Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 21, no. 5, pp. 857–872 (2017)
DOI: 10.7155/jgaa.00442

Node Overlap Removal by Growing a Tree

Lev Nachmanson[1] *Arlind Nocaj*[2] *Sergey Bereg*[3] *Leishi Zhang*[4]
Alexander Holroyd[1]

[1]Microsoft Research, Redmond, USA
[2]University of Konstanz, Konstanz, Germany
[3]The University of Texas at Dallas, Richardson, USA
[4]Middlesex University, London,UK

Abstract

Node overlap removal is a necessary step in many scenarios including laying out a graph, or visualizing a tag cloud. Our contribution is a new overlap removal algorithm that iteratively builds a Minimum Spanning Tree on a Delaunay triangulation of the node centers and removes the node overlaps by "growing" the tree. The algorithm is simple to implement, yet it produces high quality layouts. According to our experiments it runs several times faster than the current state-of-the-art methods.

Submitted: December 2016	Reviewed: December 2016	Revised: January 2017	Reviewed: March 2017	Revised: May 2017
Reviewed: June 2017	Revised: June 2017	Accepted: July 2017	Final: July 2017	Published: October 2017
	Article type: Regular paper		Communicated by: Y. Hu and M. Nöllenburg	

E-mail addresses: levnach@microsoft.com (Lev Nachmanson) arlind.nocaj@uni-konstanz.de (Arlind Nocaj) besp@utdallas.edu (Sergey Bereg) L.X.Zhang@mdx.ac.uk (Leishi Zhang) holroyd@microsoft.com (Alexander Holroyd)

1 Introduction

In a drawing of a graph the nodes are typically rendered as shapes of specific sizes. Some graph layout algorithms (such as Sugiyama's scheme [24, 4]) respect the node sizes, while others (such as Multidimensional Scaling [17] and many versions of Force Directed layouts) disregard the node sizes and treat all nodes as points. Algorithms of the later type produce drawings where nodes might overlap and hide important visual attributes. To remedy the problem, an overlap removal algorithm is necessary. The task of an overlap removal algorithm is to remove the overlap while keeping the layout similar to the original one.

We propose a novel overlap removal algorithm which we call *Growing Tree*, or *GTree*. The main idea of GTree is to resolve the overlap by "growing" a special tree which is built on graph nodes. We compared GTree with PRISM [6], which is considered to be the state-of-the-art overlap removal algorithm.

GTree is the default overlap removal algorithm in MSAGL[1], and PRISM is the default in Graphviz [2]. To make a fair comparison we implemented GTree in Graphviz, and PRISM in MSAGL, and then we ran comparisons by using both tools.

2 Related Work

In the extensive literature on overlap removal, apparently the first paper on the problem (which the authors called *layout adjustment*) was by Misue *et al.* [22]. They presented several algorithms including *force scan* where spring forces between nodes are applied in vertical and horizontal scans. The layout adjustment algorithms in [22] aim to preserve the "mental map" of the layout.

A simple solution of the node overlap removal is to scale up the drawing with a minimum scale factor, which works if no two node centers coincide. While the shape of the layout is preserved in this method, it may produce huge drawings.

Marriott *et al.* [21] explored a scaling method with two scale factors, one for x-coordinates and one for y-coordinates. The disadvantage of this method is that it produces drawings with extreme aspect ratios.

Some force-directed methods [5] have been extended to take the node sizes into account [19, 18, 25], but it is difficult to guarantee overlap-free layouts without increasing the repulsive forces excessively. Dwyer et al. [2] show how to avoid node overlaps with Stress Majorization [7]. This method can remove node overlaps during the layout step, but it needs an initial state that is overlap free; sometimes such a state is not available.

In Cluster Busting [20, 8] the nodes are iteratively moved towards the centers of their Voronoi cells. The process has the disadvantage of distributing the nodes uniformly in a given bounding box.

Imamichi et al. [15] approximate the node shapes by circles and minimize a function penalizing the circle overlaps.

[1]https://github.com/Microsoft/automatic-graph-layout
[2]http://www.graphviz.org/

Starting from the center of a node, RWorldle [23] removes the overlaps by discovering the free space around a node by using a spiral curve and then utilizing this space. The approach requires a large number of intersection queries, which are time consuming. This idea is extended by Strobelt et al. [23] to discover available space by scanning the plane with a line or a circle.

Another set of algorithms focuses on the idea of defining pairwise node constraints and translating the nodes to satisfy the constraints [22, 11, 21, 13]. These methods consider horizontal and vertical problems separately, which often leads to a distorted aspect ratio [6]. A Force-transfer-algorithm is introduced by Huang et al. [14]; horizontal and vertical scans of overlapped nodes create forces moving nodes vertically and horizontally; the algorithm takes $\mathcal{O}(n^2)$ steps, where n is the number of the nodes. Gomez et al. [9] develop Mixed Integer Optimization for Layout Arrangement to remove overlaps in a set of rectangles. The paper discusses the quality of the layout, which seems to be high, but not the effectiveness of the method, which relies on a mixed integer problem solver.

The ProjSnippet method [10] generates good quality layouts. The method requires $\mathcal{O}(n^2)$ amount of memory, at least if applied directly as described in the paper, and the usage of a nonlinear problem solver.

