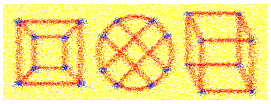

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 21, no. 1, pp. 55–80 (2017)
DOI: 10.7155/jgaa.00406

An Incremental Layout Method for Visualizing Online Dynamic Graphs

Tarik Crnovrsanin Jacqueline Chu Kwan-Liu Ma

Computer Science,
University of California at Davis, Davis, USA

Abstract

Having the ability to draw dynamic graphs is key to better understanding evolving relationships and analyzing the patterns and trends in a network. Traditional force-directed methods are not suitable for laying out dynamic graphs because of their design for static graphs. An alternative is to create an incremental version of the force multilevel multi-pole method (FM^3); however, previous solutions are more susceptible to graph degradation, that is, graph illegibility due to long edges or edge crossings. This is typically caused when distant components are connected, resulting in long and overlapping edges. We present our incremental version of FM^3 with a refinement scheme, which solves this problem by "refining" the parts of the graph with high energy. Our resulting visualization maintains readability of the graph structure and is efficient in laying out these changing networks. We evaluate the effectiveness of our method by comparing it with two previous online dynamic graph methods.

Submitted:	Reviewed:	Revised:	Reviewed:	Revised:
November 2015	March 2016	May 2016	August 2016	September 2016
	Accepted:	Final:	Published:	
	October 2016	November 2016	January 2017	
	Article type:	Communicated by:		
	Regular paper	E. Di Giacomo and A. Lubiw		

This research is sponsored in part by the U.S. National Science Foundation via grants NSF DRL-1323214 and NSF IIS-1320229, the U.S. Department of Energy through grant DE-FC02-12ER26072, and also the UC Davis RISE program.

E-mail addresses: tecrnovr@ucdavis.edu (Tarik Crnovrsanin) sjchu@ucdavis.edu (Jacqueline Chu) ma@cs.ucdavis.edu (Kwan-Liu Ma)

1 Introduction

Graphs are ubiquitous, found in fields ranging from biology and chemistry to sociology, software engineering, and cyber security. Graph drawing can be used as a visual means to understand and analyze this relational data. In graph visualization, the nodes represent entities and links represent relationships. Although techniques exist to aesthetically and efficiently lay out static graphs [19, 24, 25, 29, 30], these are not intended for applications with graphs that change over time [13].

Real-world applications, such as Facebook and Twitter, require analysis of dynamic networks. Finding the optimal way to visualize dynamic graphs remains a challenging research topic. The primary goal for dynamic graph visualization is to ensure the stability of the layout [8, 17, 28, 31] and preserve the mental map [1, 34, 36, 37]. The mental map represents the user's underlying understanding of the graph structure. It is important that the mental map remains consistent, or stable, over time. If not, confusion may occur.

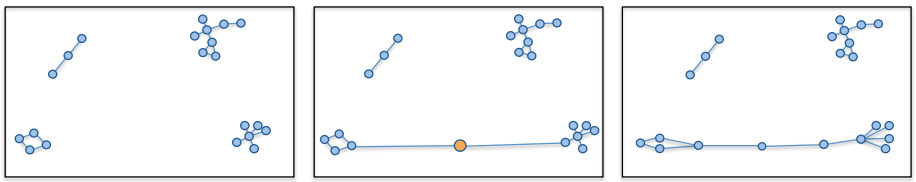
Visualizing dynamic graphs is often done by animating over the sequence of offline graphs where all the time steps are known in advance [6, 14, 17, 36]. Another approach is to display selected time steps side-by-side as small multiples [44]. Given prior knowledge of the complete time sequence, one can optimize the layout for animation and specific analysis goals [7, 13, 14, 32]. However, this dependency on prior knowledge restricts these methods from online applications.

In online applications, such as real-time monitoring of networks or systems, the graph is constantly updated and its behavior over time cannot be predicted. This added complexity, coupled with the lack of available online dynamic graph data, makes generating optimal layouts an even more challenging problem with limited attention in existing research [9, 17, 21, 32].

We have examined existing online dynamic graph layout methods and found their results to have undesirable limitations on performance or layout quality. Some are too expensive to use for real-time applications. Others are more successful when the algorithm involves anchoring large portions of the graph and allowing only a small subset of the graph to move. Nevertheless, these methods come with several trade-offs. Although the non-anchored nodes move to their ideal locations in most cases, an undesirable effect happens when two disconnected components merge. Nodes usually cannot reach their ideal positions immediately, as shown in Figure 1. This linking of disconnected graphs may also lead to edge crossings. These parts of the graph stay in suboptimal positions, and can only improve their placement if new nodes or edges are added to the same neighborhood.

In this paper, we present an incremental version of the multilevel multi-pole layout method that is suitable for visualizing online dynamic graphs. It is publicly available as an open-source library. The library is written in C++ and for modern graphics cards. The library can be found at `http://vidi.cs.ucdavis.edu/Projects/Streaming` with an example that requires Qt. Our work makes the following contributions to online dynamic graph drawing:

- Our incremental layout method maintains graph readability and is efficient when laying out the network changes.



(a) (b) (c)

Figure 1: An undesirable limitation. A graph has many disconnected components (a), and a node is introduced which links two components together (b). One layout method [17] allows the new node (in orange) and its neighbors to move after the new node is added. If these nodes cannot reach their ideal position in a single time step, they are affixed to the same positions (c) until new nodes or edges are later introduced into the same neighborhood.

- The refinement technique reduces long edges by using the nodes' energy to determine correct placement.
- The refinement technique can be applied independently or in tandem with an existing force-directed layout method.
- The layout method is fast because our implementations for both the layout and refinement calculations are GPU-accelerated.

We evaluated our method using several dynamic graph data sets, including ones from real-world online applications, and compared the layouts with those produced by previous methods. Also, we expand on our previous version of this research [12] by discussing scalability of our incremental method with refinement. The test results demonstrate the effectiveness and usability of our method when visualizing online dynamic networks.

2 Related Work

Force-directed approaches have several features that are beneficial for visualizing evolving networks. These methods produce aesthetically appealing layouts and smooth animation due to their underlying physical model, which moves the graph towards a low energy state. The addition and removal of nodes can be treated as the creation and removal of particles. Using force calculations, the algorithm applies forces on these "particles," which affect their position in the graph. Beck et al. [4] provide a comprehensive review of state-of-the-art techniques in dynamic graph visualization. However, force-directed approaches are not without fault: Minor changes to the graph, such as the linking of two disconnected components, can have a large impact on the forces.

Several static force-directed layouts for large graphs have been proposed [24]. Generally, these force-directed layouts attempt to minimize the number of force calculations. Walshaw [46] used a multilevel approach, where a coarser graph is generated

recursively through an edge collapse operation. Harel and Koren [25] used a similar multilevel approach, based on an approximation of the k-center problems. Quigley et al. [39] employed a multi-pole approach, where distant nodes were grouped together into a single node cluster for force calculation. Their implementation used a quadtree for clustering. Gajer et al. [18] used a maximum independent set filtration technique to coarsen the graph. Hachul et al. [24] combined an efficient multi-pole and multilevel method to lay out the graph. Instead of using a quadtree, Hachul et al. used a kd-tree which guarantees $O(N \cdot \log N)$ for all pair repulsion forces. Koren et al. [30] took a different approach by optimally minimizing the quadratic energy function. This minimization problem is expressed as a generalized Eigenvalue problem. To achieve a fast layout, they use an algebraic multi-grid algorithm. Harel and Koren's technique [26] embeds the graph in a high-dimensional space and then projects down to 2D.

Dynamic graphs are often visualized using timelines, animation, or as a hybrid of the two. Although our focus is on animation, timelines provide a comprehensive overview of the transition between time steps. Greilich et al. [22] introduced TimeArcTrees, which vertically aligns the graph nodes and facilitates the comparison of nodes across time steps. Burch et al. [10] extended TimeArcTrees using parallel edge splatting. In this technique, the graph is first arranged into a bipartite graph and edge density is applied to reduce the overplotting of the edges. Reda et al. [40] used timelines to visualize the evolution of communities. They used a sorting strategy to place large and constantly changing communities on top of the visualization, based on an influence factor. Vehlow et al.'s [45] approach visualizes both dynamic graphs and the dynamic community structure in a single image. They also apply a reordering strategy to the communities, similar to that of Reda et al.'s, and advance color assignment to improve edge crossings. As Beck et al. pointed out, these methods do not scale well with larger graphs.

