# Dynamic Force-Directed Graph Layout for Software Visualization

Ivan Iliev[1], Haralambi Haralambiev[1], Milena Lazarova[2], Stanimir Boychev[1]

*Abstract* – **Drawing graphs of software systems in a meaningful way is both computationally and aesthetically problematic. The paper presents a graph layout that improves the existing methods making them more suitable for use in the area of software visualization and comprehension. The suggested graph layout is an extended graph drawing aimed at better computational cost and aesthetic results of visualization of complex software systems.**

*Keywords* – **Software visualization, dynamic, graph, layout, drawing, force-directed.**

## I. INTRODUCTION

Graphs have many applications in different areas of computer science. They are used to represent networks of communication, data organization, computational devices and flow, as well as many others. Two of the main objectives of software visualization are to ease the process of understanding an unfamiliar software system and to allow visual identification of anomalies within the software structure and its evolution [1]. The nature of graphs as an abstract model lends itself to software visualization through being able to show multiple objects and relations as vertexes and edges of a graph.

There are two general types of graph layouts algorithms - static and dynamic. The static methods address the problem of constructing a one-time graph drawing. Dynamic methods are concerned with preserving the user's mental map so that different drawings maintain the overall structure of the graph and only reflect individual changes. In order to support visualization of the evolution ofsoftware systems, the obvious choice are dynamic algorithms because they preserve the mental map.

## II. RELATED WORK

Frishman and Tal [2] focus on the dynamicity of a layout by using pinning weights. These weights are assigned to vertices between consecutive layouts based on their distance to modification. Dynamicity, however, is only a part of the graph drawing problem. The most popular solutions for drawing graphs are force-directed ones.

Force-directed methods model the vertices as physical bodies with different forces between them. These kinds of algorithms are based on the effect of such forces acting on an initial graph for a fixed number of iterations or until an energy function is minimized. There are two parts of a force-directed layout – a force model and a technique for finding minimum energy configurations.

One of the earliest force models for graph drawing was proposed by Eades [3] and is widely used today. Eades' model is based on the mechanical model which presents graph vertices as rings and graph edges as springs connecting the vertices. When too far the springs apply attraction to the rings bringing them closer together and when too close repulsion is exerted pushing the rings apart. There have been several modifications and extensions to Eades' force model most notably by Fruchterman & Rengold [4], Yifan Hu [5], whose work uses that of Fruchterman & Reingold, and Kamada & Kawai [6]. Fruchterman & Reingold use Hooke's law to model the spring forces and apply repulsion between all pairs of vertices where as attraction is only applied between neighbours leaving us with an overall time complexity of $O(|V|^2 + |E|)$ per iteration. They have also used a simulated annealing optimization technique from Davidson & Harel [7]. Hu [4] proposed several modifications to Fruchterman & Reingold's algorithm which lead to layouts of better quality and improve the computational efficiency significantly by reducing repulsion calculations to $O(|V|\log|V|)$ per iteration through the use of a QuadTree/OcTree spatial decomposition data structure. Kamada & Kawai [6] on the other hand require that the graph theoretical distance between all pairs of vertices is computed and forces along with the energy model are based on this distance. The overall computational complexity of Kamada & Kawai's algorithm is $O(|V|.|E|)$ per iteration and an $O(|V|^2)$ memory complexity.

A big part of drawing a graph for use in software visualization is overlap removal. In order for information to be visually comprehensive the graph drawing should not be cluttered and overlaps between vertex bounding regions should be avoided. There are two ways to remove overlaps in a force-directed algorithm. The first one is to modify the force model to rapidly repulse vertices whose bounding regions are overlapping and thus produce a layout without overlaps. Such a method has been suggested by Li, Eades and Nikolov [8]. Gansner and Hu [9] suggest a second way, in which a post-processing step is used after the layout algorithm has finished, computing and altering the final drawing so that overlaps are avoided.

In addition to vertex overlap removal, edge bundling algorithms are used to reduce clutter and thus achieve a more visually comprehensive drawing. Edge bundling algorithms group edges with similar paths into bundles improving overall

[1]Ivan Iliev, Haralambi Haralambiev and Stanimir Boychev are with the Applied Research and Development Center at Musala Soft, 36 Dragan Tsankov blvd, 1057 Sofia, Bulgaria
E-mails:
{ivan.iliev,haralambi.haralambiev,stanimir.boychev}@musala.com

[2]Milena Lazarova is with theComputer Systems Department at Technical University of Sofia, 8 Kliment Ohridski blvd, 1756 Sofia, Bulgaria, E-mail: milaz@tu-sofia.bg

visibility in the drawing by making multiple edges look like a single one – a bundle. An algorithm for edge bundling is suggested by Danny Holten and Jarke J. van Wijk [10].

