HD-GraphViz : Highly Distributed Graph Visualization on Tiled Displays

Sangwon Chae*
University of California, Irvine
Irvine, CA, U.S.A.
chaes@uci.edu

Aditi Majumder
University of California, Irvine
Irvine, CA, U.S.A.
majumder@ics.uci.edu

M. Gopi
University of California, Irvine
Irvine, CA, U.S.A.
gopi@ics.uci.edu

ABSTRACT

In this paper, we propose a distributed approach for visualizing graph datasets using distributed force-directed layout algorithm on multiple displays each of which is controlled by a compute device. In this distributed network of compute-display nodes, the display devices form a geometrically contiguous tiled display wall, and the data distribution among the compute-display nodes for algorithmic purposes is related to the data distribution to its corresponding display nodes for visualization purposes. This approach collocates computation and visualization within the same node and minimizes network bandwidth requirement for data exchange between nodes.

Leveraging the advantages of the above novel system design, we also present a distributed force directed graph layout algorithm. In this distributed algorithm, we address two issues specific to tiled displays: number of edges crossing the display panel boundaries (across two different compute-display clients) and the location of graph vertices close to panel boundaries which may lead to incorrect visual analytic conclusions. We integrate the cost functions expressing the above two effects along with the cost function of the edge-crossings within each display into the graph layout algorithm. Using this distributed approach, for the first time a very large graph has been laid out in a large real estate provided by the tiled display. The proposed system is evaluated on various parameters including the scalability, the impact on display real-estate utilization, and finally, network bandwidth usage.

Keywords

Information Visualization, distributed graph layout, large dataset

1. INTRODUCTION

Information visualization is absolutely critical to make sense out of the data from many real world applications. Many data such as social networks, bioinformatics, cellular phones, wireless sensor network, genetic information can be represented as vertex-edge graphs [3] [4] [14]. However, an exponential increase in the amount of these data has become an important issue for visualization tools [6].

Many visualization tools have used zooming [5] and fish-eye approaches to give users some control to explore specific parts of the graph. However, these techniques may result in distorting visually dominant features of the graph. Additionally, aesthetic layouts prescribe evenly distributed vertices and edges, and edges with all same length; non-compliance of which may lead to misinterpretation of the graph. Further, minimizing the number of edge-edge intersections while visualizing the graph is an important parameter to discern the features of the graph, but it is an NP-complete problem [16] [17] [18]. Adhering, to these principles is first compute intensive: for example, force-directed layout and orthogonal layout algorithms are of time complexity $O(N^3)$, and second may not be even possible to visualize such large graphs in small displays – for example, a million pixel display has space for at most a million vertices of the graph, and we cannot avoid edge-edge intersections or co-located vertices when the entire graph is visualized [1] [11] [17] [19]. So a combination of limited computers and display resources leads to cluttering of visual information on the displays, thus drastically reducing the effectiveness and usefulness of any visualization technique. However, there are a distributed force-directed layout algorithm [20] and parallel force-directed layout algorithms [9] [13] [21] in order to solve the issues. The distributed algorithm can describe how to handle multiple displays in order to overcome spatial limitation, but it shows clustered results for each display. Besides, the maximum number of processors is only eight, and it cannot present results when the number is greater than eight. In addition, parallel algorithms can uses GPU to handle large graph, but it cannot prevent overloaded information on a pixel because of spatial limitations with limited number of displays.

In this paper, we propose a scalable distributed approach to visualize large graph data sets using multiple compute-display nodes. In this system, the display panels abut each other to form a physically large display, and the computers that drive them are connected to each other using a network backbone. In the distributed framework we relate the display real-estate utilization and load balancing by distributing the data for computation among the compute nodes and

*Corresponding author

ICVGIP '12, December 16-19, 2012, Mumbai, India

visualize the same data on the corresponding display node. This minimizes the network bandwidth required for communication between compute nodes. Using these principles, we propose a novel distributed force-directed graph layout algorithm to handle very large graphs, and for maximum utilization of the entire available display real-estate. We also propose an effective solution for the data partitioning problem between the multiple compute-display nodes to load balance and to reduce the communication between the compute nodes. In essence, the system provides a framework to resolve all the graph visualization issues that emanate from lack of computing or display resources in a unified manner.

In the remainder of this paper, we describe the overview of graph layout algorithms in Section 2. Then, we present the proposed distributed graph layout algorithm based on the force-directed layout algorithm on tiled displays in Section 3. Section 4 provides the overview of the system and Section 5 shows the results, examples, and evaluation of the system. Finally, we conclude with possible directions for future work in Section 6.