For animation, several algorithms were developed to handle offline dynamic graphs. Diehl and Görg [13] built a metagraph from the time series to help preserve the mental map. Erten et al. [14] used a modified GRIP algorithm to render 2D and 3D animations of evolving graphs. Kumar and Garland [31] abstracted the graph through stratification which employs a hierarchical force-directed layout algorithm. Brandes et al. [7] used a spectral method to improve animation between time steps. Collberg et al. [11] created GEVOL, a system for visualizing the evolution of software. Sallaberry et al. [41] clustered each time step and linked those clusters throughout time. The clustering provides a node order which is used by their previous work [35] to generate a graph layout.

Online dynamic graphs are series of graphs in which time steps and the changes that occur within them are not known ahead of time. Lee et al. [32] created an algorithm that preserves the mental map while generating aesthetically pleasing graphs. The drawback is that the algorithm is slow from recalculating the full layout at each time step. Brandes and Wagner [9] instead used Bayesian decision theory to generate the graph. Frishman and Tal [17] created a novel force-directed algorithm that can handle large graphs. Their implementation runs on the GPU and provides a 17 times speedup over their CPU version. Gorochowski et al. [21] looked at using the age of the node to stabilize the graph. The age was calculated based on when the node appeared and how much movement it saw through its lifetime. Hayashi et al. [27] studied how

initial node placement affects the responsiveness of their layout algorithm. Our work is similar to Frishman and Tal's approach, but differs due to how the multilevel multi-pole is constructed and includes a refinement step.

Another research area of online dynamic graphs is on constrained graphs. Frishman and Tal [15] looked into dynamically drawing predetermined clustered graphs. North [36] addressed drawing dynamic directed acyclic graphs, while Görg et al. [20] looked into drawing sequences of orthogonal and hierarchical graphs.

A common speedup technique for force-directed layouts is to implement the algorithm on the GPU. Auber and Yves [2] implemented a force-directed algorithm that uses Euler's method. Gumerov and Duraiswami [23] parallelized the fast multi-pole method and managed to achieve a 30 to 60 times improvement over their CPU version. Stock and Gharakhani [43] developed a GPU version of the N-body algorithm. Although this paper does not directly discuss graphs, the N-body algorithm is used in many of the fast force-directed layout methods [16, 19, 23]. Frishman and Tal [16] created a multilevel force-directed graph layout. In turn, Godiyal et al. [19] implemented a GPU version of FM^3 that is 30 percent faster than Frishman and Tal's algorithm. This speedup difference can be attributed to the approximation method for the all-pairs repulsive forces used in Frishman and Tal's approach. It has a larger aggregated error compared to that of FM^3 with the possibility of becoming unbounded for unstructured distributions [42].

Measuring the aesthetics of a layout is a difficult process. Bennett et al. [5] thoroughly summarized the heuristics used to measure the aesthetics of graph visualization. Measures of interest include edge lengths and crossings, with the idea that both edge crossings and lengths should be minimized. Frishman and Tal [17] introduced the notion of potential energy to measure the layout quality. Lower energy implies low stress on the graph. Lower energy also leads to shorter edges. Gorochowski et al. [21] used the same energy calculations to compare their work against Frishman and Tal's. To provide a fair comparison, we also use the same energy calculations when comparing our layout with theirs.

Our layout method achieves better performance with the help of our novel refinement technique that gradually alleviates areas of high energy. Energy is defined as the amount of force applied to a node. The evaluation in Section 4 indicates that our method produces aesthetically pleasing layouts, at the cost of greater movement of nodes. This movement is necessary to bring pairs of nodes connected with long edges closer and thus improves overall readability.

3 Methodology

The methodology section describes the process of converting FM^3 to an incremental algorithm. We explain how our refinement technique works and how it can be applied either independently from or integrated into the layout method.

3.1 An Incremental Algorithm

Our incremental algorithm is a modified version of FM^3, which is a fast, multilevel multi-pole force-directed layout method. Compared to traditional force-directed layouts, FM^3's efficiency is due to the use of the multilevel and multi-pole strategies.

First, the multilevel component reduces the number of expensive calculations by constructing a skeleton of the layout, which can be quickly computed. FM^3 starts with the original, or finest, graph, G_0, and uses maximal independent set to select super nodes. A super node is a single node that represents a large set of nodes from finer levels. These super nodes make up a coarser graph, G_1, and the process is repeated until the user-specified threshold–such as the number of nodes or in-between level changes–is met. At this point, this subset of nodes represents the coarsest level, G_K.

Secondly, the multi-pole component approximates the all node-pair repulsive force calculation. Nodes that are sufficiently far from a given node, v, are grouped together before the repulsive forces are calculated. In FM^3, this partition of distant nodes is determined by using a kd-tree. The grouped nodes are represented as a single node, which is located at the group's centroid. The resulting force of this representative node is the sum force of all the nodes within a respective partition.

During the layout step, a force calculation is applied to graph G_K, in which the resulting node positions are used for the initial layout of the finer graph, G_{K-1}. These steps are repeated until the original graph G_0 is drawn. At each step, nodes have a smaller range of movement than that of the previous level. This process of reducing movement over time is called simulated annealing, which in turn reduces the temperature. Temperature represents a dampening factor to the amount of distance a node can move. More details of FM^3 can be found in Godiyal [19], on which our GPU-accelerated implementation is based.

FM^3 is not designed for online dynamic graph drawing. To make it incremental, we need to:

1. Include an initial layout construction step
2. Add a merging step, which places new nodes and selects nodes to move
3. Modify the multilevel calculation step
4. Pick a specific force model for the force calculation step
5. Add an animation step for smooth transition of the layout rendering

We describe each of these five steps in more detail below:

Initial Layout Construction: For the initial layout, L_0, we use standard FM^3 layout with a degree metric for the selection of the super nodes, which is described in the Multilevel Calculation section.

Merging: This stage attempts to place new nodes at their ideal positions by using positioned nodes from the previous layout. Initial node placement is imperative because error is introduced when previously positioned nodes are at suboptimal positions. This error propagates across layouts, making it difficult to correct in subsequent time steps.

Our approach minimizes this error by assigning coordinates to new nodes in the following manner. Positioned nodes from L_{i-1} are copied over to L_i. If a new node v is not connected to any other positioned node, v is placed in a random position within the

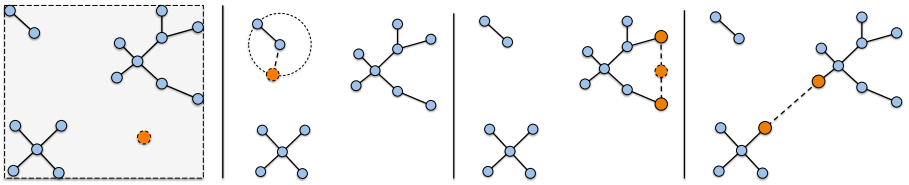


Figure 2: The figure shows how our algorithm assigns positions to new nodes. A dashed node and edge indicate a new node and edge, respectively. Nodes colored in orange represent nodes that are flagged to move by our algorithm. (a) A node with no edges is placed randomly inside the bounding box of the graph. (b) A node connected to a positioned node is placed at a desired length, dl, from the positioned node. (c) A node connected to at least two positioned nodes is placed at the centroid of the positioned nodes. (d) When an edge is added or removed between two positioned nodes, our algorithm flags both nodes to move.

bounding box, as shown in Figure 2a, where the bounding box is the smallest rectangle that all nodes fit within. If v is connected to one positioned node u, v is placed randomly around u at a distance dl, where dl is the desired length between two connected nodes in our spring-based energy model. Figure 2b shows an example of an orange node that is placed at distance dl from its already positioned neighbor. If v is connected to at least two positioned nodes, v is placed at the geometric center of all the connected nodes, shown in Figure 2c. All affected nodes are flagged to move.

