An Incremental Layout Method for Visualizing Online Dynamic Graphs

Tarik Crnovrsanin(✉), Jacqueline Chu, and Kwan-Liu Ma

University of California, Davis, USA
{tecrnovr,sjchu}@ucdavis.edu, ma@cs.ucdavis.edu

Abstract. Graphs provide a visual means for examining relation data and force-directed methods are often used to lay out graphs for viewing. Making sense of a dynamic graph as it evolves over time is challenging, and previous force-directed methods were designed for static graphs. In this paper, we present an incremental version of a multilevel multi-pole layout method with a refinement scheme incorporated, which enables us to visualize online dynamic networks while maintaining a mental map of the graph structure. We demonstrate the effectiveness of our method and compare it to previous methods using several network data sets.

Keywords: Dynamic graphs · Streaming data · Graph layout

1 Introduction

In many fields of study, from biology to chemistry to sociology, software engineering and cyber security, an essential task is to identify and understand relationships of interest among different entities. Graphs in the form of nodes and links are commonly used to represent such relations. Graph drawing is an indispensable tool for visually studying the relationships. Many techniques have been introduced for aesthetically and efficiently laying out static graphs [11,13,14,16,17], but a large class of real-world applications involve graphs that change over time [8].

Visualizing dynamic graphs is often done by animating over the sequence of graphs [3,9,10,22] or by displaying selected ones side-by-side as small multiples [25]. Finding the best way to visualize dynamic graphs remains a challenging research topic. When laying out dynamic graphs for visual analysis, the primary goal is to ensure the stability of the layout [5,10,15,18] and preserve the mental map [1,21–23].

Most previous dynamic graph algorithms address the problem of laying out offline graphs consisting of the entire sequence of graphs. With prior knowledge of the complete time sequence, we can best optimize the layout for animation and specific analysis goals [4,8,9,19]. For online, real-time monitoring or analysis applications, however, the graph is constantly updated and how the graph might change over time cannot be predicted. Making optimal layouts for such evolving graphs is an even more challenging problem, which has received limited attention

E. Di Giacomo and A. Lubiw (Eds.): GD 2015, LNCS 9411, pp. 16–29, 2015.
DOI: 10.1007/978-3-319-27261-0_2

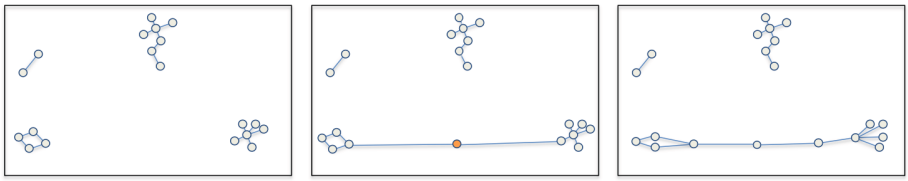


Fig. 1. An undesirable limitation. A graph has many disconnected components (a), and a node is introduced linking two components together (b). One layout method [10] allows the new node (in orange) and its neighbors to move after the new node is added. If these nodes cannot reach their ideal position 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 (Color figure online).

in existing research [6,10,12,19]. One reason is that online dynamic graph data were not readily available, but the situation has begun to change with the rapid growth of mobile, online, and real-time monitoring applications. Consequently, demands for the ability to understand online dynamic graph data have arisen in various fields.

We have examined previous online dynamic graph layout methods and found they have some undesirable limitations in layout quality or the connectivity of the graph. Some are too expensive to use for real-time applications. In order to speed up the process of laying out an online dynamic graph, a commonly employed approach is to anchor large portions of the graph and allow only a small subset of the graph to move; nevertheless, this speedup comes with several tradeoffs. One tradeoff is that when a new node or an edge is introduced, only that node and its neighbors are allowed to move at that instance. In most cases, nodes are placed near their ideal spots. If two disconnected components merge, the nodes usually cannot reach their ideal spots at once, as depicted in Fig. 1. In addition, linking disconnected graphs may lead to edge crossings. Parts of the graph would stay in suboptimal positions, unless new nodes or edges are added to the same neighborhood to allow the layout algorithm to fix this problem.

In this paper, we present an incremental version of the multilevel multi-pole layout method that is suitable for visualizing online dynamic graphs. Our work makes the following contributions to online dynamic graph drawing:

- The incremental layout method reduces the computation cost while best characterizing inherent network structure and maintaining graph readability.
- Our refinement technique reduces edge crossings and 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 is fast because our implementation for both the layout and refinement calculations are GPU-accelerated.

We evaluated our methods using several dynamic graph data sets, including ones from real-world online applications, and compared the layouts with those

produced by previous methods. The test results demonstrate the effectiveness and usability of our method.