2. BACKGROUND

Graph drawing is graphical visualization of vertices and edges connecting them. However, with large numberer of vertices and edges, the placement of vertices on the display real-estate becomes critical for effective visual communication without misinterpretation and miscommunication. Assigning geometric coordinates for the vertices such that it optimizes certain visualization parameters, such as minimum edge-edge intersections, clustering of related vertices, etc., is called graph layout.

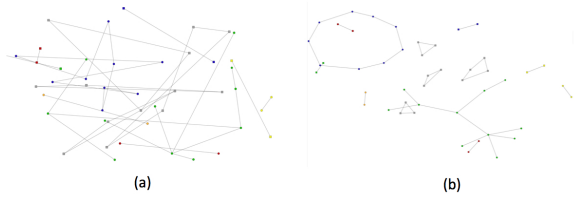

Figure 1: Two different graph layouts of the same graph. The left layout has many edge crossings and lack clustering of related vertices, while the layout on the right is better for visual communication.

Figure 1 illustrates the usefulness and the purpose of good graph layout algorithms. There are many types of graph layout algorithms and the time complexity of these algorithms vary anywhere from $O(N)$ to $O(N^3)$ [6] [17], where N is the total number of vertices in the graph.

In this work, we present a distributed version of the force-directed layout algorithm [10] [19] that makes use of the availability of integrated compute-display nodes as in tiled displays. It assumes that every vertex has the same electrical charge and uses Coulomb 's law to compute the repulsive force among all vertices and every edge is considered as a spring and Hooke's law is used to compute the attractive force between connected vertices. Iteration stops when the positions of the vertices stabilize. However, this force-directed layout algorithm has a time complexity of $O(N^3)$ [10] [19].

Tiled display units are made out of a matrix of LCD display panels or projectors, each of which is driven by a computer (also a computational node). All these computers are connected to each other using a backbone network. The distributed force-directed [15] [22] graph layout algorithm makes use of this architecture of integrated computer-display nodes that are also interconnected, to effect efficient computation, display, and communication among the nodes.

The goal of this work is to design a good graph layout algorithm that can handle very large data sets. Force directed layout algorithm is optimal in minimizing the edge crossings [7], but has high computational complexity. Distributed algorithms usually come with data partitioning, and thus reduce the overall execution time on the data. Distributed algorithms can also handle very large data sets, if they minimize communication between the computational nodes. We propose a scalable distributed force directed layout algorithm,*HD-GraphViz*, that provides high quality graph layout with minimal edge intersections for clarity and discernibility of the graphs, while at the same time reduce the execution time and highly scalable to handle very large data sets. *HD-GraphViz* uses and exploits the compute-display cluster infrastructure for efficient implementation.

3. ALGORITHM

The *HD-GraphViz* physical system set up consists of multiple displays (LCD panels) arranged as rectangular matrix of tiled displays, each display is driven by a computer, and all these computers are connected to each other using Ethernet network. Thus, each display panel along with its computer forms a compute-display node, and the set of all compute-display nodes form a networked distributed computing environment. In addition, the system consists of a data node that initially distributes data to all the computers and a compute-display node that visualize the information of all vertices and edges.

The *HD-GraphViz* is a distributed force-directed graph layout algorithm that is an iterative method that moves the position of the vertices at every iteration until the layout converges or stopped after a certain number of iterations. The advantage of force directed layout, and hence *HD-GraphViz* is that it would work with any graph – not just special graphs like trees. In this section, we elaborate on the proposed distributed force-directed graph algorithm that handles issues like data distribution that is specific to distributed environment and resolves issues like edge-edge intersections of display boundaries that are unique to a tiled display in a unified manner.

3.1 Data Distribution

Tiled displays consist of $m \times n$ matrix of displays. Each display shows only a part of an entire image, and all displays together form the complete image. Hence for visualization purposes, it helps to have the data that is shown in a specific display locally in its compute-display node. Thus, the data distributed to individual compute-display nodes in order to visualize a graph is implicitly determined by the final layout of the graph and the position of the display in the grid of displays. The portion of the data shown in a particular compute-display node is spatially and topologically related to each other.

In order to compute the layout for the entire graph and to make use of all the computing resources available, in-

stead of having the entire data set at every compute-display node, we propose to distribute the data based on the positions of graph data to all the available compute-display nodes. Since the complexity of force directed graph layout algorithm is cubic on the number of vertices of the graph, even marginal reduction in the size of the graph (in this case by data distribution) will have a tremendous impact on the computation time. Unlike the data distribution of the final layout for visualization purposes, during initial distribution, the data that ends up in the same node need not have any spatial or topological relationship to each other.