In PRISM [6, 12], a Delaunay triangulation on the node centers is used as the starting point of an iterative step. Then a stress model for node overlap removal is built on the edges of the triangulation and the stress function (expressing the energy of the spring system) of the model is minimized

$$\sum_{(i,j)\in E_P} w_{ij}(||x_i - x_j|| - d_{ij})^2, \tag{1}$$

where $d_{ij} = s_{ij}||x_i^0 - x_j^0||$ is the ideal distance for the edge (i, j), (x_i^0, y_i^0) is the current position of the node i, s_{ij} is a scaling factor, $w_{ij} = 1/d_{ij}$ is a weighting factor, and E_P is the set of edges of the Delaunay triangulation or, in general, a proximity graph (V, E_P).

Dwyer et al. [3] reduce the overlap removal to a quadratic problem and solve it efficiently in $\mathcal{O}(n \log n)$ steps. According to Gansner and Hu [6], PRISM is superior to this method in quality and speed.

In general, PRISM is considered the current state-of-the-art. We will compare GTree with PRISM. Like PRISM, GTree also begins by building the Delaunay triangulation as PRISM, but then the algorithms diverge.

We remark that a somewhat similar tree-growing procedure appears in the remarkable sampling algorithm for branched polymers of Kenyon and Winkler [16].

We continue below with the description of GTree.

3 GTree Algorithm

Input to GTree is a set of nodes V, where each node $i \in V$ is represented by an axis-aligned rectangle B_i with the center p_i. We assume that for different

$i, j \in V$ the centers p_i, p_j are different too. If this is not the case, we randomly shift the nodes by tiny offsets. We denote by D a Delaunay triangulation of the set $\{p_i : i \in V\}$, and let E be the set of edges of D.

On a high level, our method proceeds as follows. First we calculate the triangulation D, then we define a cost function on E and build a minimum cost spanning tree on D for this cost function. Finally, we let the tree "grow". The steps are repeated until there are no more overlaps. The last several steps are slightly modified. Now we explain the algorithm in more detail.

First we define cost function c on E, following the definition in PRISM.

Namely, let (i, j) be an edge of E. If rectangles B_i and B_j do not overlap (their interiors do not intersect), then $c(i, j) = dist(B_i, B_j)$, which is the minimum of distances between a point in B_i and a point in B_j. Otherwise, for a real number t let us denote by $B_j(t)$ the rectangle obtained from B_j by shifting it to the new center at $p_i + t(p_j - p_i)$. There is a unique $t_{ij} > 1$ such that the rectangles B_i and $B_j(t_{ij})$ touch each other. Let $s = \|p_j - p_i\|$, where $\|\|$ denotes the Euclidean norm. We set $c(i, j) = -(t_{ij} - 1)s$. One can see, that the cost is negative for an edge connecting overlapped nodes. See Figure 1 for an illustration.

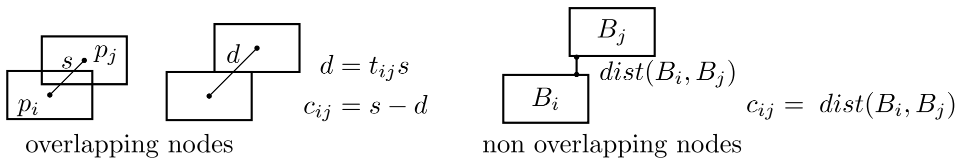


Figure 1: Cost function c_{ij} for edges of the Delaunay triangulation. For overlapping nodes $-c_{ij}$ is equal to the minimal distance that is necessary to shift the boxes along the edge direction so they touch each other.

Now we have a weighted Delaunay graph with cost function c, and we compute a minimum spanning tree T on this graph. In our implementation we use Prim's algorithm to find T.

In the next step, we create a rooted tree from T by selecting a root vertex in T randomly. Then GTree proceeds by growing the rooted tree, similar to the growth of a tree in nature. The tree growing procedure can be described as follows. Let i be a vertex of the rooted tree. For each child j of i we compute the new position of j. If the rectangles of nodes i and j do not overlap, we keep the vector from i to j unchanged. Otherwise, we keep the direction of this vector constant but increase its length, by moving j further away from i, until the rectangles stop overlapping. Vertex i is chosen by the depth-first search algorithm starting from the root. This process is described in Algorithm 1.

The number t_{ij} in line 5 of Algorithm 1 is the same as in the definition of the cost of the edge (i, j) when nodes B_i and B_j overlap, and is 1 otherwise.

The algorithm does not update all positions for the child sub-tree nodes immediately, but updates only the root of the sub-tree. Using the initial positions of a parent and a child, and the new position of the parent, the algorithm

Algorithm 1: Growing T

```
   Input: Current center positions $p$ and root $r$
   Output: New center positions $p'$
1  $p'_r = p_r$
2  GrowAtNode ($r$)
3  function GrowAtNode (i)
4      foreach $j \in Children(i)$ do
5          $p'_j = p'_i + t_{ij}(p_j - p_i)$
6          GrowAtNode (j);
```

obtains the new position of the child in line 5. In total, Algorithm 1 works in $O(|V|)$ steps. The choice of the root of the tree does not matter. Different roots produce the same results modulo a translation of the plane by a vector. Indeed it can be shown that after applying the algorithm, for any $i, j \in V$ the vector $p'_j - p'_i$ is defined uniquely by the path from i to j in T.

While an overlap along any edge of the triangulation exists, we iterate, starting from finding a Delaunay triangulation, then building a minimum spanning tree on it, and finally running Algorithm 1. See Figure 2 for an example.