2 Related Work

Online dynamic graphs are series of graphs in which time steps are not known ahead of time. Lee et al. [19] was one of the earliest to work with online graphs. The algorithm preserves the mental map while generating aesthetically pleasing graphs. The drawback is that the algorithm is slow, recalculating the full layout at each time step. Brandes and Wagner [6] instead used Bayesian decision theory to generate the graph. Their work characterizes the tradeoff between dynamic stability and local quality using conditional probabilities. Frishman and Tal [10] created a novel force-directed algorithm that can handle large graphs. Their GPU implementation provides a 17 times speedup over the CPU version. Gorochowski et al. [12] used 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 life. Che et al. [7] proposed a novel layout algorithm that enforces graph component shapes by using Laplacian constrained distance embedding. However, the Gorochowski et al. and Frishman and Tal algorithms do not address the disconnected component problem mentioned in the introduction.

Our layout method addresses this problem by using a novel refinement technique that gradually alleviates areas of high energy. Energy is defined as the amount of force applied to a node. In Sect. 4, our evaluation shows that our method produces more aesthetic graphs at the cost of more movement in the graph. This movement is necessary to reduce long edges and edge crossings that occur.

3 An Incremental Algorithm

Our incremental algorithm is a modified version of FM^3, which is a fast, multlievel, multi-pole, force-directed layout method. What makes FM^3 fast is that it does not calculate all the repulsion forces, which is the most expensive operation of any force-directed calculation. For a single time step, given the finest level of the graph $G = (V, E) = G_0$, FM^3 reduces computation by partitioning and collapsing G_0 until reaching a prescribed number of nodes. This subset of nodes represents the coarsest level, K. A force calculation is applied to this graph G_K, where the resulting node positions are used as the initial layout for the next finer graph, G_{K-1}. These steps are repeated until the original graph G_0 is drawn. More details of FM^3 can be found in Godiyal [11], which our GPU-accelerated implementation is based on.

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

1. Include an initial layout construction step
2. Add a merging step, which includes placing new nodes and selecting nodes to move

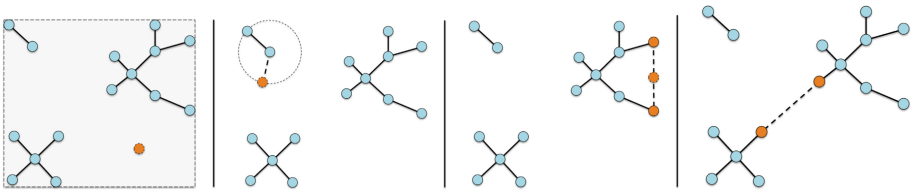


Fig. 2. The figure shows how our algorithm assigns positions to new nodes. A dashed node and edge indicate a new node and edge, respectively. Nodes colored in orange represent nodes that are flagged to move by our algorithm. (a) A node with no edges is placed randomly inside the bounding box of the graph. (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 position nodes. (d) When an edge is added or removed between two positioned nodes, our algorithm flags both nodes to move (Color figure online).

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.

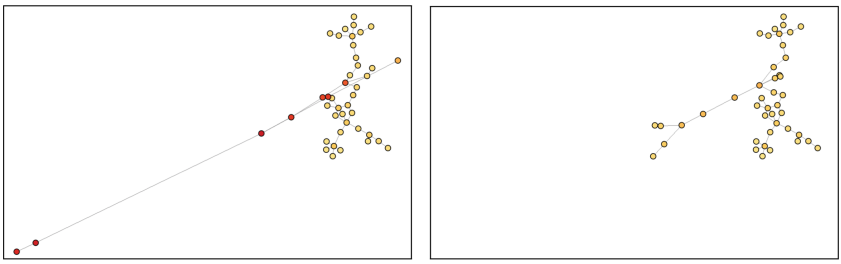
We do not modify the multi-pole calculation step. We describe each of these five steps in more detail below.

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 affixed nodes from the previous layout. Initial node placement is imperative because error is introduced when previously positioned nodes are at their 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 bounds of the graph, as shown in Fig. 2a. If v is connected to one positioned node u, v is placed randomly around u at a distance dl. dl is the desired length between two connected nodes in our spring-based energy model, as seen in Fig. 2b. If v is connected to at least two positioned nodes, v is placed at the geometric center of all the connected nodes, shown in Fig. 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 Fig. 2a, and node v is placed randomly around u at a radius of dl, equivalent to Fig. 2b. Both nodes are selected to move. Also, our method moves affixed nodes when a new edge is introduced to another node–whether new or affixed–namely, when node u is connected and node v is



(a) (b)

Fig. 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) (Color figure online).

