Efficient Node Overlap Removal Using a Proximity Stress Model

Emden R. Gansner and Yifan Hu

AT&T Labs, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932
{erg,yifanhu}@research.att.com

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 is to avoid overlaps while retaining the structural information inherent in a layout using little additional area. This paper presents a new node overlap removal algorithm that does well by these measures.

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 [23] while preserving the node size until the nodes no longer overlap. 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. Various authors [2, 13, 21, 26] have extended the spring-electrical model [4, 7] to take into account node sizes, usually as increased repulsive forces. Node overlap removal can also be built into the stress model [19] 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 rely on overly large repulsive forces, or the type of post-processing step considered next.

An alternative approach is to remove 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. A number of such algorithms have been proposed. For example, the Voronoi cluster busting algorithm [10, 22] works by iteratively forming a Voronoi diagram from the current layout and moving each node to the center of its Voronoi cell

I.G. Tollis and M. Patrignani (Eds.): GD 2008, LNCS 5417, pp. 206–217, 2009.

until no overlaps remain. Although roughly maintaining relative node positions, the overall affect is to lose much of the layout structure.

Another group of post-processing algorithms is based on maintaining the orthogonal ordering [25] of the initial layout as a way to preserve its shape. A force scan algorithm and variants were proposed [14, 17, 21, 25] based on these constraints. More recently, Marriott et al. [3,23] have presented a quadratic programming algorithm which removes node overlaps while minimizing node displacement and keeping the orthogonal ordering. An orthogonal ordering invariant is fairly effective at preserving structure, but it still cannot ensure that relative proximity relations between nodes are preserved, while at other times, it is too restrictive. Also, some of these algorithm require, in practice, separate horizontal and vertical passes which often results in a layout with a distorted aspect ratio (e.g., Fig. 2, bottom right).

In this paper, we discuss (Sect. 2) metrics for the similarity between two layouts which we believe better quantifies the desired outcome of overlap removal than minimized displacement or such simpler measures as aspect ratio or edge ratio. We then present (Sect. 3) a node overlap removal algorithm based on a proximity graph of the nodes in the original layout. In Sect. 4, we evaluate our algorithm and others using the proposed similarity measures.

In the following, 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.

2 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.

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 [5]. However, calculating all pairwise distances is expensive for large graphs, both in CPU time and in the amount of memory, so instead we form a *Delaunay triangulation* (DT) of the original graph, then measure the distance between vertices along the edges of the triangulation for the original and new layouts. 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

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

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

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

where

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

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, but are clearly different. 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 [3]. 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 define the displacement dissimilarity as

$$\sigma_{\text{disp}}(x^0, x) = \min_{p\in\mathbb{R}^2,\theta,r\in\mathbb{R}} \sum_{i\in V} \|rTx_i + p - x_i^0\|^2, \tag{1}$$

where r is the scaling, θ the rotation with $T = T(\theta)$ its rotation matrix, and $p \in \mathbb{R}^2$ is the translation. Solving this is a known problem in Procrustes analysis [1, 11] and the solution (the Procrustes statistic) is

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

where 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. 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.

3 A Proximity Stress Model for Node Overlap Removal

Our goal now is to remove overlaps while preserving the shape of the initial layout by maintaining the proximity relations. 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 a proximity graph [18]. Here again, we work with the Delaunay triangulation.

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

$$t_{\text{ij}} = \max\left(\min\left(\frac{w_i + w_j}{|x_i^0(1) - x_j^0(1)|}, \frac{h_i + h_j}{|x_i^0(2) - x_j^0(2)|}\right), 1\right). \tag{3}$$

For nodes that do not overlap, $t_{\text{ij}} = 1$. For nodes that do overlap, such overlaps can be removed if we expand the edge by this factor. Therefore we want to generate a layout such that an edge in the proximity graph has the ideal edge length close to $t_{\text{ij}}\|x_i^0 - x_j^0\|$. In other words, we want to minimize the following stress function

$$\sum_{(i,j)\in E_P} w_{\text{ij}} \left(\|x_i - x_j\| - d_{\text{ij}}\right)^2. \tag{4}$$

Here $d_{\text{ij}} = s_{\text{ij}}\|x_i^0 - x_j^0\|$ is the ideal distance for the edge $\{i, j\}$, s_{ij} is a scaling factor related to the overlap factor t_{ij} (see (6)), $w_{\text{ij}} = 1/||d_{\text{ij}}||^2$ is a scaling factor, and E_P is the set of edges of the proximity graph. We call (4) the *proximity stress model* in obvious analogy with the standard stress model [19]

$$\sum_{i\neq j} w_{\text{ij}} \left(\|x_i - x_j\| - d_{\text{ij}}\right)^2, \tag{5}$$

where d_{ij} is the graph theoretical distance between vertices i and j, and w_{ij} is a weight factor, typically $1/{d_{\text{ij}}}^2$.