In our merging stage, the insertion or deletion of edges affects node placement. If an edge is inserted between two new nodes, u and v, node u is randomly placed inside the bounding box, similar to Figure 2a, and node v is placed randomly around u at a radius of dl, equivalent to Figure 2b. Both nodes are selected to move. Also, our method moves positioned nodes when a new edge is introduced to another node–whether new or positioned–namely, when node u is connected and node v is not. Since it is not restricted by other nodes, node v is randomly placed around node u at a distance dl as if it were a new node and is marked to move. Another instance of node placement is the change of connectivity between positioned nodes u and v. When an edge is removed, the two affected nodes are flagged to move because their current positions are invalid and should move closer to their respective components. After adding an edge, we flag both nodes to move, shown in Figure 2d, to minimize overlapping edges in case these components are distant from one another.

Multilevel Calculation: In FM^3, the process of picking a super node is done randomly or by indexing [19]. When dealing with multiple levels from the coarsening of G_0, our method is more deterministic when selecting a super node than FM^3's multilevel approach. In our implementation, super nodes are selected by their degree in descending order. A new node will have a low chance of becoming a super node, but the likelihood increases when its degree increases.

Having a multilevel representation of the graph reduces the computation time. In incremental layout methods, including ours, only nodes within a certain vicinity have

their forces calculated. Also, coarser graph levels have cheaper computation compared to the original graph because force calculations are applied to the super nodes. Starting from the coarsest level, the super node's resulting movement is used to interpolate the movement of its adjacent nodes at the next finer graph level, until the finest level G_0 is reached. In our method, when we calculate L_i, where L_i is the layout calculation at i^{th} time, we compute the layout 250 times at the coarsest level. The number of iterations to compute the layout decays linearly until we reach the finest level. At the finest level, we compute the layout 30 times. These are the default parameters which work well in practice. The user can change these to meet their needs.

Our method uses a contribution factor to restrict the range of movement for the super nodes. In the traditional FM^3, nodes at coarser graph levels have greater range of movement than those at finer levels. This is caused by the nature of simulated annealing. The contribution factor counterbalances this large disparity of movement in favor of maintaining the mental map. This also prevents suboptimal node positioning per level, which, if not addressed, can ultimately degrade the final graph level, G_0. Each super node's contribution factor is determined by how many of its nodes are allowed to move. For example, if there is only one node that is allowed to move under a super node, then the super node will only move slightly.

Force Calculation: The repulsive forces are modeled as

$$\vec{F}^{rep} = \frac{C \cdot (\vec{u} - \vec{v})}{\|\vec{u} - \vec{v}\|^3} \tag{1}$$

to achieve a greater spreading of disconnected graph components. The spring forces can be modeled as [19]

$$\vec{F}^{spring} = \|\vec{u} - \vec{v}\| \cdot log\left(\frac{\|\vec{u} - \vec{v}\|}{dl}\right) \cdot (\vec{u} - \vec{v}) \tag{2}$$

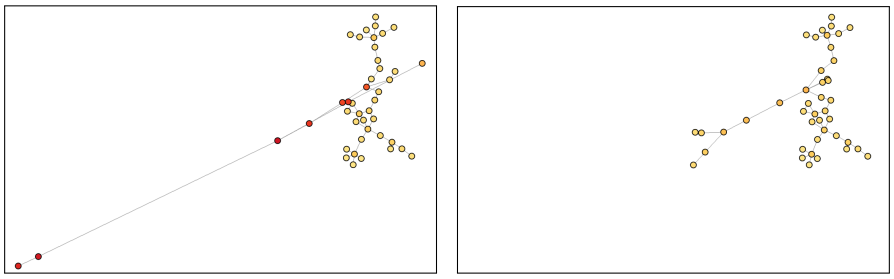
where $\vec{u}$ and $\vec{v}$ represent the positions of node u and v, and $\|\vec{u} - \vec{v}\|$ is the norm of node u and v. C is the repulsive constant and dl is the desired length between two nodes. In practice, we found that $C = 4.0$ and $dl = 0.055$ work well, but these parameters can be modified as needed.

The force calculation is modeled as $\vec{F}^{total} = \vec{F}^{rep} + \vec{F}^{spring}$. The repulsive calculation is computed for all node pairs, whereas the attractive calculation is computed only for node pairs connected by an edge.

Animation: Animation is employed to display the graph changes between L_{i-1} to L_i. Existing nodes smoothly transition into their new positions from L_{i-1} to L_i. New nodes do not exist in L_{i-1} and must be introduced into L_i. By default, we use Graph Diaries [3], an animation mode that uses different stages to emphasize graph changes, such as deletion, movement, and addition.

3.2 Refinement Method

We introduce our refinement scheme, which consists of two phases. The first phase detects and marks nodes with high energy and the second phase allows those marked



(a) (b)

Figure 3: Energy levels are mapped from yellow to red, where red represents high energy. Refinement allows only nodes with high energy (a) to move until they reach a low energy state, which is represented in yellow (b).

nodes to move. As previously mentioned, force-directed layouts are built on top of a physical model. The physical model attempts to move graph nodes towards a state of low energy: the lower the energy, the higher the layout quality will be.

Energy and force are related by $\vec{F} = \nabla En$ [17]; we can compute energy by integrating the force, $\vec{F}$. Given a force model and two nodes, positioned at $\vec{u}$ and $\vec{v}$, the repulsive energy is calculated by

$$En^{rep} = \frac{-C}{\|\vec{u}-\vec{v}\|} \tag{3}$$

The spring energy is calculated by

$$En^{spring} = \frac{1}{9} \cdot (\|\vec{u}-\vec{v}\|^3 \cdot (log\left(\frac{\|\vec{u}-\vec{v}\|}{dl}\right) - 1) + dl^3) \tag{4}$$

The total energy for node $\vec{v}$ is computed by summing over all edges connected to $\vec{v}$ and all $\vec{v}$ and $\vec{u}$ node pairs: $En(v) = En^{rep} + En^{spring}$.

$$En(v) = \sum_{u,v \in V, u \neq v} \frac{-C}{\|\vec{u}-\vec{v}\|} + \sum_{u:(u,v) \in E} \frac{1}{9} \cdot (\|\vec{u}-\vec{v}\|^3 \cdot (log\left(\frac{\|\vec{u}-\vec{v}\|}{dl}\right) - 1) + dl^3) \tag{5}$$

where $\vec{u}$ and $\vec{v}$ represent the positions of node u and v, and $\|\vec{u}-\vec{v}\|$ is the norm of node u and v. Ideally, we want an approach that will gradually modify the graph by only moving a subset of the nodes. If all nodes were allowed to move to a lower energy state, the mental map would be broken. Once we quantify the energy for individual nodes, we need to determine when a node's energy is high in relation to the entire system. Every node or edge that is introduced increases the total energy of the system, making it difficult to determine the baseline of high energy. A simple approach is to subtract graph G_k from G_{k-1} to see which nodes have high energy. However, this is only conclusive for the current time step and nodes that gradually increase in energy over time will not have a large enough difference to be detected.

Instead, we use the mean of the nodes' energy, μ, and compare it with each node's energy, yielding a definition of high energy. As new nodes and edges are added, the total and mean energy will increase. We define a node to have high energy when $\frac{abs(En(v)-\mu)}{\mu} > K$, where K is a user-defined constant and En(v) is the energy of a particular node v. In our implementation, we define K to be 1.

To move the nodes in the second phase, we compute the layout for the finest level of the graph. Although the original graph, G_0, does not leverage the multilevel algorithm, we run the layout step for a subset of the graph that has been marked to have high energy. A demonstration of the concept is shown in Figure 3. In addition, refinement runs the layout step for a fixed number of iterations. This is a adjustable parameter; reducing the number of steps trades quality for speed. In our implementation, we have set this number to 20. We also modify the temperature factor to anneal nodes to their final positions. This factor affects the mental map's quality [38] and complements our force model. In our system, we set temperature to 1.0.