This paper focuses on drawing undirected straight-line edge graphs using iterative force-directed methods combined with non-iterative ones,based on the research in [4] and [5]. The idea of pinning weights is adopted to support dynamicity. The algorithm introduces semantic clustering and an optimization of per iteration computational cost of the layout, making it more suitable for use in software visualization. The post-processing overlap removal based on a proximity stress model ([8]) is used to ensure the preservation of the mental map after each removal step. The suggested algorithm can also be applied for three-dimensional visualization.

## III. Aesthetic Goals

Several drawing constraints must be chosen for the algorithm in order to achieve good layout quality for software visualization and comprehension. After reviewing and experimenting with different layouts the following aesthetic criteria [11] are settled on:

- **Semantic clustering of vertices** – due to the hierarchic nature of a software system the graph is recursively divided into clusters such that all vertices with the same parent belong in the same cluster. This allows for easy identification of software components based on their position in the graph.
- **Dynamicity** – the overall structure of the graph does not change with each consecutive drawing. Different revisions of a software system can be drawn consecutively while preserving the user's mental map.
- **Overlap removal** – all overlaps between vertex bounding regions should be removed by scaling the entire drawing so that the initial graph structure is fully preserved.
- **Edge bundling –** edges sharing similar paths should be grouped together into bundles.

## IV. Layout Algorithm

### *A. Initial Positioning*

In order to begin the layout process initial positions must be set for all vertices. This is currently done by assigning random coordinates within a rectangle whose width and height are equal to the sum of all vertices' widths and heights. In 3D, an analogous operation is performed with a rectangular cuboid. If a previous layout has been executed, all vertices that have already been positioned by it keep their coordinates and only the non-positioned vertices' coordinates are randomized.

### *B.Clustering*

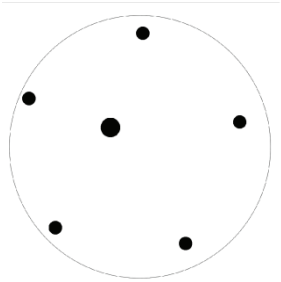

Fig.1.A circular cluster with its smallest enclosing circle

Before the force-directed iterative process can begin all vertices are ordered in circles/spheres around their respective parents, forming clusters.The smallest enclosing circle [12]/sphere [13] is calculated for each cluster (Fig.1). Each vertex, regardless of shape, is assigned an enclosing circle/sphere. The algorithm used to position the vertices in a circle is based on a simple subdivision of the circle into equally sized sectors. All child vertices are sorted by their radius in ascending order and divided into groups by certain criteria. For example, for object-oriented languages the access modifier type (private, protected, public) is a good choice. The algorithm is applied to each group. At first as many vertices as there is room for are positioned on a circle with the smallest possible radius so that there are no overlaps. After the initial vertices are positioned the radius is increased and part of the remaining vertices is positioned again. The process continues until all vertices from this group are properly ordered and then moves on to the next group. As a result, an even distribution of vertices on concentric circles around their parent as well as a clear separation of children by access type is achieved (Fig. 2). In three dimensions the process is analogous with circles replaced by spheres. A modification of Saff and Kuijlaars [14] algorithm is used for distributing points on a sphere using a golden section spiral. The ordering step is applied once at the beginning and once at the end of the layout process to initially calculate bounding circles for all vertices/clusters and to move child vertices to their final positions respectively.

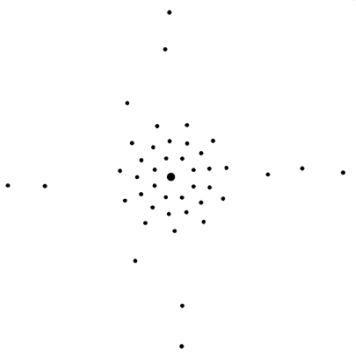

Fig.2. Methods in concentric circles around their parent class. Each method is positioned in a circle with radius based on its access modifier type (private, protected, public)

*C. Dynamicity*

In order to support dynamicity in the suggested layout pinning weights are assigned to all clusters in the following way:

- If the cluster's parent vertex is new (not existing in a previous layout) or has been removed (not existing in the current layout), the cluster's pinning weight is set to 1 allowing its free movement.
- If the cluster's parent vertex has received coordinates from the previous layout and has persisted, its pinning weight is set to 0 prohibiting further movement.

The assignment of pinning weights in that manner leads to the preservation of the mental map between consecutive drawings.