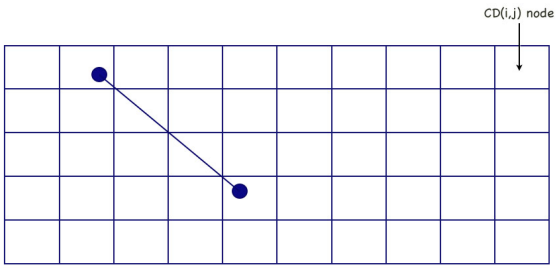

Figure 2: The method to divide dataset based on positions of vertices. Vertex A and B are placed on CD(1,2) and CD(4,5) nodes respectively. An edge connects two vertices, and both the nodes CD(1,2) and CD(4,5) get the information about both the vertices. Further, the edge passes through the nodes CD(2,3) and CD(3,4) also, so they also have the information about these two vertices in order to draw the edge correctly.

3.2 Per Node layout

Following the initial distribution of graph information to the compute-display nodes, each node modifies the layout of the graph independently, using the same force-directed layout algorithm. Figure 3 is the pseudo codes of the proposed distributed graph layout algorithm.

```
set initial positions of all vertices
while n iterations
   all CD*(i,j) communicate with a data node to divide dataset
   for each vertex u in CD(i,j)
      for each vertex v in CD(i,j)
         calculate repulsive forces between vertices
      for each edge e
         calculate attractive force between u and v
         if u and v are not in the same CD(i,j)
            calculate repulsive forces between vertices at different CD nodes
      for each vertex u in each CD(i,j)
         update positions based on repulsive and attractive force
```

Figure 3: Pseudocode for implementing the distributed force-directed layout algorithm.

As we can see, the distributed version of the algorithm is similar to that of the original force-directed layout algorithm [10]. The main loop of the method computes the attractive and repulsive forces on every vertex, and this process continues until there is equilibrium between these forces or for a fixed number of iterations that user chooses. One basic difference between a centralized and distributed algorithm is the case when the two neighboring vertices with an edge between them, are in two different display panels (compute-display nodes). This case is further studied later in this paper in the context of reducing the edges crossing the display boundaries, and not placing the vertices close to the display boundaries and corners. Note that both the host compute-display nodes of the two vertices will have information about the other vertex as the vertices are edge connected neighbors. The updated forces on a vertex computed by a compute-display node that is not the host of the vertex, is communicated to the host node. The final position of the vertex is computed by the host node of that vertex, and the new position may be outside the boundaries of the host display. In that case, this vertex is handed over to that compute-display node to whose space it has moved.

The best case computational complexity of the distributed force directed layout is $O(\frac{N^3}{P^3})$ where P indicates the number of processing nodes. The worst case complexity of this algorithm is $O(N^3)$ – consider a complete graph with N vertices in which one or more vertices are assigned to each compute-display node. Since each vertex is connected to every other vertex, each CD node has to have information about all the other vertices and compute the attraction and repulsion forces for all the vertices in the graph, thus leading to the complexity of the sequential algorithm, $O(N^3)$. Figure 4.(a) shows a result of the above proposed *HD-GraphViz* algorithm with 100 vertex graph on a single compute-display node. It is exactly same as the original force-directed layout algorithm. We can see that they are well-balanced on single screen. Figure 4.(b) presents a result with the same size of a data set on four tiled display that is four individual displays and computing machines. The red line in the figure shows the display boundaries of the four displays. Although each compute-display node carries out the computation only on a limited set of verities of the graph, note the quality of the final layout computed by the distributed algorithm is comparable to that of the original centralized layout algorithm in Figure 4.(a).

3.3 Data Redistribution

The graph visualization on tiled display has two issues: edge crossing across panel boundaries and vertex on panel boundaries. The goal is to minimize these two artifacts in order to discern all vertices for understanding the features of graph dataset.

3.3.1 Reducing cross-panel edge-crossings

Figure 5 describes the case when a position of a vertex is at center of tiled displays. In this case, four displays share a node's information. Since most display panel boundaries have a certain thickness, it is possible that these boundaries obstruct the recognition of the vertex by the user. Therefore, we should move the node to one of four compute-display nodes to secure a clear view.