When there are no overlaps on the edges of the triangulation, as noticed by Gansner and Hu [6], overlaps are still possible. We follow the same idea as PRISM and modify the iteration step. In addition to calculating the Delaunay triangulation we run a sweep-line algorithm to find all overlapping node pairs and augment the Delaunay graph D with each such a pair. As a consequence, the resulting minimum spanning tree contains non-Delaunay edges catching the overlaps, and the rest of the overlaps are removed. This stage usually requires much less time than the previous one.

One can view our GTree algorithm as a minimization of the stress function from Equation (1) when the proximity graph (V, E_p) is a tree.

It is possible to create an example where the algorithm will not remove all overlaps, see Figure 3. However, such examples are extremely rare and have not been seen yet in practice of using MSAGL or in our experiments. MSAGL applies random tiny changes to the initial layout which prevents GTree from cycling.

In the following paragraphs we describe some situations when GTree provably converges.

Observe first that, if all nodes are boxes of the same size with centers belonging to the same straight line, then GTree removes the overlap using just one iteration. Furthermore, in this situation GTree preserves the node order along the line. Example 4 with nodes on two horizontal lines shows that one iteration of GTree is not sufficient.

We show that, for some layouts, GTree converges to overlap-free layouts while preserving some properties of the initial layout.

Theorem 1 *Suppose that*

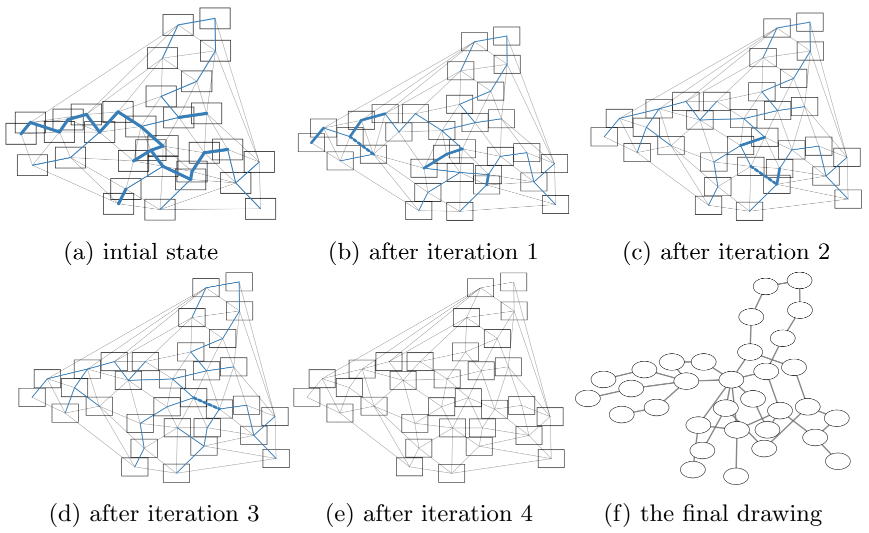


Figure 2: *GTree iterations.* The lines connecting the rectangle centers represent E. The blue edges form tree T. The edges of T connecting overlapped nodes are thick. A few edges of T are in addition dashed: They correspond to new created overlaps. In this run an edge representing overlap, such an edge is thick and blue, is elongated by not more than 1.5 times. Surprisingly, the overlap is removed in four iterations.

1. *the centers of the nodes in V are located on horizontal lines such that the distance between any two lines is at least d, and*
2. *all nodes are closed rectangles $B_i, i = 1, \ldots, n$ having the same width and height, and height less than d.*

Then GTree removes all node overlaps and finds an overlap-free layout of G in at most n iterations, while keeping the node positions on the same horizontal lines.

Proof:

We define a graph F_V for a given set of nodes V and their positions in the plane as follows. The set of vertices of F_V is V. Two nodes i and j from V are connected by an edge if and only if

(i) i and j have their centers on the same horizontal line,

(ii) no other center on the same horizontal line lies between the centers of i and j,

(iii) and the rectangles of i and j intersect each other; $B_i \cap B_j \neq \emptyset$.

It is straightforward to check that F_V is a subgraph of a minimum spanning tree produced by GTree. At each step the number of connected component in

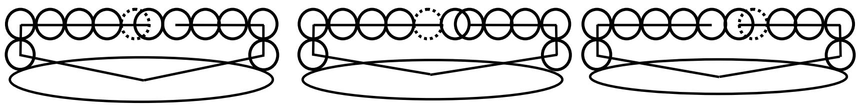

(a) initial state (b) iteration 1 (c) before iteration 2

Figure 3: In this configuration each pair of neighboring ellipses touch each other, except of two that overlap. The solid straight lines display the minimum spanning tree that is used by GTree. The ellipse with the dashed border plays a role of the tree root. By unfortunate choosing the tree edges we can bring the algorithm to cycle. As we see, after applying the algorithm at c) we again arrive to a), the initial state. The cycling will be broken when at least one of the edges edges adjacent to the centers of three bottom ellipses is left out of the tree.

F_V cannot increase. If it remains the same, then after the step there are no overlaps. Since the initial number of components is at most n, this concludes the proof. □

Corollary 2 *The conclusion of Theorem 1 still holds if the widths of the rectangles on the same line are the same, and the projections to y-axis of any two rectangles on different lines are disjoint.*

4 Comparing PRISM and GTree by Measuring Layout Similarity, Quality, and Run Time

Our data includes the same set of graphs that was used by the authors of PRISM to compare it with other algorithms [6]. The set is available in the Graphviz open source package[3]. We also used a small collection of random graphs and a collection of about 10,000 files residing here[4]. For the experiments we use a modified version of Dot, where we can invoke either GTree or Prism for the overlap removal step, and we also used MSAGL, where we implemented PRISM and GTree. MSAGL was used only to obtain the quality measures. We ran the experiments on a PC with Linux, 64bit and an Intel Core i7-2600K CPU@3.40GHz with 16GB RAM.