Refinement can be applied in two ways: independently from or integrated into the layout method. Independent refinement runs between layout calculations and repeats at regular intervals until the next layout calculation starts with newly-read data. Integrating refinement into the layout method involves applying the first phase of refinement, after the layout method has marked nodes, but before the layout calculation. In this case, not only are new nodes and their neighbors marked to move, as discussed in Section 3, but also nodes with high energy are included. We expect that refinement is best used when it runs independently from the layout method. By using multiple and smaller layout calculations, independent refinement has more opportunities to shift high energy nodes to a lower energy state. Another benefit of applying refinement independently is that it makes refinement a viable option for existing layout methods, as the layout methods do not have to be modified to run refinement. If a layout method does not have sufficient time between time steps, then integrated refinement is preferred. However, such integrated refinement has limited opportunities to fix the graph, as it is only called once before the main layout algorithm is executed.

Naively calculating the energy for nodes takes $O(N^2+E)$ time, where N is the number of nodes and E is the number of edges. The cost comes from calculating the energy for all pairs of nodes. Computing a single iteration of FM^3 is $O(N \cdot log(N)+E)$. When comparing the two costs, the computation of energy is more expensive. By leveraging FM^3's multi-pole method, we can reduce refinement's energy calculation by approximating energy to achieve the same $O(N \cdot log(N) \cdot E)$ cost. The exact measurement of energy is not needed, as the refinement algorithm is only applied to nodes that have high energy. Since FM^3 uses a kd-tree for traversal [19], this adds another $O(N \cdot log(N))$ cost for the multi-pole estimation, but for large N the estimation will be faster.

4 Evaluation

In this section, we evaluate our layout method visually and use a series of metrics to examine the stability, quality, and speed of our layout method for comparison with existing methods. We apply our refinement technique to these methods to show the benefits of relieving high energy areas when nodes are placed in suboptimal positions.

We discuss the details of the metrics used to characterize the graphs' state. We use a combination of real and synthesized data sets that vary in both size and number of time steps.

4.1 Layout Methods

We evaluate our layout method by comparing it with two advanced online dynamic layout methods, which we refer to as "pinning", by Frishman and Tal [17], and "aging", by Gorochowski et al. [21]. For pinning, Frishman and Tal modified their previous work on static layout methods [16], which was similar to FM^3. Pinning reduces layout calculations by allowing recently updated nodes and their neighbors to move. Nodes are assigned a pinning weight that determines the nodes' range of movement. In addition, neighbors of the newly added nodes are allowed to move, based on their distance-to-modification, or the number of hops, from the newly added node. This process is done by applying a breadth-first-search-like algorithm that places new nodes in a D_0 node set, then traverses the edge list to create D_1 set. This is repeated until D_{max} sets have been created. Pinning also adds a node, called the center of gravity, which pulls all nodes toward it. This ensures that nodes and small components will not drift far from the center. Aging introduces an "aging factor" that quantifies a relationship between the node's age and how much of its immediate neighborhood has changed over time. Nodes that are younger, that is, experience a large amount of change around them, have a larger range of movement. They also have an aging rate that determines how fast nodes age per time step. We could not find existing implementations of these algorithms, so we implemented them according to their respective papers.

While implementing pinning, we noticed that a few pieces of information are missing. We have made several assumptions in efforts to replicate the paper's results and provide a fair comparison against our layout method.

For example, the paper does not state how node movement is affected when an edge is added between existing nodes. Although they are assigned to the D_0 node set, existing nodes do not move because their assigned pinning weight is high. As a result, these nodes do not properly transfer movement to their neighbors. Neglecting this case increases the likelihood of edge crossings, which degrade the graph's readability over time. Instead, we have decided to assign pinning weights of zero to these nodes, allowing these possibly distant components to move closer to minimize edge crossings.

Additionally, with initial node placement, the paper does not specify how far new nodes should be placed from their reference points. When new nodes with no edges are added, we place them away from the center at a distance based on the relationship between the central node's attractive forces and other nodes' repulsion forces. We have included a scaling factor to modulate this calculated distance because graph layouts' bounding boxes vary in size. When a new node v is connected to a positioned node u, we place node v at a distance dl away from u on a line towards the center node. In our implementation, the central node is positioned at (0,0) and is the bounding box's center when calculating the forces. Since the attractive forces for the central node are not mentioned, we picked a magnitude of 0.02 which best replicates their results when using our force model.

Frishman and Tal's algorithm stops coarsening after four levels or "several" hundred nodes. We used 200 for our implementation. Also, their distance-to-modification pass has a cutoff at D_{max}, which we set to 6 for the first four data sets and 4 for the remaining three data sets.

Since aging can be applied to any incremental layout, we have decided to incorporate it into our method. To do so, we have made assumptions on what the aging rate is for our data sets. The aging rate is used to tune the trade-offs between graph structure and evolutionary changes. We set the aging rate as 0.5 in our implementation across all our data sets.

4.2 Data sets

We use seven data sets with varying size and velocity. Throughout these graphs' evolution, there are multiple instances of addition and deletion of nodes and edges. As a result, some data sets' sizes grow, while others stay the same.

The first data set is taken from McFarland's study [33] which documents student interaction in a classroom. The visualization of this graph shows clusters that expand, shrink, and split over time. The network's edges appear and disappear, reflecting the nature of interactions among the classrooms involved in this study. This data set is our smallest graph, with 20 nodes and 82 time steps. We use the McFarland data set for direct comparison with pinning and aging algorithms since their results are shown in Gorochowski et al.'s work.

The second and third data sets are from Stack Overflow, a forum where individuals post questions about programming. Users not only answer questions, but also provide feedback to the questions and supplied answers. Users are rewarded points when they post popular questions, answers, or comments. The first Stack Overflow data set is a one-month trial run of the collection in November 2014. The data set starts with 304 nodes and 606 edges and expands to 4,000 nodes and 5,000 edges. The second data set, Stack Overflow Live, is a live feed of the site. At the time of the recording, the data set started with 80 nodes and 80 edges and ended with 638 nodes and 964 edges. Both data sets are characterized by many small, independent components that merge together over time. In the Stack Overflow data sets, nodes and edges do not disappear because all user activity is archived.

The Facebook, HepPh, and YouTube data sets are acquired from a website, which hosts collections of streaming graph data sets [47]. Visualization of the graphs is shown in Figure 4(a, b, c). The Facebook data set starts with 822 nodes and 1,160 edges and expands to 1,268 nodes and 2,004 edges. The data set focuses on new or lost connections between individuals. The data set, an example of a small world graph, is characterized by one large cluster and many small clusters. The HepPh data set is a citation network collected from `arxiv.org` on high energy physics phenomenology. It starts at 21,716 nodes and 284,710 and expands to 21,874 nodes and 289,386 edges. The Youtube data set is a social network where a link is formed when a user subscribes to another user. The network has a large number of nodes that are connected by a single edge. The network starts at 35,209 nodes and 108,286 edges and grows to 36,354 nodes and 110,996 edges. Both HepPh and YouTube experience only the addition of nodes and edges to their network.