In order to prevent a vertex to be placed at the edge between two adjacent tiled displays, we add repulsive forces on every boundary of display panels. In Figure 6, the blue color line is repulsive forces from each boundary of displays. The red two-way arrow shows how repulsive force work. A vertex close to a tile boundary will move away from it toward the interior of the panel.

Since the goal is to minimize edge-crossings across the panel boundaries, we move a vertex to one of its edge connected neighbor's panel if it locally reduces the number of

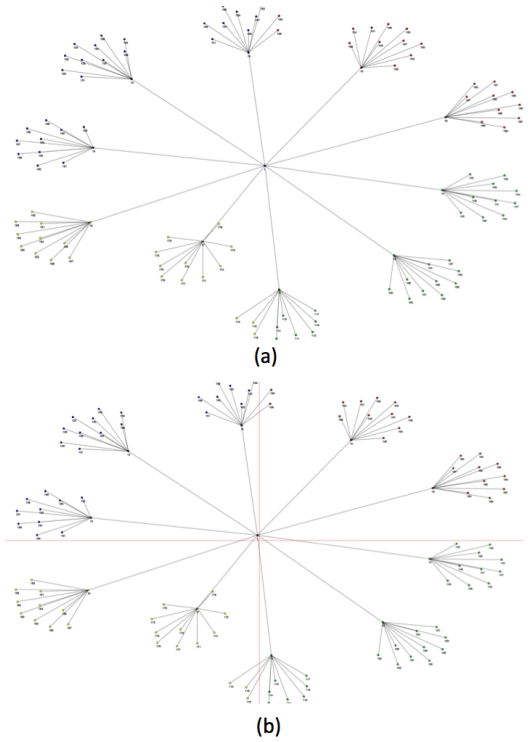

Figure 4: An example of a tree graph with the proposed algorithm on single processor with one and four virtual displays. The red lines of (b) indicate the borders of virtual displays.

edges crossing the panel boundaries. In case of tree, we move all the leaf vertices to its parent's display node, in case they are different. This is sure to reduce the number of edge crossings.

Figure 7 shows an example of tree graph data across displays. There are three vertices which have the same ancestor. Because those vertices are placed on different compute-display nodes from the compute-display node of the ancestor, it causes edge-crossings over the panel displays. Those edge-crossings are unnecessary if all vertices are located at the same compute-display node. Thus, we move descendant vertices to the same node that has the ancestor. The right picture of Figure 7 describes that there is no edge-crossings among the ancestor and the descendants at last.

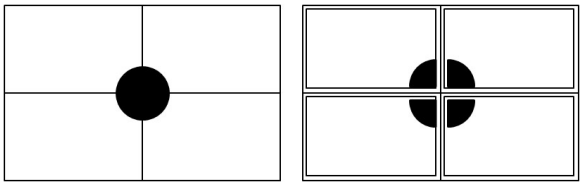

Figure 5: An example of the case that a vertex is at the center of mutiple display panels. It is possible to recognize the position of a node if the display panel boundaries do not have any thickness (left), but it is more difficult to discern the position of the vertex if thickness of display panels is wide (right).

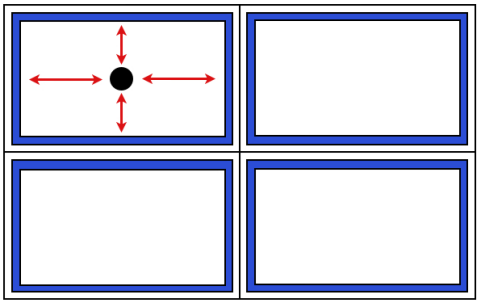

Figure 6: An example of repulsive force on the boundaries of the display panels. The blue color line is the repulsive force from the display boundaries and the red arrow shows how repulsive force is working. It moves a vertex into a node.

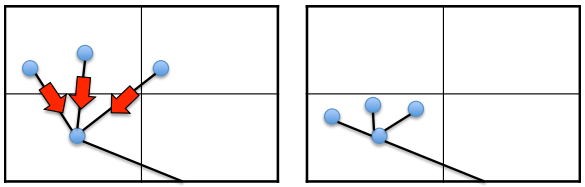

Figure 7: An example of how to minimize edge-crossings over different display panels. Left figure shows that three descendant vertices are placed on different node from the ancestor vertex's node. In this case, there are 4 edge-crossings. To reduce the edge-crossings, those vertices have to move to the same node of the ancestor. Right figure shows the result after moving all vertices to the same node of the ancestor. There is no edge-crossing among the ancestor and the descendants.