Some of resulting layouts can be seen in Figures 5,7,8. In Figure 5, we see that the layouts of PRISM have less drawing area but the nodes are uniformly condensed within this area. This tendency can be seen in almost all drawings of PRISM. On the other hand, the layouts produced by GTree usually occupy larger area but the "structure" of the initial layout is preserved better, helping to maintain the "mental map". It can be seen especially on drawings with clusters. The main reason explaining this phenomenon is that the proximity graph in PRISM includes *all* Delaunay edges. Some of these edges belong to the boundary

[3]http://www.graphviz.org
[4]https://github.com/Microsoft/automatic-graph-layout/

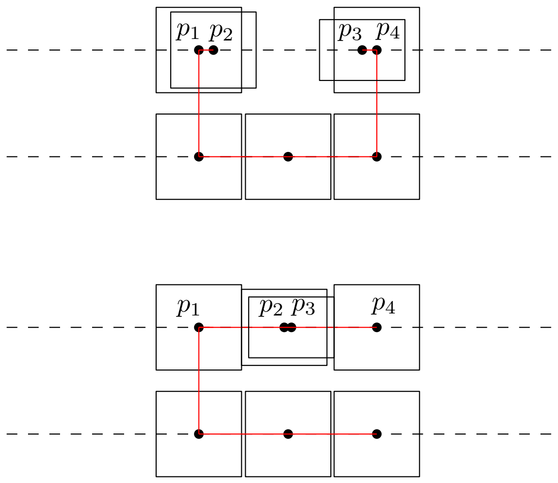


Figure 4: There could be more than one iteration of the algorithm GTree for a layout in two lines. The rectangles B_2 and B_3 are shown using different heights for clarity. The minimum spanning trees of GTree are shown in red.

of the initial layout and, according to the stress function from Equation (1), PRISM attempts to preserve their length and, therefore, the perimeter of the layout. Also Delaunay triangulation may have long edges that do not correspond to an overlap. PRISM again tries to preserve their length in the stress function which constricts overlap removal for other edges. In contrast, the minimum spanning tree used in GTree has fewer those kind of edges.

We consider the *area* of the final layout as one of the quality measures. Usually PRISM produces a smaller area than GTree, see Table 1.

In addition to comparing the areas, we compare some other layout properties. Following Gansner and Hu [6], we look at *edge length dissimilarity*, denoted as σ_{edge}. This measure reflects the relative change of the edge lengths of a Delaunay Triangulation on the node centers of the original layout.

The other measure, which is denoted by σ_{disp}, is the Procrustean similarity [1]. It shows how close the transformation of the original graph is to a combination of a scale, a rotation, and a shift transformation. PRISM and GTree perform similar in the last two measures as Table 1 shows.

To distinguish the methods further, we measure the change in the set of k closest neighbors of the nodes. Namely, let $p_1, \dots, p_n$ be the positions of the node centers, and let k be an integer such that $0 < k \leq n$. Let $I = \{1, \dots, n\}$ be the set of node indices. For each $i \in I$ we define $N_k(i) \subset I \setminus \{i\}$, such that $|N_k(p, i)| = k$, and for every $j \in I \setminus N_k(p, i)$ and for every $j' \in N_k(p, i)$ holds $\|p_j - p_i\| \geq \|p_{j'} - p_i\|$. In other words, $N_k(p, i)$ represents a set of k closest neighbors of i, excluding i. Let $p'_1, \dots, p'_n$ be transformed node centers. To see how much the layout is distorted nearby node i, we intersect $N_k(p, i)$ and $N_k(p', i)$. We measure the distortion as $(k - m)^2$, where m is the number of