Because DT is a planar graph, which has no more than $3|V| - 3$ edges, the above stress function has no more than $3|V| - 3$ 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 $s_{\text{ij}} = t_{\text{ij}}$. 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 $t_{\text{ij}}\|x_i^0 - x_j^0\|$, we will find that t_{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

$$s_{\text{ij}} = \min(t_{\text{ij}}, s_{\text{max}}) \tag{6}$$

and try to remove overlaps a little at a time. Here $s_{\text{max}} > 1$ 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.

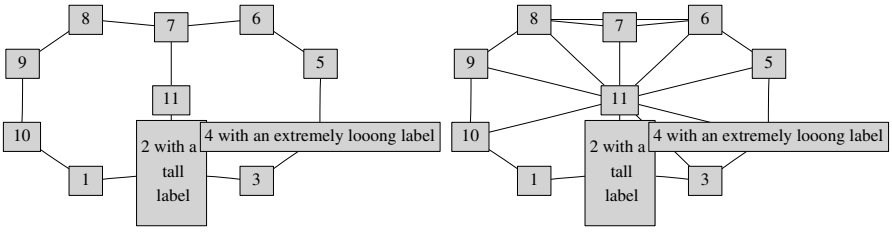

Fig. 1. (a): A graph layout where nodes 2 and 4 overlap. (b): the proximity graph (Delaunay triangulation) of the current layout. No two nodes linked by an edge of the proximity graph overlap.

After minimizing (4), 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. Fig. 1(a) shows the drawing of a graph after minimizing (4) 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 (Fig. 1(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-line algorithm [3] 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 (4). This process is repeated until the scan-line algorithm finds no more overlaps.

We call this algorithm PRISM (PRoxImity Stress Model). Concerning its complexity, Delaunay triangulation can be computed in $O(|V|log(|V|))$ time [6, 12, 20]. The scan-line algorithm can be implemented to find all the overlaps in $O(l|V|(log|V| + l))$ time [3], 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 (4), like the standard stress model (5), can be solved using the stress majorization technique [8] with a conjugate gradient algorithm. Because we use DT as our proximity graph and it has no more than $3|V| - 3$ edges, each iteration of the conjugate gradient algorithm takes a time of $O(|V|)$.

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

4 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` [9] 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) [16], a multi-level, 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[1] 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 in `Graphviz` of the solve_VPSC algorithm [3][2], hereafter denoted as VPSC, as well as VORO, the Voronoi cluster busting algorithm [10,22]. The final algorithm is the ODNLS algorithm of Li et al. [21], which relies on varied edge lengths in a spring embedder.

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 extremely 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 Fig. 2 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 (Fig. 2, second row). VPSC clearly produced a drawing that is overly stretched in the vertical direction, but its σ_{dist} is actually smaller than that of PRISM! 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

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

[2] A stand alone version of solve_VPSC by the authors of this algorithm has also been tried but was found to offer no advantage over VPSC. VPSC itself was also contributed originally by the same authors to `Graphviz`.

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

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

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

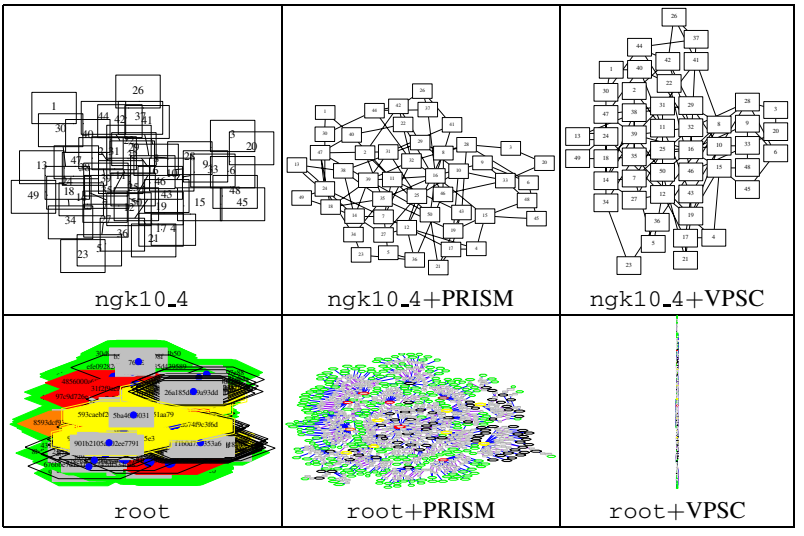

Fig. 2. Divergence of dissimilarity measures: for both graphs, σ_{dist} estimates that VPSC gives layout closer to the original, while σ_{disp} predicts the opposite