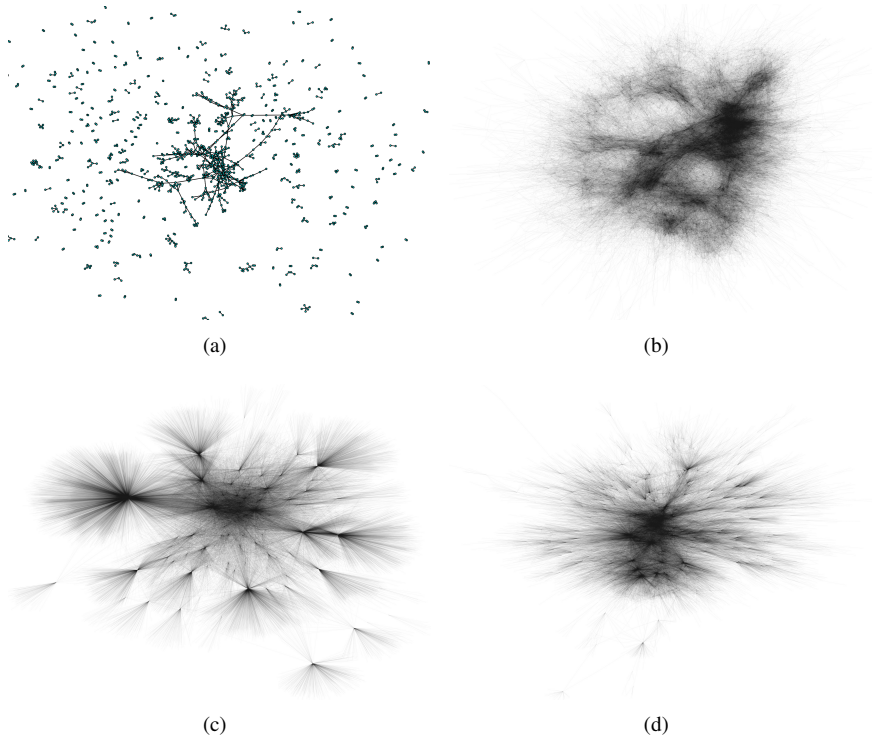


Figure 4: Visualization of the larger data sets: (a) Facebook, (b) HepPh, (c) YouTube, and (d) Internet. For HepPh, YouTube, and Internet nodes were not rendered to improve readability. The Facebook data set has a single large component and many small ones. The other data sets all have a single large component, but vary in their structure. The YouTube data set stands out amongst the three graphs with a large number of star burst patterns, where many nodes have a single connection to the same node.

The last data set is a communication network of routers that comprise the internet that was collected in 2001 and spans over two months. The graph grows and shrinks from 34,379 nodes and 143,648 edges to 34,395 nodes and 143,784 edges. The data set is visualized in Figure 4d. Unlike the first four data sets, these last three rarely have distant component merging. This is due to the fact that each of these data sets contains one large component, and any additions to the graph connect to this large component.

4.3 Metrics

Stability, quality, and timing metrics are used to assess the effectiveness of the different layouts. These metrics are the same as the ones used by Frishman and Tal [17] and Gorochoski et al. [21] in their comparison study. Stability is synonymous to the

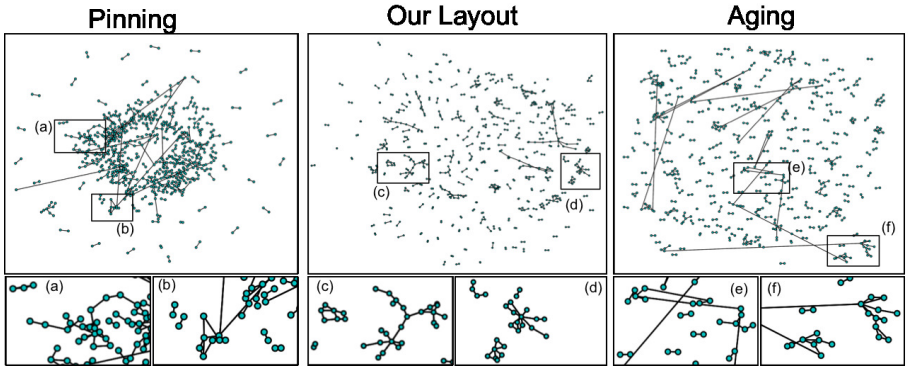


Figure 5: Visualization of the Stack Overflow-Live data using pinning, our layout method with independent refinement, and aging at the same instance. Pinning tends to have nodes near the center due to its central attractive force, whereas aging and our layout have nodes spread out across the viewing space. Pinning and aging generate long edges and edge crossings (a,b,e,f)–characteristics which degrade the graph over time. With refinement, our method relieves this problem by shifting parts of the graph to lower the system's energy (c,d).

preservation of the mental map. Stability measures the amount of change in a graph by quantifying the change in position for all nodes or the distance a node moved. A new node's change in position is zero in the time step when it is first introduced

Timing is measured before and after every layout computation call. We use the average time across layout computations to assess the speed of layout methods. The speed of our refinement technique is difficult to measure because it runs when the layout is waiting for new data. Therefore, it is not part of the layout step and can be considered free as it is not taking away from the computation of the layout.

Selecting a quality metric to evaluate dynamic layouts is difficult. There have been few studies looking at the importance of preserving the mental map in dynamic layouts [37, 38]. We define quality as the measurement of energy, where low energy produces aesthetically pleasing graphs–nodes are placed at optimal edge lengths from each other, making the graph's structure easy to comprehend. We use the total energy of the system to match the metrics used by Frishman and Tal [17] and Gorochowski et al. [21]. Since our refinement technique uses our energy model to determine which nodes have high energy, we simply sum the energy for all nodes:

$$En^{total} = \sum_{i=1...n} En(i) \tag{6}$$

where n is the total number of nodes.

To ensure fair comparisons of layout quality, all layout implementations use the same force model. Aging naturally uses our force model, since it is built upon our layout method. Our pinning implementation uses our spring-system force model.

Data sets	Metrics	**Energy**	**Δ pos**	**Time**
McFarland	Pinning	1,584	2.48	0.020
	Aging	**25.12**	0.7470	0.021
	Our Layout	1,159	**0.745**	**0.008**
Stack Overflow-Live	Pinning	137,651	**119**	0.067
	Aging	546,130	272	0.061
	Our Layout	**24,764**	350	**0.059**
Stack Overflow	Pinning	1,457k	**151**	**0.084**
	Aging	113,188k	658	0.085
	Our Layout	**862k**	658	**0.084**
Facebook	Pinning	14,803k	**308**	0.208
	Aging	186,310k	1,019	**0.131**
	Our Layout	**9,724k**	3,042	0.133
HepPh	Pinning	311,468k	652	1.121
	Aging	496,579k	**224**	0.474
	Our Layout	**52,150k**	708	**0.468**
YouTube	Pinning	1,719,475k	2,838	1.846
	Aging	**378,747k**	**860**	**0.679**
	Our Layout	393,727k	1,184	0.689
Internet	Pinning	8,077,376k	1,012	2.254
	Aging	316,010k	**122**	2.550
	Our Layout	**278,588k**	307	**0.543**

Table 1: Comparison of layout methods using energy, Δ position, and time. Lowest quantities are in bold. Results are the average of these metrics, characterizing the graphs' state throughout the observed session. Energy is the total energy in the system, Δ position is the change in nodes' position, and time is measured in seconds.

4.4 Analysis of Our Layout Method

The results of our study are given in Table 1, Table 2, Figure 4, Figure 5, Figure 6, and `http://vis.cs.ucdavis.edu/Videos/Incr.mp4`.

The evaluation is conducted on a Macbook Pro laptop. It has an NVIDIA GeForce GT 750M graphics card, a 2.3 GHz Intel Core i7 processor, and 16 gigabytes of RAM.

Quality, Stability, and Timing Comparisons: Table 1 is the quantitative comparison among our layout method, pinning, and aging. Figure 5 shows a visual comparison of the three layouts for the Stack Overflow Live data set. In the pinning results, a distinct ring of nodes forms. The ring is a consequence of the pinning implementation, which places new nodes with no edges around this ring. Nodes are spread out in aging and our layout method because nodes are placed randomly inside the graph's bounding box.

From Table 1, we can see in most cases our layout has the lowest energy. We observe around 1.5 to 29 times improvement over pinning and 1.3 to 133 times for aging.

Data sets	Metrics	**Energy**	**Δ pos**	**Time**
McFarland	Pinning	1,584	**2.48**	**0.020**
	Pinning+ref	**671**	6.92	**0.020**
Stack Overflow-Live	Pinning	137,651	**119**	**0.067**
	Pinning+ref	**42,943**	294	0.082
Stack Overflow	Pinning	1,457k	**151**	**0.084**
	Pinning+ref	**43.8k**	317	0.124
Facebook	Pinning	14,803k	**308**	**0.208**
	Pinning+ref	**76.6k**	409	0.231
HepPh	Pinning	311,468k	**652**	1.121
	Pinning+ref	**82,765k**	1135	**1.109**
YouTube	Pinning	1,719,475k	**2,838**	**1.846**
	Pinning+ref	**1,587,272k**	4,384	2.31
Internet	Pinning	8,077,376k	**1,012**	**2.254**
	Pinning+ref	**517,841k**	2,382	2.733

Table 2: Comparison of pinning with and without refinement, using energy, Δ position, and time to measure the performance. Lowest quantities are in bold. Results characterize the graphs' state throughout the observed session. Energy is the total energy in the system, Δ position is the change in nodes' position, and time is measured in seconds.