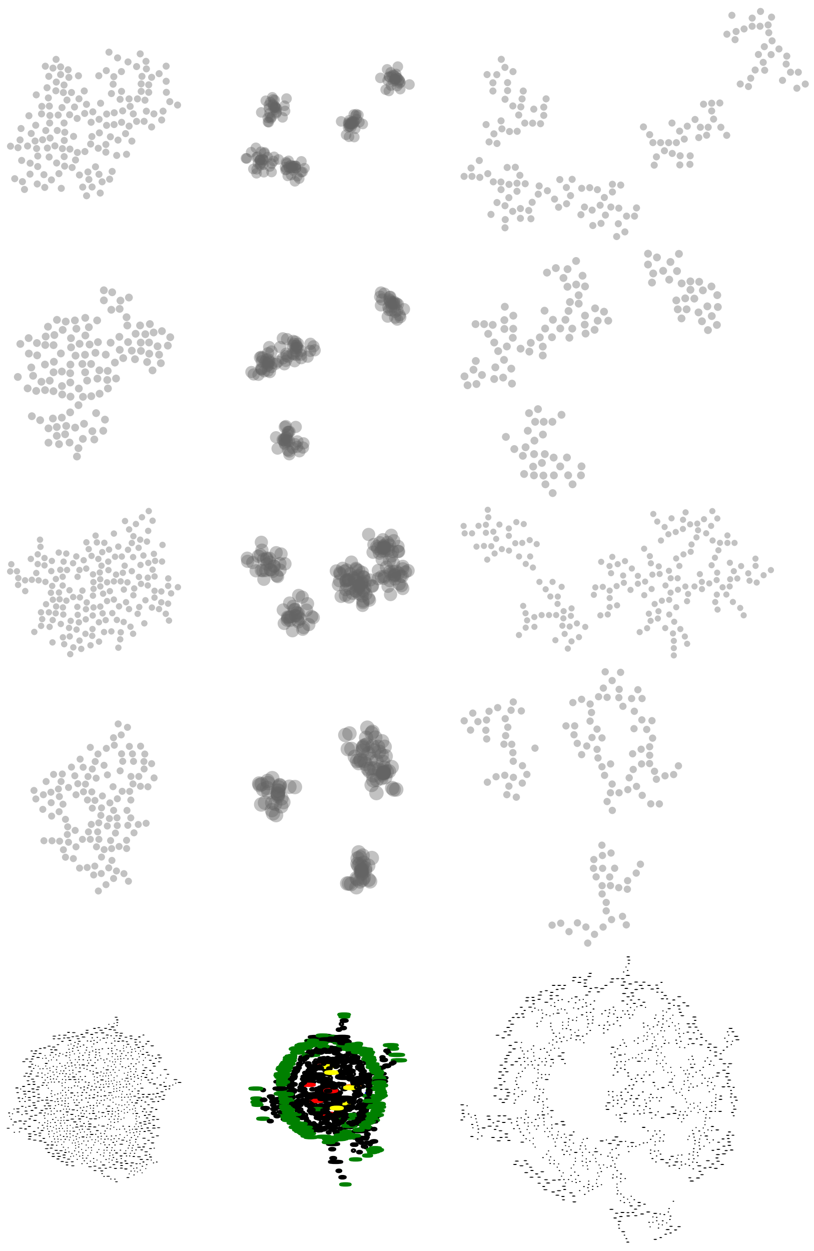


Figure 5: Comparison between PRISM, original, and GTree layouts. In four top rows the initial layouts were generated randomly. At the bottom are the drawings of nodes of graph "root" which was initially laid out by the Multi Dimensional Scaling algorithm of MSAGL. In our opinion, the initial structure is more preserved in the right column, containing the results of GTree.

elements in the intersection. If a node preserves its k closest neighbors then the distortion is zero.

Our experiments for k from 8 to 12 show that under this measure GTree produced a smaller error, showing less distortion, on 8 graphs from 14, and on the rest PRISM produced a better result, see Table 2. GTree produced smaller error on all small random graphs.

Table 1: Similarity to the initial layout (left) and number of iterations for different graph sizes and different initialization methods (right). PR stands for PRISM (σ_{edge} and σ_{disp}) and the final layout area.

Graph	PR	GTree	PR	GTree	PR	GTree
dpd	0.34	0.28	0.37	0.36	0.82	0.84
unix	0.22	0.19	0.24	0.20	2.38	2.38
rowe	0.29	0.26	0.23	0.24	0.68	0.73
size	0.39	0.37	0.24	0.26	1.09	1.28
ngk10_4	0.30	0.30	0.27	0.30	0.00	0.00
NaN	0.56	0.44	0.73	0.51	4.03	4.34
b124	0.55	0.53	0.97	0.83	5.52	6.22
b143	0.67	0.70	1.12	0.93	3.62	3.88
mode	0.54	0.50	0.59	0.53	1.53	2.29
b102	0.71	0.77	1.43	1.27	4.50	6.62
xx	0.75	0.70	1.65	1.42	6.21	9.57
root	1.09	1.19	2.89	2.45	34.58	91.87
badvoro	0.88	0.92	2.27	2.42	25.68	47.43
b100	0.84	0.98	3.08	3.14	20.64	37.38

init. layout:			**neato**		**SFDP**	
Graph	$\lvert V \rvert$	$\lvert E \rvert$	PR	GTree	PR	GTree
dpd	36	108	4	7	3	6
unix	41	49	3	4	12	5
rowe	43	68	5	4	13	7
size	47	55	7	3	9	5
ngk10_4	50	100	6	3	14	7
NaN	76	121	8	3	24	6
b124	79	281	14	4	30	12
b143	135	366	21	6	37	12
mode	213	269	37	8	11	6
b102	302	611	60	24	113	19
xx	302	611	83	18	50	19
root	1054	1083	95	18	99	22
badvoro	1235	1616	40	20	50	23
b100	1463	5806	80	24	136	28

Table 2: k closest neighbors error, the Multi Dimensional Scaling algorithm of MSAGL was used for the initial layout. PR stands for PRISM.

	$k=8$		$k=9$		$k=10$		$k=11$		$k=12$	
Graph	PR	GTree	PR	GTree	PR	GTree	PR	GTree	PR	GTree
dpd	7.75	6.06	9.61	7.36	9.5	8	10.14	8.5	9.97	7.64
unix	8.56	7.05	10.51	8.8	10.95	10.02	11.66	10.54	13	11.41
rowe	6.28	8.09	7.09	9.95	7.49	10.49	9.12	11.4	11.05	12.51
size	4.68	6.09	5.47	6.47	6.28	7.57	6.89	8.13	8.26	10.02
ngk10_4	6.76	7.4	7.52	9.26	8.28	11.38	10.72	13.74	11.92	14.66
NaN	11.83	8.95	14.46	11.5	17.32	13.88	19.88	16.37	22.17	19.7
b124	11.03	11.44	13.22	13.56	14.76	15.54	15.91	17.32	18.23	20.04
b143	13.49	12.39	16.31	14.99	19.49	17.93	23.11	21.04	26.53	24.43
mode	16.91	11.46	20.58	13.95	24.68	16.85	29.54	19.92	34.48	22.56
b102	15.99	14.62	19.61	18.78	23.38	22.77	27.28	26.77	32.15	31.45
xx	15.68	15.65	19.01	19.45	23.05	23.37	26.98	27.35	31.29	32.47
root	17.09	15.7	20.89	19.36	25.48	23.3	30.48	27.66	35.74	32.83
badvoro	16.18	15.15	20.16	18.98	24.37	23.28	29.18	28.03	34.29	33.29
b100	18	19.25	22.11	23.65	26.79	28.69	32.03	34.46	37.44	40.5