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 Fig. 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–a single node that represents a large set of nodes from the finer levels–is done randomly or by indexing [11]. 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. A super node is selected by having the highest degree. 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 alleviates 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 done on 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, we compute the layout 250 times at the coarsest level. The number of iterations to compute the layout linearly decays per level until we reach the finest level. At the finest level, we compute the layout 30 times.

Our method uses a contribution factor to restrict the super nodes' range of movement. This solves the problem of nodes at coarser levels of the graph having greater range of movement than those at finer levels. Without the contribution factor, large disparities of movement occur due to simulated annealing. This causes suboptimal node positioning, which ultimately degrades the final graph level at G_0. The contribution factor is determined by how many nodes are allowed

to move under the super node. For instance, 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: Our repulsive forces are modeled as

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

to achieve a greater spreading of disconnected graph components. Our spring forces can be modeled as [11]

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

where C is the repulsive constant and dl is the desired length between two nodes. In practice, we found that C as 4.0 and dl as 0.055 works best with our implementation.

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 [2], an animation mode that uses different stages to emphasize graph changes, such as deletion, movement, and addition.

3.1 Refinement Method

Our method allows nodes to reposition themselves if high energy, which is characterized by long edges and edge crossings, exists between their components. This occurrence is not adequately addressed by previous methods. We minimize this effect by refining a subset of the graph which not only reduces the cost of refinement, but shortens long edges–a result of minimizing the total energy in these components.

We expect that refinement is best used when it runs independently from the layout method. However, a layout method may not have a sufficient time window to apply refinement in between time steps. A possible option is to incorporate refinement directly into the layout method. However, such integrated refinement has limited opportunities to fix the graph as it is only called once before the main layout algorithm is executed.

We describe our implementation, which makes refinement a viable option for existing layout methods. In our refinement technique, 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. In addition, refinement runs the layout step for a set number of iterations. This is an adjustable parameter, in which reducing the number of steps trades quality for speed. In our implementation, we have set this number to 20. We modify the temperature factor, defined in FM^3,

to "anneal" nodes to their final positions. This factor affects the mental map's quality [24] and complements our force model. In our system, we set temperature to 1.0.

Ideally, we want an approach that will gradually modify the graph, but only move high energy nodes. This reduces the overall energy in the system. We calculate the energy per node by deriving the relation $F = \nabla En$ [10]. Given two nodes, positioned at u and v, the repulsive energy is calculated by

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

The spring energy is calculated by

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

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

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

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 the all-pair computation. Computing a single iteration of the layout is $O(N * log(N) + E)$, making the computation of energy more expensive. In most cases, the cost of one energy computation is cheaper than the cost of computing the entire layout. It is possible to achieve the same cost for the energy computation by leveraging FM^3's multi-pole method to estimate the energy instead. Since FM^3 uses a kd-tree for traversal [11], this adds another $O(N * log(N))$ cost for the multi-pole estimation. The estimation will be faster with large graphs.

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 introduced node or edge increases the total energy of the system, making it difficult to define 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 be detected.

Instead, we take the mean of the nodes' energy, μ, and compare it against each node, yielding a definition of high energy. The mean scales with the total energy, U^{Total}, and allows us to compare the individual nodes. Thus, we define a node to have high energy when $\frac{abs(U^{Total} - \mu)}{\mu} > K$, where K is a user-defined constant. In our implementation, we define K to be 1.

4 Evaluation

In this section, we evaluate our layout method visually and use a series of metrics to examine the stability, quality and time of our layout method for comparison

Table 1. Comparison of layout methods using energy, Δ position, and time. 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.

Layout method	McFarland			Stack overflow-live			Stack overflow			Facebook		
	Energy	Δpos	Time	Energy	Δpos	Time	Energy	Δpos	Time	Energy	Δpos	Time
Pinning	1584	2.48	0.020	137651	**119**	0.067	1457k	**151**	**0.084**	14803k	**308**	0.208
Aging	**25.12**	0.747	0.021	546130	272	0.061	113188k	658	0.085	186310k	1019	**0.131**
Our Layout	1159	**0.745**	**0.008**	**24764**	350	**0.059**	**862k**	658	**0.084**	**9724k**	3042	0.133

against 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 the number of time steps.