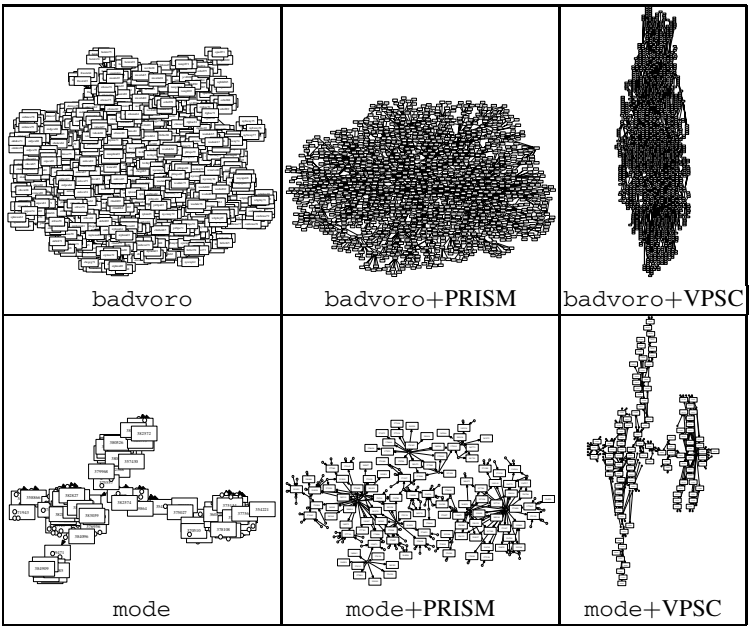

Fig. 3. Comparing PRISM and VPSC on two graphs. Original layouts are scaled to have an average edge length that equals 4 times the label size.

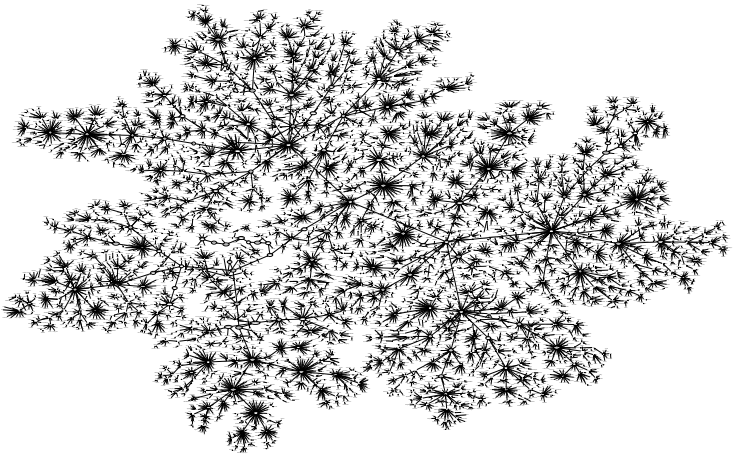

Fig. 4. The second largest component from the Mathematics Genealogy Project

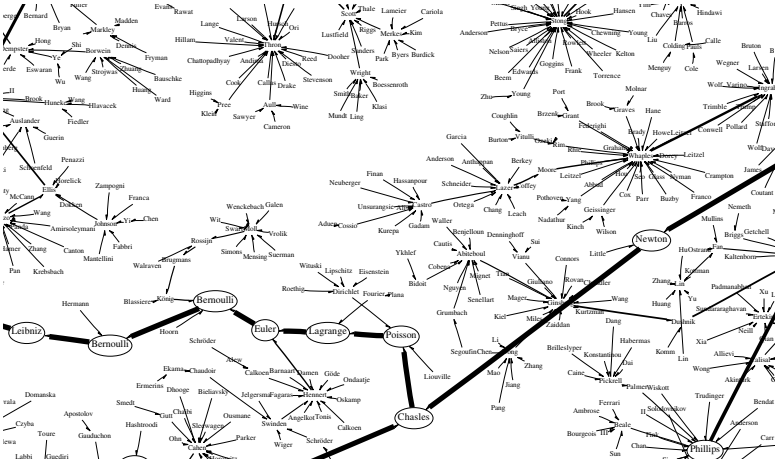

Fig. 5. Close-up view of the center-left part of Fig. 4

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.

We experimented with layouts initially scaled sufficiently so that relatively fewer nodes overlap. For example, when initial layouts were scaled to give an average edge length equal to 4 times the average node size, we found that the performance of VPSC was improved. Nevertheless it still suffered from extreme aspect ratio on at least 5 out of the 14 graphs. Figure 3 shows two of these graphs.