We ran tests on the graphs from a subdirectory of MSAGL called AndyFiles[5]. Let us call this set of graphs collection A. Each graph from A represents the control flow of a method from a version of the .NET framework. A contains 10077 graphs. The graph sizes do not exceed several thousands. We used the Multi Dimensional Scaling algorithms of MSAGL for the initial layout in this test. The results of the run are summarized in Table 3.

[5]https://github.com/Microsoft/automatic-graph-layout/

Table 3: Statistics on collection A. Here k-cn stands for k-closest neighbors, and "iters" stands for the number of iterations. Each cell contains the number of graphs for the measure on which the method performed better. We can see that PRISM produced a layout of smaller area than the one of GTree on 8498 graph, against 1579 graphs where GTree required less area. From the other side, GTree gives better results on all other measures. The columns of k-cn and "iters" do not sum to 10077, the number of graphs in A, because some of the results were equal for PRISM and GTree.

Method	k-cn	σ_{edge}	σ_{disp}	area	iters	time
PRISM	3237	4741	4114	8498	46	7
GTree	4088	5336	5963	1579	9986	10070

Runtime Comparison

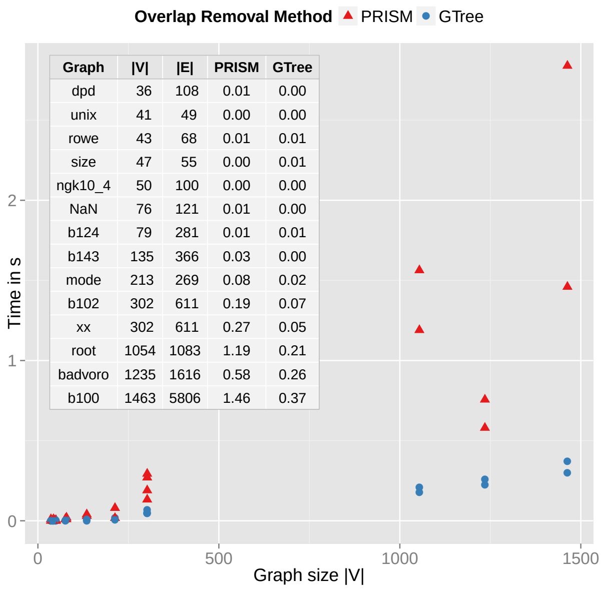
Both methods remove the overlap iteratively using the proximity graph. However, while PRISM needs $\mathcal{O}(|V| \cdot \sqrt{|V|})$ time to solve the stress model, GTree needs only $\mathcal{O}(|V|)$ time per iteration with the growing tree procedure. Therefore, GTree is asymptotically faster in a single iteration. In addition, as Table 1 (right) shows, GTree usually needs fewer iterations than PRISM, especially on larger graphs. The overall runtime can be seen in Figure 6.

In addition, we ran experiments on large random graphs. For $n = 10000$ and $n = 100000$ we created a set of n circles with a radius 66 and a random chosen center inside of a square with a side of length 1000. Then we removed the overlaps by running PRISM and GTree and repeated this experiment 10 times. For $n = 10000$ PRISM required from 50 to 65 iterations for, while GTree required from 19 to 24. For $n = 100000$ PRISM required between 77 and 100, while GTree required from 25 to 29 iterations. It shows that GTree outperforms PRISM on larger graphs.

In Figure 7 we experiment with the way we expand the edges. Instead of the formula $p'_j = p'_i + t_{ij}(p_j - p_i)$, which resolves the overlap between the nodes i and j immediately, we use the update $p'_j = p'_i + \min(t_{ij}, 1.5)(p_j - p_i)$. As a result, the algorithm runs a little bit slower but produces layouts with smaller area.

5 Conclusion & Future Work

We proposed a new overlap removal algorithm that uses the minimum spanning tree. We compared the algorithm with PRISM, which is the current state-of-the-art method of overlap removal. We found out that GTree is asymptotically faster, and in general faster than PRISM. We applied four quality measures to the results. GTree was better in every measure except one; GTree drawings usually require more area. GTree is much simpler and easier to implement than PRISM. We hope that GTree will be used widely.



Graph	|V|	|E|	PRISM	GTree
dpd	36	108	0.01	0.00
unix	41	49	0.00	0.00
rowe	43	68	0.01	0.01
size	47	55	0.00	0.01
ngk10_4	50	100	0.00	0.00
NaN	76	121	0.01	0.00
b124	79	281	0.01	0.01
b143	135	366	0.03	0.00
mode	213	269	0.08	0.02
b102	302	611	0.19	0.07
xx	302	611	0.27	0.05
root	1054	1083	1.19	0.21
badvoro	1235	1616	0.58	0.26
b100	1463	5806	1.46	0.37

Figure 6: Running times for PRISM and GTree.

Although we introduced our approach in the context of graph visualization, one can notice that we never used the edges of the original graph. GTree can be used for any other purpose where overlap needs to be resolved while maintaining the initial layout.