3.3.2 Reducing data redundancy

Figure 8 shows how to reduce data redundancy. The left figure shows an example in which the vertices that may be connected by an edge are placed on different compute-display nodes and the chart on the right represent vertices CD nodes have and may be required to communicate with each other. In Figure 8.(a), vertices A and B are on CD(1,1) and vertex C is located at CD(1,2). For displaying the vertices and edges of the graph, CD(1,1) has the information of vertex A, B, C because the edge AB and part of the edge AC must be rendered. In addition, CD(1,2) has the information about vertices A and C because a part of the edge AC has to be rendered by CD(1,2) as well. In essence, if we reduce graph edge - display panel boundary intersection, this data redundancy will also reduce as a result. In Figure 8(b) after moving the vertex C into CD(1,1) node, CD(1,2) can delete all information about both vertices A and C, thus reducing the redundant storage of these vertices in more than one compute-display (CD) node.

3.4 Network Bandwidth

In our problem, network resources are consumed after every iteration of the force directed layout to communicate the transfer of vertices to their new destinations, if it is a different compute-display (CD) node than its current one. One possibility is to have a centralized approach to com-

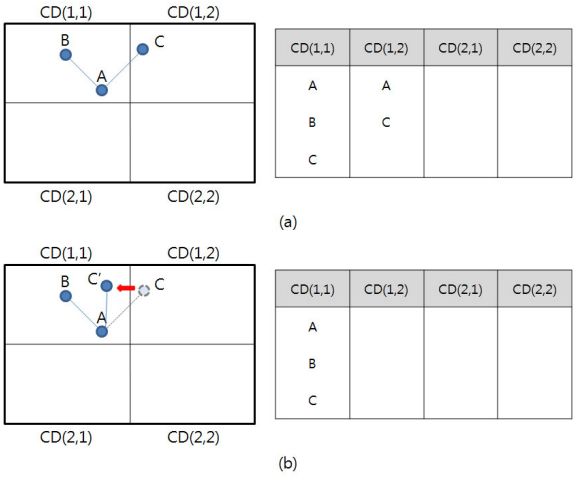

Figure 8: An example to minimize data redundancy. (a) Vertex A and B are on CD(1,1) and vertex C is on CD(1,2). However, Vertex C must be included on CD(1,1) because of the edge and vertex A must be on CD(1,2). (b) CD(1,2) sends the updated information to CD(1,1) since vertex C is moving into CD(1,1). Only updated information which is moving to other CD node, we can reduce the redundancy of dataset.

mit all changes to a master node, and let the master node redistribute this data to its new hosts. We have taken a completely decentralized approach in which each CD node will transfer only the vertices that moves out of itself to the CD that is their new destination. Usually, these vertices do not move too far, but move only to the adjacent CD node. So the communication pattern, in general, is not a clique among all the CD nodes which is not scalable, but only a constant degree neighborhood graph whose networking complexity does not increase by increase in the size of the tiled display.

In addition, this communication after every iteration resolves one key issue of distributed computing, 'synchronization'. It synchronizes all compute-display nodes with the step of the layout algorithm. The proposed algorithm is an iterative algorithm that displays the result of each step, so if all compute-display nodes have the same step, the system is synchronized.

4. SYSTEM OVERVIEW

HD-GraphViz is based on distributed computing approach, especially on an architecture provided by tiled displays. All machines are connected to each other through Ethernet. The system consists of two parts: a data node and a compute-display node. The data node focuses on data collection and redistribution to the compute-display nodes. On the other hand, the compute-display node renders the part of graph dataset from the data node. Besides, it computes the new graph after each iteration of the force directed layout and communicates the changed positions of the graph vertices to other relevant neighbors. In addition, the target system is Highly Parallelized Display Wall(HIPerWall) [2] that consists of 50 computing machines and 50 30" displays. Figure 9 shows a picture of HIPerWall that runs the distributed layout algorithm with multiple scenarios.

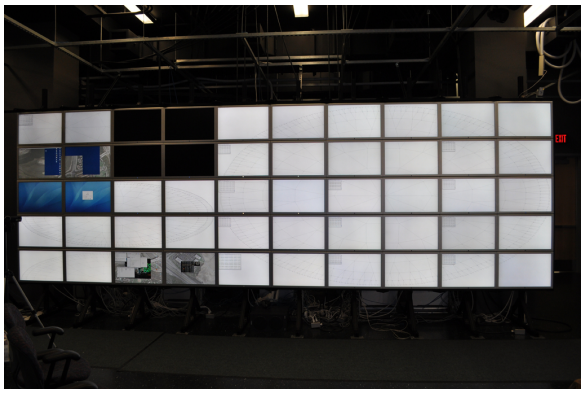

Figure 9: HIPerWall [2] is the target system that consists of 50 computers and 50 30" displays.