Overall, quantitative and visual comparison 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.

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 [24]. 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 layout using SFDP, and 15 seconds post-processing using PRISM for overlap removal. Important mathematicians (those with the most offspring) and important edges (those that lead to the largest subtrees) are highlighted with larger nodes and thicker edges. Figure 4 gives the overall layout, which shows that PRISM preserved the tree structure of the layout very well after node overlap removal. Figure 5 gives a close up view of the details of a small area in the center-left part of Fig. 4. Additional drawings of this and other components of the Mathematics Genealogy Project graph, including that of the largest component, are available [15].

5 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 contribution 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; Yehuda Koren and Stephen

North for helpful discussions; Stephen Kobourov for bringing our attention to the term "Frobenius metric"; and the referees for valuable suggestions and references.

References

1. Borg, I., Groenen, P.: Modern Multidimensional Scaling: Theory and Applications. Springer, Heidelberg (1997)
2. Chuang, J.H., Lin, C.C., Yen, H.C.: Drawing graphs with nonuniform nodes using potential fields. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 460–465. Springer, Heidelberg (2001)
3. Dwyer, T., Marriott, K., Stuckey, P.J.: Fast node overlap removal. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 153–164. Springer, Heidelberg (2006)
4. Eades, P.: A heuristic for graph drawing. Congressus Numerantium 42, 149–160 (1984)
5. Erten, C., Efrat, A., Forrester, D., Iyer, A., Kobourov, S.G.: Force-directed approaches to sensor network localization. In: Raman, R., Sedgewick, R., Stallmann, M.F. (eds.) Proc. 8th Workshop Algorithm Engineering and Experiments (ALENEX), pp. 108–118. SIAM, Philadelphia (2006)
6. Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica 2, 153–174 (1987)
7. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force directed placement. Software - Practice and Experience 21, 1129–1164 (1991)
8. Gansner, E.R., Koren, Y., North, S.C.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
9. Gansner, E.R., North, S.: An open graph visualization system and its applications to software engineering. Software - Practice & Experience 30, 1203–1233 (2000)
10. Gansner, E.R., North, S.C.: Improved force-directed layouts. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 364–373. Springer, Heidelberg (1999)
11. Gower, J.C., Dijksterhuis, G.B.: Procrustes Problems. Oxford University Press, Oxford (2004)
12. Guibas, L., Stolfi, J.: Primitives for the manipulation of general subdivisions and the computation of voronoi. ACM Trans. Graph. 4(2), 74–123 (1985)
13. Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. J. Graph Algorithms and Applications 6, 179–202 (2002)
14. Hayashi, K., Inoue, M., Masuzawa, T., Fujiwara, H.: A layout adjustment problem for disjoint rectangles preserving orthogonal order. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 183–197. Springer, Heidelberg (1999)
15. Hu, Y.F.: Drawings of the mathematics genealogy project graphs, `http://www.research.att.com/~yifanhu/GALLERY/MATH_GENEALOGY`
16. Hu, Y.F.: Efficient and high quality force-directed graph drawing. Mathematica Journal 10, 37–71 (2005)
17. Huang, X., Lai, W.: Force-transfer: A new approach to removing overlapping nodes in graph layout. In: Proc. 25th Australian Computer Science Conference, pp. 349–358 (2003), `citeseer.ist.psu.edu/564050.html`
18. Jaromczyk, J.W., Toussaint, G.T.: Relative neighborhood graphs and their relatives. Proc. IEEE 80, 1502–1517 (1992)
19. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters 31, 7–15 (1989)
20. Leach, G.: Improving worst-case optimal Delaunay triangulation algorithms. In: 4th Canadian Conference on Computational Geometry. pp. 340–346 (1992), `citeseer.ist.psu.edu/leach92improving.html`

21. Li, W., Eades, P., Nikolov, N.: Using spring algorithms to remove node overlapping. In: Proc. Asia-Pacific Symp. on Information Visualisation, pp. 131–140 (2005)
22. Lyons, K.A., Meijer, H., Rappaport, D.: Algorithms for cluster busting in anchored graph drawing. J. Graph Algorithms and Applications 2(1) (1998)
23. Marriott, K., Stuckey, P.J., Tam, V., He, W.: Removing node overlapping in graph layout using constrained optimization. Constraints 8(2), 143–171 (2003)
24. Department of mathematics at North Dekota State University: The mathematics genealogy project, `http://genealogy.math.ndsu.nodak.edu/`
25. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Vis. Lang. Comput. 6(2), 183–210 (1995)
26. Wang, X., Miyamoto, I.: Generating customized layouts. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 504–515. Springer, Heidelberg (1996)