Finding a measure of how well an overlap removal algorithm preserves clusters of the initial layout seems to be an interesting challenge.

One of the directions for future work could be exploring other families of proximity graphs (V, E_P) and analyze how they affect the final layout.

One issue concerns breaking ties between edges of equal cost when constructing the spanning tree. In particular, the example in Figure 3 shows that if ties are broken arbitrarily then the algorithm may not converge (at least in contrived instances). To address this, we propose breaking ties at random: at each step, any set of edges with equal costs are assigned a uniformly random order, which is used to select edges for the spanning tree. With this method, it is straightforward to show that the example in Figure 3 converges in expected number of steps $\Theta(|V|)$ (although we believe that typical instances converge much faster).

Conjecture 3 *Suppose that, in choosing the spanning tree in the GTree algorithm, we break ties between edges of equal cost uniformly at random. Then the algorithm stops with probability one.*

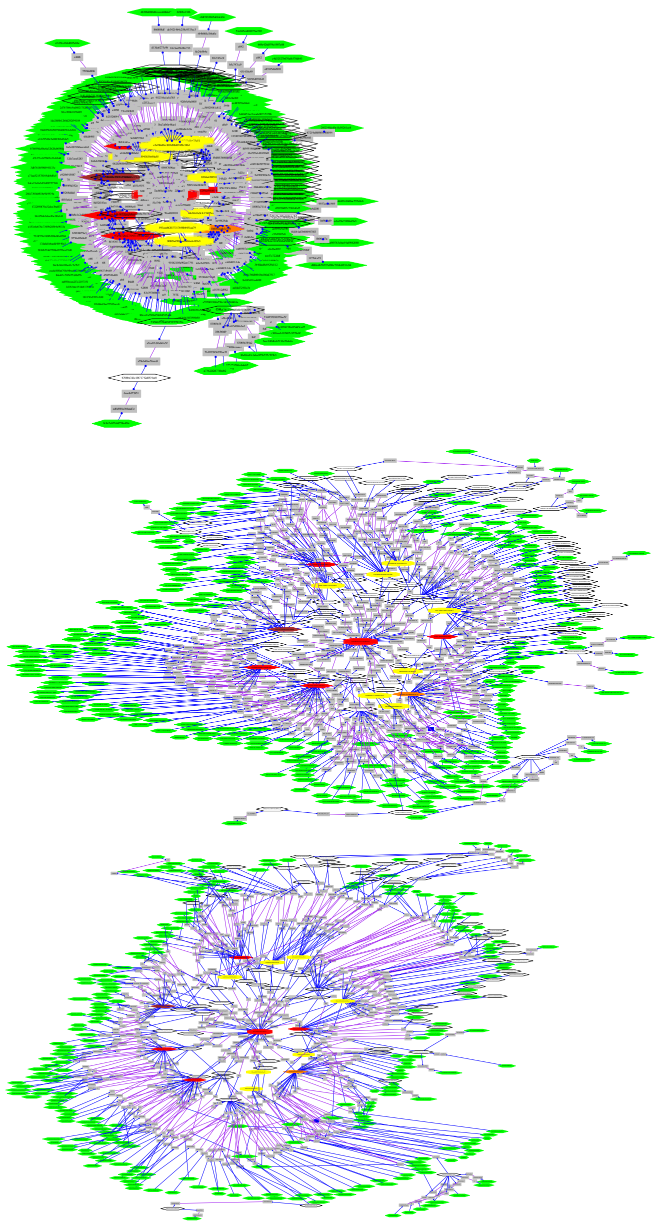


Figure 7: root graph with 1054 nodes and 1083 edges. (a) initial layout with NEATO, (b) applying PRISM, (c) applying GTree.

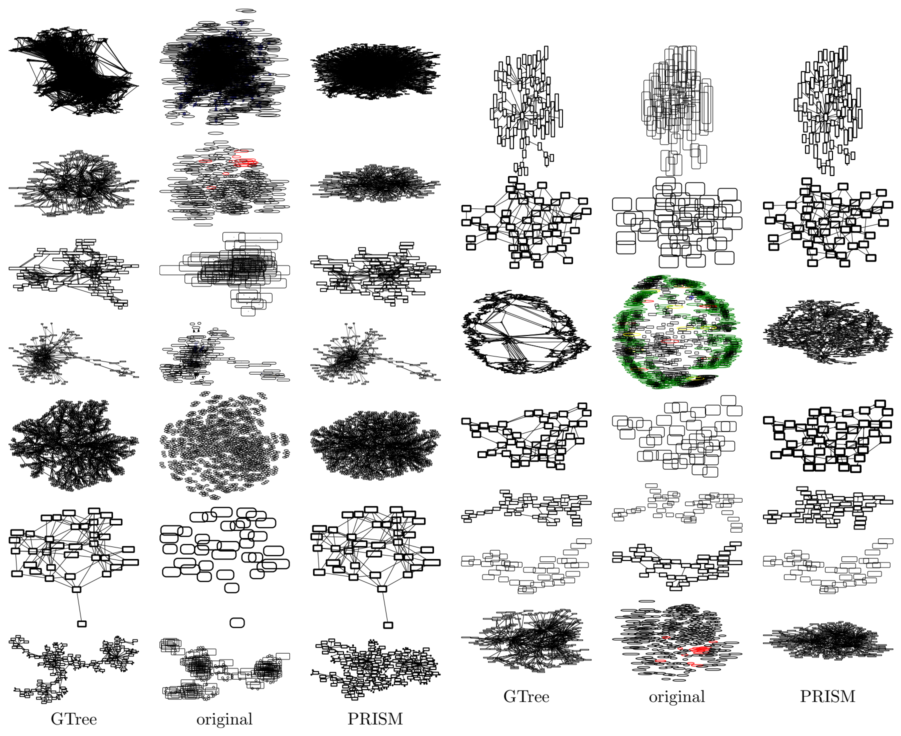


Figure 8: Results for GTree and PRISM initialized with SFDP. From top to bottom and left to right: b100, b102, b124, b143, badvoro, dpd, mode, - NaN, ngk10_4, root, rowe, size, unix, and xx. To make the original drawings more readable they have been changed; In most cases the nodes were diminished and the edges removed. The drawings were scaled differently.