*D. Force-directed iterative process*

Since child vertices will be recursivelylocated around parent ones, only top-level vertices need to be positioned initially (i.e. ones without parents themselves). Therefore, the force-directed iterative layout is applied only for the top-level clusters reducing the overall complexity of the algorithm based on their number. In software systems with many top-level clusters the speed up will not be substantial but still significant enough. Before beginning the iterative process all edges between top-level clusters are counted and the result is stored for each pair. The Fruchterman & Reingold force model is used as modified by Hu in [4] combined with the adaptive step length optimization for simulated annealing. The following actions are performed in each iteration:

1. For each top-level cluster the QuadTree/OcTree is recursively opened.
2. Repulsive forces are calculated between the current top-level cluster and clusters contained within tree nodes "close enough" to it. If the tree node is "far" from the current cluster, repulsion is applied based on the distance between the two. The criteria for "close enough" and "far" are the same as defined in [4] except for the case when there is an overlap in the bounding areas of the current cluster and tree node. If such an overlap exists the tree node is considered "close enough" regardless of the other criteria. Repulsive displacements are stored for each cluster's parent vertex based on the forces calculated.
3. Attractive forces are calculated between each pair of clusters with a positive calculated edge count. The power of the attractive force is based on the number of edges between the two clusters. Attractive displacements are stored for each cluster's parent vertex based on the calculated force.
4. Movements are performed for each vertex based on its stored displacements. Each movement is limited by the pinning weight of a vertex and by the current temperature of the layout used in the simulated annealing optimization. The coordinates for the centres of each bounding circle of top-level clusters are updated with the vector between the cluster's parent vertex previous and new positions.
5. An adaptive cooling step is executed which reduces or increases the current layout temperature.

*E. Post-processing*

The iterative process ends when a convergence criterion is satisfied or a maximum number of iterations are exceeded. The layout process finishes after an overlap removal post-processing step and an edge bundling step are performed. Snapshot of the edge bundling effect is shown in Fig. 3. A full layout of a software system is depicted in Fig. 4.Pseudo code for the entire layout process in 2D is given in Code. 1 and Code. 2.

```
layoutGraph(Graph):
totalWidth = sum(Graph.Node.width)
totalHeight = sum(Graph.Node.height)
foreachNodeinGraph do
    Node.X = random()*totalWidth
    Node.Y = random()*totalHeight
Clusters = buildClusters()
foreachClusterinClusters do
    calculateBoundingCircle(Cluster)
    positionChildren(Cluster)
setPinningWeights()
while not converged and currentIteration<
MAX_ITERATIONSdo
    calculateDisplacements()
    performMovements()
    cool()
    currentIteration++
foreachClusterinClusters do
    positionChildren(Cluster)
removeOverlaps()
bundleEdges()
```

```
cool(T):
reduce or increase the current layout temperature.
```

Code.1. Pseudo code for the layout process and cooling function

## V. RESULTS

With the circular/spherical arrangement of vertices into clusters and having to only run the force-directed algorithm on top-level ones, the time complexity of each iteration is O(|T|log|T| + |TE|) where T is the set of top-level vertices and TE is the set of edges between them. The arrangement step takes O(|V|) time to complete. The most computationally expensive part of the layout is the force-directed step. Usually in most graphs of software systems the ratio |T|/|V| is quite small due to the low number of top-level vertices which results in a lower complexity per iteration and reduces the computational time of the layout significantly.

```
calculateDisplacements(Graph, QuadTree):
foreachNode in Graph do
      Q = QuadTree.ROOT
while not empty(Q) do
   TreeNode = dequeue(Q)
   if far(Node,TreeNode) then
      calculate repulsive displacements between Node
      and TreeNode's centre of gravity
   else if leaf(TreeNode) then
      foreachContainedNodein
      containedNodes(TreeNode)
       calculate repulsive displacements  between
      Nodeand ContainedNode
   else
      enqueue(TreeNode.Children)
foreachEdge between Clusters do
   calculate attractive displacements between Edge.From
   and Edge.To based on Edge.Count
-----------------------------------------------------
performMovements():
foreachClusterinClusters do
   Displ = getDisplacement(Cluster.Parent)
   WPin  = getPinningWeight(Cluster.Parent)
   LT = getLayoutTemperature()
   move(Cluster.Parent, min(Displ,  WPin*LT))
   move(Cluster.Centre, min(Displ,  WPin*LT))
```

Code.2.Pseudo code for calculating displacements and moving vertices based on them

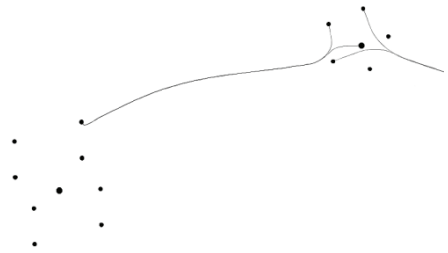

Fig.3. Edge bundling effects on edges – edges sharing a common path are grouped.