The low energy is attributed to the layout gradually repairing itself. This is apparent visually, where our layout method better handles merging of distant components than the other two methods. Our layout reduces long edges, whereas these problems are evident in the other two layouts due to their high energy.

In our experiments, a layout's stability is analyzed using an average change in position. There is a clear difference in how the algorithms behave among the smaller and larger data sets. The pinning layout has the smallest average Δ position on small data sets, while aging has the smallest average Δ position on the larger data sets. In most cases, our layout has a higher Δ position because nodes are shifting into a better position to reduce energy, therefore trading stability for lower energy. Aging also suffers from a large Δ position in the smaller data sets. As we noted in Section 4.2, the last three data sets have few to no distant components merging. This difference plays a large role in how well the layout methods do in stability. For example, aging takes a large penalty when distant components merge. In all accounts, our refinement technique increases node movement in favor of gradually fixing the graph, which is evident in Figure 5.

In our implementation of the layout methods, we found there is little difference in speed when computing layouts on the small graphs. As graphs become larger, differences start to arise. Pinning takes the longest, while aging and our layout are three times faster. Pinning's longer time is likely attributed to its distance-to-modification component, where the number of nodes it visits increases rapidly as the size of the graph and node degree increase. Since aging uses our incremental layout code, it is not

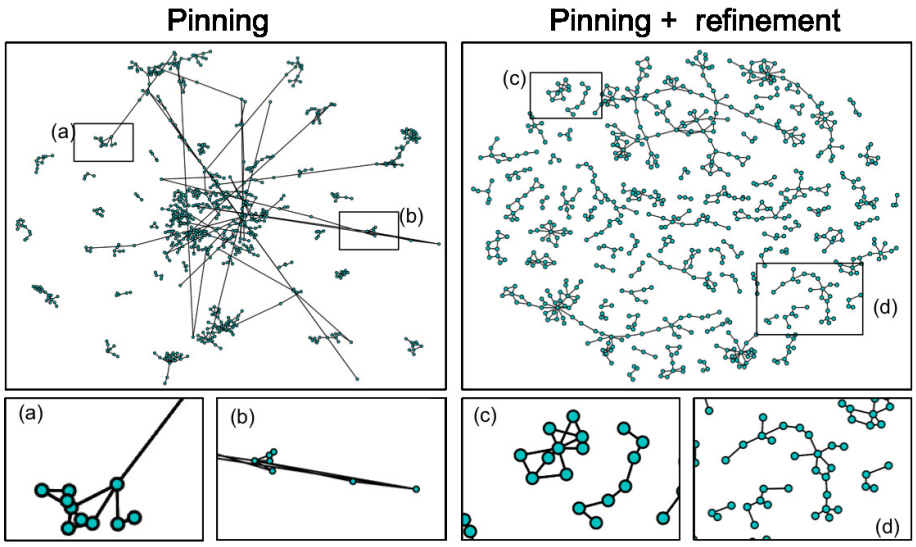


Figure 6: Visualization of the Stack Overflow data, comparing pinning and pinning with independent refinement at the same instance. Many of the same trends found in Figure 5 are observed in this visualization. Pinning suffers from long edges and edge crossings (a,b), which are fixed when refinement is added (c,d).

a surprise it attains times similar to those attained by our method. Aging benefits from the performance gains our layout method provides.

It is interesting to see aging performing best on the YouTube data set. We can see YouTube has a different structure than the rest of the large data sets, shown in Figure 4. There seems to be a strong preferential attachment in the data set, showing that "the rich get richer." New nodes are likely connected to nodes with high degree and are placed near to their ideal placement. Therefore, the refinement step does not have as many opportunities to flag high energy nodes.

Refinement with Pinning: Table 2 shows the results of applying our refinement technique to pinning. With refinement, pinning has 1.2 to 200 times lower energy. As expected, the refinement version takes longer to calculate than pinning by itself. However, this extra time is negligible, as refinement is meant to run while the layout is idle. Similar to Table 1, pinning with refinement has a higher change in position than pinning alone has. From Figure 6, we can see that the extra movement is used to fix long edges and spread out nodes.

Figure 6 shows the benefits of our refinement technique. We can see that long edges and edge crossings are less evident in the images on the right of the figure. The added benefit is that refinement helps spread out the nodes in each component, making it easier to see the structure.

5 Discussion

The previous layout methods used in our evaluation have unique benefits. Pinning maintains graph stability using pinning weights to restrict node movement. Aging provides an anchor point for graph exploration by moving nodes based on their evolutionary changes. However, our layout algorithm places nodes at their optimal positions by considering each node's energy. Our refinement technique identifies high energy components in the graph and reduces the system's energy by gradually moving nodes to a lower energy state. Our results show that our refinement technique can be used to improve existing layout methods with respect to both layout quality and readability, creating graph drawings that best visualize the relations among involved entities.

5.1 Scalability and Performance

Scalability of an incremental layout method depends on the volume and velocity of the incoming data. Volume is the amount of incoming data at a time and velocity is the rate of new data flowing in. Incremental layouts are efficient when a small portion of the graph changes due to the introduction of new nodes and edges. Although the computation time naturally increases when the volume of data increases, incremental layouts scale better than conventional layouts. This improved scaling comes from the reduced amount of work the incremental layout has to do before computing the force calculation. For example, the incremental layout only calculates one kd-tree per level and does not have to calculate the forces for all levels. Incremental layouts are preferable, as long as this overhead is low. By using our incremental layout method with our refinement technique, we find promising potential in its scalability, in terms of quality, stability, and speed, which is a result of reducing the system's energy when possible.

Visual scalability is dependent on the data set and tasks performed. As dynamic graphs are an extension of static graphs, any visual scalability problems found in static graphs will occur in dynamic graphs. For example, as data size increases, tracking individual nodes becomes more difficult. If the task is to generate an overview of the graph, then the size plays a smaller role over the graph's structure. We can see this demonstrated in Figure 4, where each of the larger data sets has a unique structure. Animation is helpful when the graph remains small in size. Bach et al. [3] suggest to aggregate the data when the graph becomes too large.

Our layout method is written as an independent library, which can be quickly ported into other graph drawing systems. The library is easy to thread; the rendering and animation stages run separately from the layout calculation. However, the downside is that our current implementation spends a portion of time converting the input into the proper GPU format for the library. This does not invalidate our timing results because all three methods were implemented with the same pipeline. A closer coupling of the GPU to the data can be leveraged for faster loading and formatting of the graph, but at the risk of being unable to easily port and thread the library.

5.2 How Refinement Works

To show how refinement works, we compare the energy of the graph with and without refinement. We chose to compare three levels of refinement that use three, five, or ten refinement steps between each layout step. The resulting plots are shown in Figure 7. The y-axis is the total energy of the graph in log scale and x-axis is the number of layout steps. The blue line is the computation of the layout without refinement and green is with refinement. In particular, for the refinement case, each line contains dots where larger dots represent a typical layout computation and small dots represent an incremental computation.

We can see a gradual increase in energy for the computation without refinement steps, shown in Figure 7. It is interesting to note that the energy dips in some of the time steps. These dips are attributed to the placement of nodes as part of the incremental layout. Nodes with no edges are moved to their newly connected nodes and thus can be placed in a more optimal position. This is also true for any nodes that have been flagged to move, providing another chance to reduce the energy.

The plots show that with each step of refinement, the energy goes down. This trend continues until a layout step is done, when energy shoots up. The large spikes that we see in particular time steps, for example in 8, 16, and 18, are attributed to distant components merging. What happens is that two forces are acting on a node: the repulsive and spring force. Spring force is polynomial, while repulsive force is linear. Large energies are produced when either connected nodes are too close–thus having both the repulsive and spring force acting on them–or are far away.