4.1 Layout Methods

The evaluation of our layout method is done by comparing it against two advanced online dynamic layout methods called "pinning", by Frishman and Tal [10], and "aging", by Gorochowski et al. [12]. Pinning reduces calculation by allowing recently updated nodes and their neighbors to move. Nodes closer to the recently updated node have wider range of movement. Aging uses an "aging factor" that is quantified by a relationship between the node's age and how much its immediate neighborhood has changed over time. Nodes that are younger, or experience a large amount of change around them, have more freedom to move. We could not find existing implementation of these algorithms, so we implemented them according to their respective papers. Any assumptions made when implementing these methods can be found in the Appendix (http://vis.cs.ucdavis.edu/papers/tarik_incremental_appendix.pdf).

4.2 Data Sets

We use four data sets with varying size and velocity. The first data set is taken from McFarland's study [20] which documents student interaction in a classroom. The visualization of this graph shows clusters that expand, shrink, and split over time. This data set is our smallest graph, with 20 nodes and 82 time steps. We use the McFarland data set for direct comparison against 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 4000 nodes and 5000 edges. The second data set, Stack Overflow Live, is a live feed of the site.

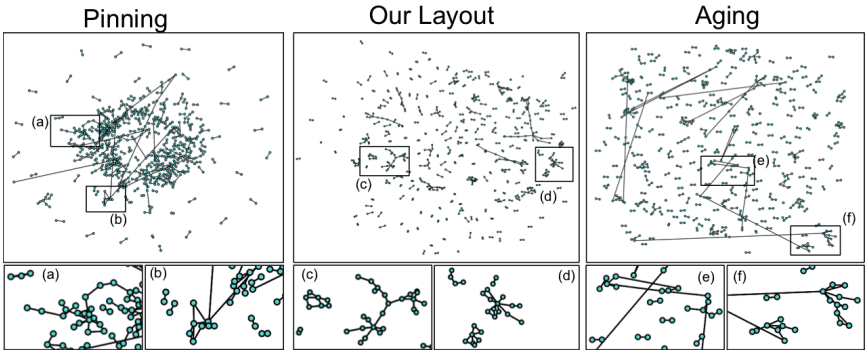


Fig. 4. Visualization of the Stack Overflow-Live data using pinning, our layout method with independent refinement, and aging at the same instance. Pinning tends to have nodes near the center due to its central attractive force, whereas aging and our layout have nodes spread out across the viewing space. 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).

At the time of the measuring for generating Table 1, 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.

The last data set is from Facebook and is acquired from a website hosting collections of streaming graph data sets [26]. This data set starts with 822 nodes and 1160 edges and expands to 1268 nodes and 2004 edges. The Facebook data set focuses on connections between individuals. The data set, an example of a small world graph, is characterized by one large cluster and many small clusters.

4.3 Metrics

Stability is synonymous to the 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. New nodes' change in position is 0 at the first time step they are 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 [23,24]. We define quality as the measurement of energy, where low energy produces aesthetically-pleasing graphs–nodes are placed at optimal

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.

Layout method	McFarland			Stack oveflow-live			Stack overflow			Facebook		
	Energy	Δpos	Time	Energy	Δpos	Time	Energy	Δpos	Time	Energy	Δpos	Time
Pinning	1584	**2.48**	**0.020**	138k	**119**	**0.067**	1457k	**151**	**0.084**	14803k	**308**	**0.208**
Pinning+ref	**671**	6.92	**0.020**	**42.9k**	294	0.082	**43.8k**	317	0.0124	**76.6k**	409	0.231

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 [10] and Gorochowski et al. [12]. Since our refinement technique uses our energy model to determine which nodes have high energy, we simply sum the energy for all nodes as such

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

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.

4.4 Analysis of Our Layout Method

The results of our study are given in Tables 1 and 2, Figs. 4 and 5, 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 amongst our layout method, pinning, and aging. Figure 4 shows a visual comparison of the three layouts for 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 bounding box.

From Table 1, we can see in most cases our layout has the lowest energy. We observe around 1.5 to 5.5 times improvement over pinning and 19 to 133 times for aging. The low energy is attributed to the layout gradually repairing itself. This translates visually, where our layout method better handles merging of distant components than the other two methods. Our layout reduces long edges or edge crossings, whereas these problems are evident in the other two layouts due to their high energy.