5. RESULT

5.1 Scalability

The main goal of the proposed algorithm is to handle large size dataset using the force-directed layout. We defines the size of large dataset is more than 1,000 vertices [8]. *HD-GraphViz* uses multiple nodes to visualize dataset on tiled displays. Figure 10 represents the processing time on multiple compute-display nodes. X-axis shows the number of compute-display nodes, 1, 2, 4, 8, 10, and up to 24, respectively, and y-axis shows the average processing time. As we can see, the processing time from 1 node to 24 nodes is almost more than 500 times improved. The processing time we calculate includes the sum of the time required for sending data, processing data on each compute-display node, and receiving data. On average, the processing time takes more than 95 % and communication time takes less than 5 %. But the communication time can increase unexpectedly. Also, the processing time, as computed is the longest time taken among all the compute-display nodes. If the data is not uniformly distributed, this time may get as worse as a single node computation. Finally, with more CD nodes, the inter-node communication time may dominate over the processing time, and hence there may not be any substantial improvement in the total processing time (in Figure 11, see the difference between 18 and 22 nodes). When we run 50,000 vertices on a single node, it takes more than a day per one step while it takes only 235 sec on 24 nodes.

5.2 Quality of Data Layout

In this subsection, we describe the quality of data layout. There are three issues to visualize graph data on tiled display. The first one is that some vertices are placed on the boundaries, another is how to reduce edge-crossings over the display panel boundaries and the other is how to keep the originality of the graph on tiled display compared to the graph on a single node. We describe the results of what we proposed in Section 3.

5.2.1 Repulsive forces on panel boundaries

The proposed layout algorithm prevents any vertex to be located at the boundaries in order to increase discernibil-

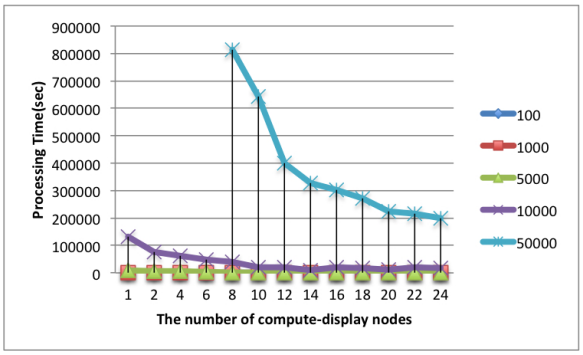

Figure 10: Dataset vs. average processing time on multiple nodes. The more compute-display nodes we have, the less processing time we can have. However, its performance is saturated after 18 to 22 compute-display nodes.

ity. Figure 11 shows the before and after repulsive on panel boundaries. The red lines presents the panel boundaries and we can see that some of vertices are placed on boundaries in Figure 11.(a). Whereas, there is not any vertex on boundaries after we apply the repulsive force on boundaries in Figure 11.(b). Therefore, we have more discernible graph over the boundaries.

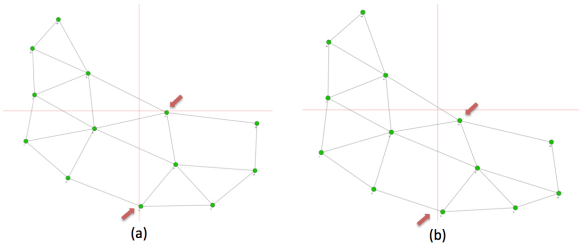

Figure 11: An example of repulsive on the panel boundaries. The red line represents a panel boundary. (a) shows that one of vertices is located at the boundary, but there is not a vertex on the boundary any more in (b).

5.2.2 *The reduction of edge-crossings over panels*

HD-GraphViz layout algorithm can reduce edge-crossings over multiple panel displays with moving vertices to one of neighbor displays. However, it is restricted to tree graphs since planar graphs are more complicated. Figure 12 shows the result of graph layout before and after the algorithm applied. Figure 12.(a) presents edge-crossings over the boundaries while Figure 12.(b) describes non-edge crossings between an ancestor and its descendants. However, the root of the graph might have some edge-crossings because it is placed on the center of the screen. It causes edge-crossings and we cannot avoid those crossings at all. After all, we can reduce edge-crossings over multiple panel displays if we run tree graphs on tiled displays.