When comparing different refinement steps between the layout steps, we can see a diminishing return. This is much more apparent in Figure 7c, for the plot where ten refinement steps were used. In this plot, not all sections between the layout step exhibit a diminishing return. There seems to be a correlation between how large an energy jump is after the layout step and diminishing returns. If there is a large increase in energy, more refinement steps are necessary to reduce newly introduced energy. We conclude that there is an energy baseline; energy cannot decrease below this threshold. If the energy increase is small per layout, the new energy is closer to the baseline and thus will have a higher diminishing return.

This makes choosing the correct amount of refinement steps between the layout steps difficult even for the same graph. In general, having more refinement steps is beneficial. It can improve other parts of the graph, since refinement works on the entire graph and not just the newly introduced nodes and edges. If an interval does not have enough time to lower the energy, a future refinement step has the opportunity to do so. We can see this effect in Figure 7c (10 refinement) between layout step 12 and 16, where the energy continues to decrease until it reaches the energy baseline. In real-world applications, the data can come in at any rate. Therefore, any opportunity for refinement to run can improve the graph's quality. As we pointed out before, refinement is considered free because it runs while the layout is waiting for data to come in. Trying to estimate the amount of steps before the diminishing return occurs is not necessary. The limiting factor should be data velocity and not a user-defined parameter.

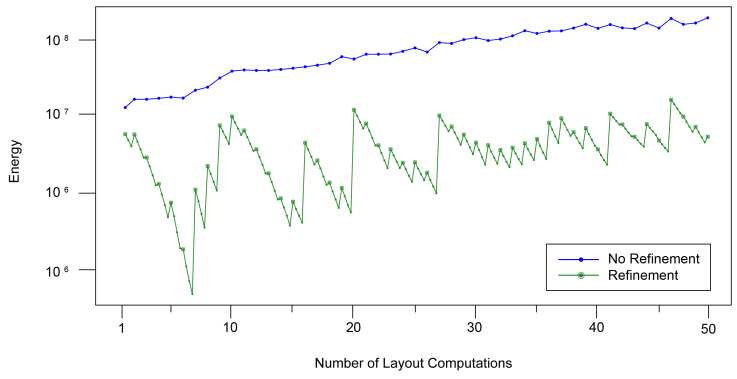
6 Conclusion

We have presented an incremental layout method and a refinement technique for visualizing online dynamic graphs that are used to create stable and aesthetic pleasing layouts. First, we have shown how to convert FM^3 into our incremental multilevel multi-pole algorithm. Second, our refinement technique is used to identify high energy nodes and move them to a low energy state. The refinement technique can be used in tandem or separately from the layout method. Lastly, we are able to employ a GPU to accelerate the layout and refinement technique. An empirical evaluation with metrics shows that our method helps improve layout quality and readability.

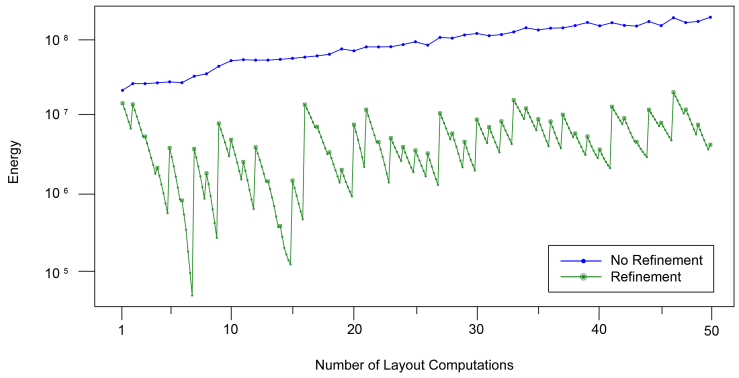
For future work, we plan to investigate how to incorporate history, that is, prior states of the graph, into online dynamic graphs. Our current method helps monitor ever-changing graphs, but provides no means for reviewing previous network changes. To improve analysis, it would be helpful to have a mechanism that allows users to explore the network's history, while still allowing the system to visualize new data. The difficulty arises in how we should store the graph's history and present it back to the user. A possible avenue of research is to store previous data in a compression format similar to that of videos. Visualizing the history can also be shown as a video or a series of small snapshots in a secondary view.

Acknowledgements

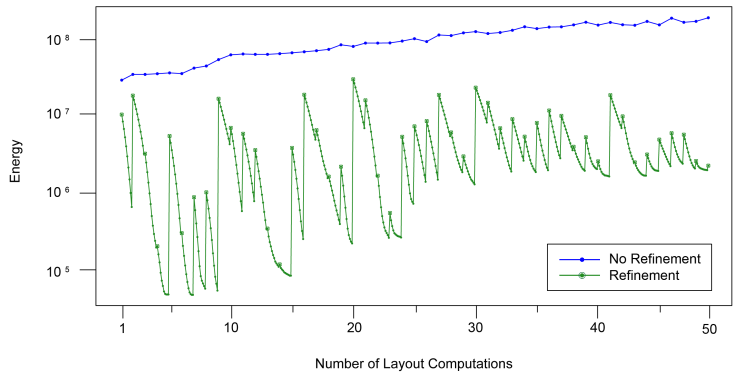
This research is sponsored in part by the U.S. National Science Foundation via grants NSF DRL-1323214 and NSF IIS-1320229, the U.S. Department of Energy through grant DE-FC02-12ER26072, and also the UC Davis RISE program.



(a)



(b)



(c)

Figure 7: The total energy in the system observed in the Facebook dataset of 50 layout steps. We observed the energy after the first layout step, which is a full FM^3 calculation. Energy is mapped to the y-axis in log scale, and the x-axis displays the number of layout computations. Both lines represent the energy produced by our incremental layout, namely that green lines show when our refinement technique is applied, and blue lines are without refinement. The three graphs show the energy for intervals 3(a), 5(b), and 10(c), which is indicative of the amount of refinement computation steps. The energy in a layout without refinement tends to gradually increase, whereas with our refinement technique, there is a noticeable decrease of energy after each energy spike.

References

[1] D. Archambault, H. C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Trans. Vis. Comput. Graph.*, 17(4):539–552, 2011. doi:10.1109/TVCG.2010.78.

[2] D. Auber and Y. Chiricota. Improved efficiency of spring embedders: Taking advantage of GPU programming. In *The Seventh IASTED International Conference on Visualization, Imaging and Image Processing*, VIIP '07, pages 169–175. ACTA Press, 2007.

[3] B. Bach, E. Pietriga, and J. Fekete. Graphdiaries: Animated transitions andtemporal navigation for dynamic networks. *IEEE Trans. Vis. Comput. Graph.*, 20(5):740–754, 2014. doi:10.1109/TVCG.2013.254.

[4] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The State of the Art in Visualizing Dynamic Graphs. In *EuroVis - STARs*. The Eurographics Association, 2014. doi:10.2312/eurovisstar.20141174.

[5] C. Bennett, J. Ryall, L. Spalteholz, and A. Gooch. The aesthetics of graph visualization. In D. W. Cunningham, G. W. Meyer, L. Neumann, A. Dunning, and R. Paricio, editors, *Computational Aesthetics 2007*, pages 57–64. Eurographics Association, 2007.

[6] K. Boitmanis, U. Brandes, and C. Pich. Visualizing internet evolution on the autonomous systems level. In S. Hong, T. Nishizeki, and W. Quan, editors, *15th International Symposium on Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 365–376. Springer, 2007. doi:10.1007/978-3-540-77537-9_36.

[7] U. Brandes, D. Fleischer, and T. Puppe. Dynamic spectral layout with an application to small worlds. *Journal of Graph Algorithms and Applications*, 11(2):325–343, 2007. doi:10.7155/jgaa.00149.

[8] U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In M. J. van Kreveld and B. Speckmann, editors, *19th International Symposium on Graph Drawing, GD 2011*, volume 7034 of *LNCS*, pages 99–110. Springer, 2011. doi:10.1007/978-3-642-25878-7_11.