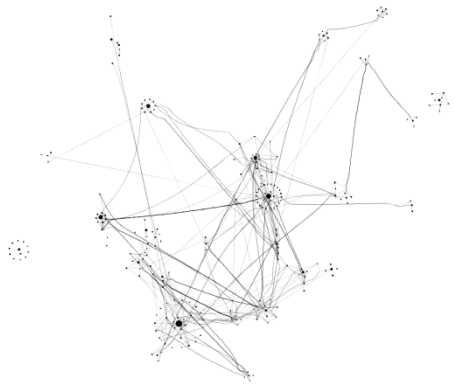

Fig.4. An overview of the Pygmy project graph (http://pygmy-httpd.sourceforge.net) laid out using this algorithm showing the effects of edge bundling and node clustering.

## VI. Conclusions and Future Work

Drawing graphs of software systems to ease program comprehension is an open problem without a definitive solution. An algorithm suited especially for the purpose of drawing such graphs in an efficient, useful and aesthetically pleasing manner is suggested and described in the paper.

Further investigation and improvement of the algorithm will be based on using an alternative metaheuristic instead of simulated annealing for solving the force-directed layout optimization problem such as genetic algorithm or ant colony optimization. Moreover, the presented algorithm could be measured against other layout algorithms on certain characteristics - performance, aesthetics, etc.

## Acknowledgement

This work was partially supported by the Bulgarian National Science Research Fund through contract ДМУ 02/18 - 2009.

## References

[1] S. Diehl, "Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software", Springer, 2007.

[2] Y. Frishman, A. Tal, "Online Dynamic Graph Drawing", Proc. Eurographics/IEEE VGTC Symp. Visualization (EuroVis '07), pp. 75-82, 2007.

[3] P. Eades, "A Heuristic for Graph Drawing", Congressus Numerantium, vol. 42, pp. 149-160, 1984.

[4] T. Fruchterman, E. M. Reingold, "Graph Drawing by Force-Directed Placement", Software Practice and Experience, vol. 21, pp. 1129-1164, 1991.

[5] Y. F. Hu, "Efficient and High Quality Force-Directed Graph Drawing", The Mathematica Journal, vol. 10, pp. 37-71, 2005

[6] T. Kamada, S. Kawai, "An Algorithm for Drawing General Undirected Graphs", Information Processing Letters, vol. 31, pp. 7-15, 1989.

[7] R. Davidson, D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", ACM Trans. Graph., vol. 15, iss. 4, pp. 301-331, 1996.

[8] W. Li., P. Eades, N, Nikolov, "Using Spring Algorithms to Remove Node Overlapping", Proc. of the 2005 Asia-Pacific Symposium on Information Visualisation, Vol.45 (APVis '05), pp. 131-140, 2005.

[9] E. Gansner, Y. Hu, "Efficient Node Overlap Removal Using a Proximity Stress Model", In Graph Drawing, Ioannis G. Tollis and Maurizio Patrignani (Eds.). Lecture Notes in Computer Science, Vol. 5417. Springer-Verlag, Berlin, Heidelberg, pp. 206-217, 2009.

[10] D. Holten, J, van Wijk, "Force-Directed Edge Bundling for Graph Visualization", Proc. of 11th Eurographics/IEEE-VGTC Symposium on Visualization (Computer Graphics Forum; Proceedings of EuroVis 2009), pp. 983 - 990, 2009

[11] G. Battista, P. Eades, R. Tamassia, I.G. Tollis, "Graph Drawing - Algorithms for the Visualization of Graphs", Prentice, 1999.

[12] F. Nielsen, R. Nock, "Approximating Smallest Enclosing Disks", Proc. of 16th Canadian Conference on Computational Geometry (CCCG), pp. 124–127, 2004.

[13] F. Nielsen, R. Nock, "Approximating Smallest Enclosing Balls", Proc. of International Conference on Computational Science and Its Applications (ICCSA)", Springer, Lecture Notes in Computer Science, vol. 3045, pp. 147-157, 2004.

[14] E. Saff, A. Kuijlaars, "Distributing Many Points on a Sphere", The Mathematical Intelligencer, Vol. 19, No. 1, pp. 5-11, 1997.