5.2.3 *Quality of Graph on tiled displays*

Figure 13 shows the result of *HD-GraphViz* on 4 nodes without any algorithm to reduce edge-crossings and 4 nodes

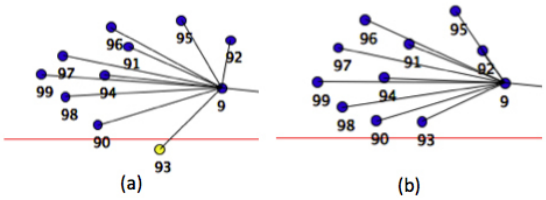

Figure 12: It shows the result before and after the reduction of edge-crossings over panel displays. The red line represents a panel boundary. There is a yellow colored vertex in (a) and it is located at different compute-display node. However, we can see all vertices are placed on the same node after the algorithm applied.

with repulsive force on displays' boundaries. We can see edge-crossing is reduced by applying the algorithm. Therefore, we keep the shape of data quality of data layout with reducing edge-crossings compared with the layout before the algorithm is applied.

Figure 14 represents the result with 3,000 vertices of a tree graph and 50,000 vertices of Facebook [12] data on a tiled display which consists of 16 compute-display nodes. The red lines describe the displays' frame. There is no misalignment and it keeps the shape of tree structure on a tiled display after the algorithm applied. The upper figure shows the symmetric graph data and the lower figure shows non-symmetric tree graph data.

6. CONCLUSION

HD-GraphViz is the scalable distributed graph layout algorithm on tiled displays in order to handle large size of datasets and provide clear view to users. It covers a tree and a planar graph. Moreover, it reduces the edge-edge interactions over panel displays as well as the force-directed layout algorithm reduces among vertices. However, we should consider one thing about the relationship between the size of data and the number of compute-display nodes. We can observe the system performance increased with more number of compute-display nodes, but it is saturated with the certain number of nodes.

There are several opportunities for continuing the research, *HD-GraphViz*. First of all, we need a way to reduce processing time or improve time complexity. For instance, we can apply clustering algorithm to *HD-GraphViz*. Before we organize graph dataset using the distributed force-directed layout algorithm, we make several clusters in order to handle the part of dataset. It is different from data distribution that we describe. In each compute-display node, we make clusters hierarchically with a part of a dataset for the node. We change all clusters to super nodes and have less number of vertices. Since the key issue of graph drawing is the graph size, especially for the force-directed layout algorithm, we can reduce processing time to organize graph with super vertices compared with original dataset. Next, we need to apply more practical large datasets to the system, *HD-GraphViz*. The size of those dataset is big enough in order to test the system. It can show what we have to do more and how to develop the system.

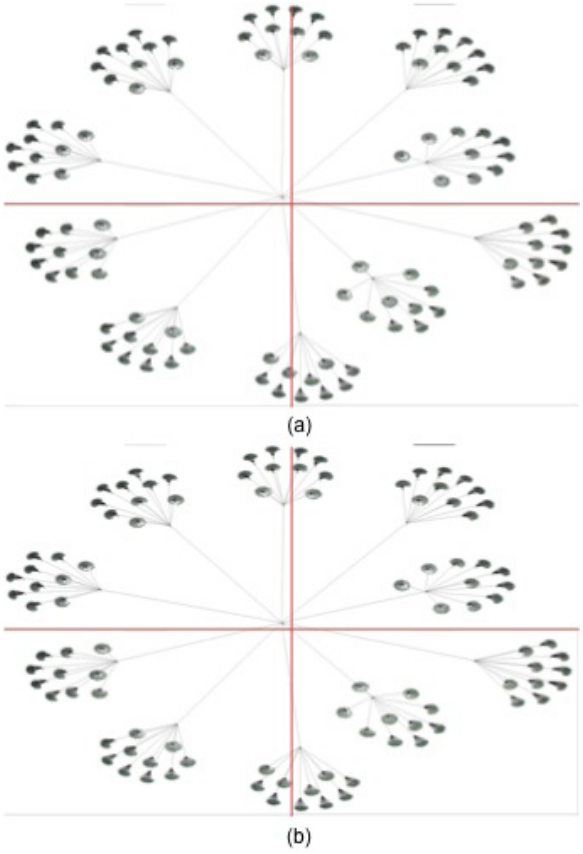

Figure 13: The result of graph layout algorithm on different environment with 3,000 vertices and 3,000 edges. (a) The distributed graph layout on 4 nodes. (b) The distributed graph layout algorithm on 4 compute-display nodes after applying repulsive force on displays' boundaries.