[9] U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In G. Di Battista, editor, *5th International Symposium on Graph Drawing, GD '97*, volume 1353 of *LNCS*, pages 236–247. Springer, 1997. doi:10.1007/3-540-63938-1_66.

[10] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2344–2353, 2011. doi:10.1109/TVCG.2011.226.

[11] C. S. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In S. Diehl, J. T. Stasko, and S. N. Spencer, editors, *ACM 2003 Symposium on Software Visualization*, pages 77–86, 212–213. ACM, 2003.

[12] T. Crnovrsanin, J. Chu, and K. Ma. An incremental layout method for visualizing online dynamic graphs. In E. Di Giacomo and A. Lubiw, editors, *23rd International Symposium on Graph Drawing and Network Visualization, GD 2015*, volume 9411 of *LNCS*, pages 16–29. Springer, 2015. doi:10.1007/978-3-319-27261-0_2.

[13] S. Diehl and C. Görg. Graphs, they are changing. In S. G. Kobourov and M. T. Goodrich, editors, *10th International Symposium on Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 23–30. Springer, 2002. doi:10.1007/3-540-36151-0_3.

[14] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. GraphAEL: Graph animations with evolving layouts. In G. Liotta, editor, *11th International Symposium on Graph Drawing, GD 2003*, volume 2912 of *LNCS*, pages 98–110. Springer, 2003. doi:10.1007/978-3-540-24595-7_9.

[15] Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In M. O. Ward and T. Munzner, editors, *10th IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 191–198. IEEE Computer Society, 2004. doi:10.1109/INFVIS.2004.18.

[16] Y. Frishman and A. Tal. Multi-level graph layout on the GPU. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1310–1319, 2007. doi:10.1109/TVCG.2007.70580.

[17] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Trans. Vis. Comput. Graph.*, 14(4):727–740, 2008. doi:10.1109/TVCG.2008.11.

[18] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. *Comput. Geom.*, 29(1):3–18, 2004. doi:10.1016/j.comgeo.2004.03.014.

[19] A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Rapid multipole graph drawing on the GPU. In I. G. Tollis and M. Patrignani, editors, *16th International Symposium on Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 90–101. Springer, 2008. doi:10.1007/978-3-642-00219-9_10.

[20] C. Görg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In J. Pach, editor, *12th International Symposium on Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 228–238. Springer, 2004. doi:10.1007/978-3-540-31843-9_24.

[21] T. E. Gorochowski, M. di Bernardo, and C. S. Grierson. Using aging to visually uncover evolutionary processes on networks. *IEEE Trans. Vis. Comput. Graph.*, 18(8):1343–1352, 2012. doi:10.1109/TVCG.2011.142.

[22] M. Greilich, M. Burch, and S. Diehl. Visualizing the evolution of compound digraphs with TimeArcTrees. volume 28, pages 975–982, 2009. doi:10.1111/j.1467-8659.2009.01451.x.

[23] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *J. Comput. Physics*, 227(18):8290–8313, 2008. doi:10.1016/j.jcp.2008.05.023.

[24] S. Hachul and M. Jünger. Large-graph layout algorithms at work: An experimental study. *Journal of Graph Algorithms and Applications*, 11(2):345–369, 2007. doi:10.7155/jgaa.00150.

[25] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002. doi:10.7155/jgaa.00051.

[26] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004. doi:10.7155/jgaa.00089.

[27] A. Hayashi, T. Matsubayashi, T. Hoshide, and T. Uchiyama. Initial positioning method for online and real-time dynamic graph drawing of time varying data. In *17th International Conference on Information Visualisation, IV 2013*, pages 435–444. IEEE Computer Society, 2013. doi:10.1109/IV.2013.57.

[28] Y. Hu, S. G. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In H. Hauser, S. G. Kobourov, and H. Qu, editors, *2012 IEEE Pacific Visualization Symposium, PacificVis 2012*, pages 33–40. IEEE Computer Society, 2012. doi:10.1109/PacificVis.2012.6183571.

[29] M. Khoury, Y. Hu, S. Krishnan, and C. E. Scheidegger. Drawing large graphs by low-rank stress majorization. *Comput. Graph. Forum*, 31(3):975–984, 2012. doi:10.1111/j.1467-8659.2012.03090.x.

[30] Y. Koren, L. Carmel, and D. Harel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling & Simulation*, 1(4):645–673, 2003. doi:10.1137/S154034590241370X.

[31] G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. *IEEE Trans. Vis. Comput. Graph.*, 12(5):805–812, 2006. doi:10.1109/TVCG.2006.193.

[32] C. Lin, Y. Lee, and H. Yen. Mental map preserving graph drawing using simulated annealing. *Inf. Sci.*, 181(19):4253–4272, 2011. doi:10.1016/j.ins.2011.06.005.

[33] D. A. McFarland. Student resistance: How the formal and informal organization of classrooms facilitate everyday forms of student defiance. *American Journal of Sociology*, 107(3):612–678, 2001. doi:10.1086/338779.

[34] 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.

[35] C. Muelder and K. Ma. Rapid graph layout using space filling curves. volume 14, pages 1301–1308, 2008. doi:10.1109/TVCG.2008.158.

[36] S. C. North. Incremental layout in DynaDAG. In F. Brandenburg, editor, *Symposium on Graph Drawing, GD '95*, volume 1027 of *LNCS*, pages 409–418. Springer, 1995. doi:10.1007/BFb0021824.

[37] H. C. Purchase, E. E. Hoggan, and C. Görg. How important is the "Mental Map"? - an empirical investigation of a dynamic graph layout algorithm. In M. Kaufmann and D. Wagner, editors, *14th International Symposium on Graph Drawing, GD 2006*, volume 4372 of *LNCS*, pages 184–195. Springer, 2006. doi:10.1007/978-3-540-70904-6_19.

[38] H. C. Purchase and A. Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In G. Stapleton, J. Howse, and J. Lee, editors, *Diagrammatic Representation and Inference: 5th International Conference, Diagrams 2008*, volume 5223 of *LNCS*, pages 60–73. Springer, 2008. doi:10.1007/978-3-540-87730-1_9.

[39] A. J. Quigley and P. Eades. FADE: graph drawing, clustering, and visual abstraction. In J. Marks, editor, *8th International Symposium on Graph Drawing, GD 2000*, volume 1984 of *LNCS*, pages 197–210. Springer, 2000. doi:10.1007/3-540-44541-2_19.

[40] K. Reda, C. Tantipathananandh, A. E. Johnson, J. Leigh, and T. Y. Berger-Wolf. Visualizing the evolution of community structures in dynamic social networks. *Comput. Graph. Forum*, 30(3):1061–1070, 2011. doi:10.1111/j.1467-8659.2011.01955.x.

[41] A. Sallaberry, C. Muelder, and K. Ma. Clustering, visualizing, and navigating for large dynamic graphs. In W. Didimo and M. Patrignani, editors, *20th International Symposium on Graph Drawing, GD 2012*, volume 7704 of *LNCS*, pages 487–498. Springer, 2012. doi:10.1007/978-3-642-36763-2_43.

[42] V. Sarin, A. Grama, and A. H. Sameh. Analyzing the error bounds of multipole-based treecodes. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1998*, page 19. IEEE Computer Society, 1998. doi:10.1109/SC.1998.10041.

[43] M. Stock and A. Gharakhani. Toward efficient GPU-accelerated N-body simulations. In *Aerospace Sciences Meetings*. American Institute of Aeronautics and Astronautics, 2008. doi:10.2514/6.2008-608.

[44] E. R. Tufte. *Envisioning information*. Graphic Press, 1990.

[45] C. Vehlow, F. Beck, P. Auwärter, and D. Weiskopf. Visualizing the evolution of communities in dynamic graphs. *Comput. Graph. Forum*, 34(1):277–288, 2015. doi:10.1111/cgf.12512.

[46] C. Walshaw. A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2003. doi:10.7155/jgaa.00070.

[47] Y. Yao. Collection and streaming of graph datasets. http://www.eecs.wsu.edu/~yyao/StreamingGraphs.html.