References

[1] I. Borg and P. Groenen. *Modern multidimensional scaling: Theory and applications.* Springer, 2005. `doi:10.1007/0-387-28981-X`.

[2] T. Dwyer, Y. Koren, and K. Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Trans. Vis. Comput. Graph.*, 12(5):821–828, 2006. `doi:10.1109/TVCG.2006.156`.

[3] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *Graph Drawing*, pages 153–164. Springer, 2006.

[4] C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. In V. Estivill-Castro, editor, *ACSC*, volume 26 of *CRPIT*, pages 369–376. Australian Computer Society, 2004.

[5] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991. `doi:10.1002/spe.4380211102`.

[6] E. R. Gansner and Y. Hu. Efficient, proximity-preserving node overlap removal. *J. Graph Algorithms Appl.*, 14(1):53–74, 2010. `doi:10.7155/jgaa.00198`.

[7] E. R. Gansner, Y. Koren, and S. C. North. Graph drawing by stress majorization. In J. Pach, editor, *Graph Drawing*, volume 3383 of *LNCS*, pages 239–250. Springer, 2004. `doi:10.1007/978-3-540-31843-9_25`.

[8] E. R. Gansner and S. C. North. Improved force-directed layouts. In S. Whitesides, editor, *Graph Drawing*, volume 1547 of *LNCS*, pages 364–373. Springer, 1998. `doi:10.1007/3-540-37623-2_28`.

[9] E. Gomez-Nieto, W. Casaca, L. G. Nonato, and G. Taubin. Mixed integer optimization for layout arrangement. In *SIBGRAPI 2013*, pages 115–122. IEEE, 2013. `doi:10.1109/SIBGRAPI.2013.25`.

[10] E. Gomez-Nieto, F. San Roman, P. Pagliosa, W. Casaca, E. Helou, M. Ferreira de Oliveira, and L. Nonato. Similarity preserving snippet-based visualization of web search results. *IEEE Trans. Vis. Comput. Graph.*, 20(3):457–470, 2014. `doi:10.1109/TVCG.2013.242`.

[11] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara. A layout adjustment problem for disjoint rectangles preserving orthogonal order. *Systems and Computers in Japan*, 33(2):31–42, 2002. `doi:10.1007/3-540-37623-2_14`.

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

[13] X. Huang and W. Lai. Force-transfer: A new approach to removing overlapping nodes in graph layout. In M. J. Oudshoorn, editor, *ACSC*, volume 16 of *CRPIT*, pages 349–358. Australian Computer Society, 2003.

[14] X. Huang, W. Lai, A. Sajeev, and J. Gao. A new algorithm for removing node overlapping in graph visualization. *Information Sciences*, 177(14):2821–2844, 2007. `doi:10.1016/j.ins.2007.02.016`.

[15] T. Imamichi, Y. Arahori, J. Gim, S.-H. Hong, and H. Nagamochi. Removing node overlaps using multi-sphere scheme. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing*, volume 5417 of *LNCS*, pages 296–301. Springer, 2008. `doi:10.1007/978-3-642-00219-9_28`.

[16] R. Kenyon and P. Winkler. Branched polymers. *American Mathematical Monthly*, 116(7):612–628, 2009. `doi:10.4169/193009709x458582`.

[17] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. `doi:10.1007/BF02289565`.

[18] W. Li, P. Eades, and N. S. Nikolov. Using spring algorithms to remove node overlapping. In S.-H. Hong, editor, *APVIS*, volume 45 of *CRPIT*, pages 131–140. Australian Computer Society, 2005.

[19] C.-C. Lin, H.-C. Yen, and J.-H. Chuang. Drawing graphs with nonuniform nodes using potential fields. *J. Vis. Lang. Comput.*, 20(6):385–402, 2009. `doi:10.1016/j.jvlc.2009.04.006`.

[20] K. A. Lyons, H. Meijer, and D. Rappaport. Algorithms for cluster busting in anchored graph drawing. *J. Graph Algorithms Appl.*, 2(1), 1998. `doi:10.7155/jgaa.00004`.

[21] 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. `doi:10.1023/A:1022371615202`.

[22] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995. `doi:10.1006/jvlc.1995.1010`.

[23] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen. Rolled-out wordles: A heuristic method for overlap removal of 2D data representatives. In *Computer Graphics Forum*, volume 31, pages 1135–1144. Wiley Online Library, 2012. `doi:10.1111/j.1467-8659.2012.03106.x`.

[24] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. `doi:10.1109/TSMC.1981.4308636`.

[25] X. Wang and I. Miyamoto. Generating customized layouts. In F.-J. Brandenburg, editor, *Graph Drawing*, volume 1027 of *LNCS*, pages 504–515. Springer, 1995. `doi:10.1007/BFb0021835`.