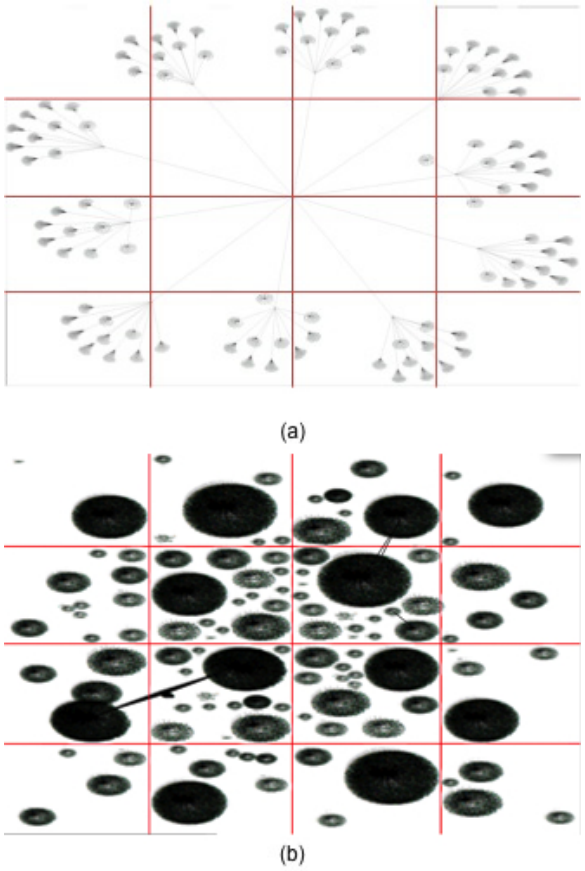

Figure 14: The result of the *HD-GraphViz* on tiled display consists of 16 compute-display nodes. The upper graph shows 3,000 vertices and the lower graph shows 50,000 vertices from Facebook [12] on 16 compute-display nodes. The red line presents the frame of displays.

7. REFERENCES

[1] http://www.dia.uniroma3.it/ gdt/gdt4/blag.php.

[2] http://hiperwall.calit2.uci.edu/.

[3] A. T. Adai, S. V. Date, S. Wiel, and E. M. Marcotte. Lgl: creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal of Molecular Biology*, 340:179–190, 2004.

[4] E. Adar. Guess: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM.

[5] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30:535–546, 2004.

[6] J. Diaz, J. D, J. Petit, and M. Serna. A survey of graph layout problems, 2002.

[7] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *In:Proc. 14th Intl. Symp. Graph Drawing(GD 06). Vol 4372 of Lecture*. Springer, 2007.

[8] A. Efrat, D. Forrester, A. Iyer, S. G. Kobourov, C. Erten, and O. Kilic. Force-directed approaches to sensor localization. *ACM Trans. Sen. Netw.*, 7(3):27:1–27:25, Oct. 2010.

[9] Y. Frishman and A. Tal. Multi-level graph layout on the gpu. *IEEE TRANS. VIS. COMPUT. GRAPH*, 13:1310–1319, 2007.

[10] T. M. J. Fruchterman, Edward, and E. M. Reingold. Graph drawing by force-directed placement, 1991.

[11] B. Genc and U. Dogrusoz. A layout algorithm for signaling pathways, 2006.

[12] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proceedings of IEEE INFOCOM '10*, San Diego, CA, March 2010.

[13] A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Graph drawing. chapter Rapid Multipole Graph Drawing on the GPU, pages 90–101. Springer-Verlag, Berlin, Heidelberg, 2009.

[14] C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. In *In 12th Symposium on Graph Drawing (GD 04)*, 2004.

[15] D. Griffiths. *Introduction to electrodynamics*. Prentice

Hall, 1999.

[16] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, New York, NY, USA, 2005. ACM.

[17] I. Herman, I. C. Society, G. Melanceon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000.

[18] S. J. Crossing number is np-complete. *Algebraic and Discrete Methods*, 4:312–316, september 1983.

[19] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31:7–15, April 1989.

[20] C. Mueller, D. Gregor, and A. Lumsdaine. Distributed force-directed graph layout and visualization. In *Eurographics Symposium on Parallel Graphics and Visualization*, May 2006.

[21] A. Tikhonova and K.-L. Ma. A scalable parallel force-directed graph layout algorithm. In *EGPGV*, pages 25–32, 2008.

[22] A. Ugural and S. Fenster. *Advanced strength and applied elasticity*. Prentice Hall PTR, 2003.