] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. on Graphics*, 13(1):43–72, 1994.

[9] C. Erten, A. Efrat, D. Forrester, A. Iyer, and S. G. Kobourov. Force-directed approaches to sensor network localization. In R. Raman, R. Sedgewick, and M. F. Stallmann, editors, *Proc. 8th Workshop Algorithm Engineering and Experiments (ALENEX)*, pages 108–118. SIAM, 2006.

[10] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[11] C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. In *Proceedings of the 27th conference on Australasian computer science (ACSC2004)*, pages 369–376. Australian Computer Society, 2004.

[12] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force directed placement. *Software - Practice and Experience*, 21:1129–1164, 1991.

[13] E. R. Gansner, Y. Koren, and S. C. North. Graph drawing by stress majorization. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, volume 3383 of *LNCS*, pages 239–250. Springer, 2004.

[14] E. R. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice & Experience*, 30:1203–1233, 2000.

[15] E. R. Gansner and S. C. North. Improved force-directed layouts. In *Proc. 6th Intl. Symp. Graph Drawing (GD '98)*, volume 1547 of *LNCS*, pages 364–373. Springer, 1998.

[16] J. C. Gower and G. B. Dijksterhuis. *Procrustes Problems*. Oxford University Press, 2004.

[17] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[18] S. Hachul and M. Jünger. Drawing large graphs with a potential field based multilevel algorithm. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, volume 3383 of *LNCS*, pages 285–295. Springer, 2004.

[19] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *J. Graph Algorithms and Applications*, 6:179–202, 2002.

[20] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara. A layout adjustment problem for disjoint rectangles preserving orthogonal order. In *Proc. 6th Intl. Symp. Graph Drawing (GD '98)*, volume 1547 of *LNCS*, pages 183–197. Springer, 1998.

[21] Y. F. Hu. Drawings of the mathematics genealogy project graphs. http://www.research.att.com/~yifanhu/GALLERY/MATH_GENEALOGY.

[22] Y. F. Hu. Efficient and high quality force-directed graph drawing. *Mathematica Journal*, 10:37–71, 2005.

[23] Y. F. Hu and Y. Koren. Extending the spring-electrical model to overcome warping effects. In *Proceedings of IEEE Pacific Visualization Symposium 2009 (PacificVis '09)*, pages 129–136, 2009.

[24] X. Huang and W. Lai. Force-transfer: A new approach to removing overlapping nodes in graph layout. In *Proc. 25th Australian Computer Science Conference*, pages 349–358, 2003.

[25] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80:1502–1517, 1992.

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

[27] G. Leach. Improving worst-case optimal Delaunay triangulation algorithms. In *4th Canadian Conference on Computational Geometry*, pages 340–346, 1992.

[28] W. Li, P. Eades, and N. Nikolov. Using spring algorithms to remove node overlapping. In *Proc. Asia-Pacific Symp. on Information Visualisation*, pages 131–140, 2005.

[29] K. A. Lyons, H. Meijer, and D. Rappaport. Algorithms for cluster busting in anchored graph drawing. *J. Graph Algorithms and Applications*, 2(1), 1998.

[30] K. Marriott, P. J. Stuckey, V. Tam, and W. He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8(2):143–171, 2003.

[31] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995.

[32] North Dakota State University Dept. of Math. The mathematics genealogy project.
http://genealogy.math.ndsu.nodak.edu/.

[33] J. R. Shewchuk. Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *LNCS*, pages 203–222. Springer, 1996.

[34] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22:21–74, 2002.

[35] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Systems, Man and Cybernetics*, SMC-11(2):109–125, 1981.

[36] C. Walshaw. A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms and Applications*, 7:253–285, 2003.

[37] X. Wang and I. Miyamoto. Generating customized layouts. In *GD '95: Proceedings of the Symposium on Graph Drawing*, volume 1027 of *LNCS*, pages 504–515. Springer, 1995.