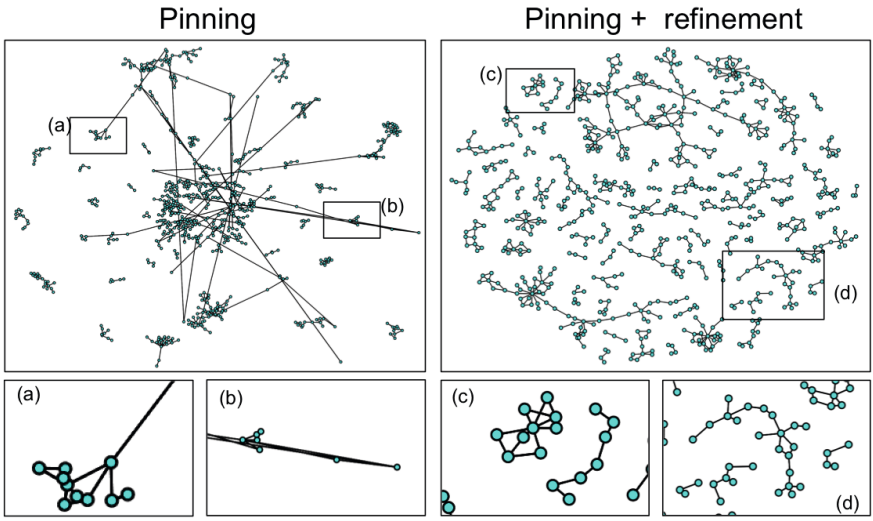


Fig. 5. Visualization of the Stack Overflow data, comparing solely pinning and pinning with independent refinement at the same instance. Many of the same trends found in Fig. 4 are observed in this visualization. Pinning suffers from long edges and edge crossings (a,b), which are fixed when refinement is added (c,d).

The layout's stability is analyzed using an average Δ position. In general, the pinning layout has the smallest average Δ position because it uses pinning weights to minimize node movement in order to produce a stable layout. Our layout has a higher Δ position because nodes are shifting into a better position to reduce energy. Aging also suffers from large Δ position. This is explained by the layout attempting to shorten long edges. In all accounts, our refinement technique increases node movement in favor of gradually fixing the graph, evident in Fig. 4.

Across the layout implementations, there are small differences in speed when computing layouts. Based on our results, additional force calculations do not necessarily increase computation time. This is likely attributed to how nodes are partitioned and bottlenecks found in the GPU. The GPU may not be fully utilized when running the force calculations. For each node, a GPU thread is created for each kd-tree leaf for force calculations. Depending on the implementation, a kd-tree can have leaves that vary in size from 4 to 20 nodes. A bottleneck occurs when the GPU is waiting for kd-tree leaf nodes that take longer to process.

Refinement with Pinning: Table 2 shows the results of applying our refinement technique to pinning. With refinement, pinning has 3 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 higher Δ position than pinning. From Fig. 5, we can see that extra movement is used to fix long edges and spread out nodes.

Figure 5 shows the benefits of our refinement technique. We can see that long edges or edge crossings are less evident on the right figure. The added benefit is that refinement helps spread out the nodes in each component, making it easier to see the structure.

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 aesthetics, creating graph drawings that best visualize the relations between involved entities.

5 Conclusion

We have presented an incremental layout method and a refinement technique for visualizing online dynamic graphs that is used to create stable and aesthetic layouts. First, we have shown how to convert FM^3 into an 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 the stability and aesthetic appeal of layouts.

Acnowledgments. 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.

References

1. Archambault, D., Purchase, H.C., Pinaud, B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Trans. Vis. Comput. Graph. **17**(4), 539–552 (2011)
2. Bach, B., Pietriga, E., Fekete, J.D.: GraphDiaries: animated transitions and temporal navigation for dynamic networks. IEEE Trans. Vis. Comput. Graph. **20**(5), 740–754 (2014)
3. Boitmanis, K., Brandes, U., Pich, C.: Visualizing internet evolution on the autonomous systems level. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 365–376. Springer, Heidelberg (2008)
4. Brandes, U., Fleischer, D., Puppe, T.: Dynamic spectral layout of small worlds. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 25–36. Springer, Heidelberg (2006)

5. Brandes, U., Mader, M.: A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 99–110. Springer, Heidelberg (2011)
6. Brandes, U., Wagner, D.: A bayesian paradigm for dynamic graph layout. In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 236–247. Springer, Heidelberg (1997)
7. Che, L., Liang, J., Yuan, X., Shen, J., Xu, J., Li, Y.: Laplacian-based dynamic graph visualization. In: Visualization Symposium (PacificVis), 2015 IEEE Pacific, pp. 69–73 (2015)
8. Diehl, S., Görg, C.: Graphs, they are changing. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 23–31. Springer, Heidelberg (2002)
9. Erten, C., Harding, P.J., Kobourov, S.G., Wampler, K., Yee, G.: Graphael: graph animations with evolving layouts. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 98–110. Springer, Heidelberg (2004)
10. Frishman, Y., Tal, A.: Online dynamic graph drawing. IEEE Trans. Vis. Comput. Graph. **14**(4), 727–740 (2008)
11. Godiyal, A., Hoberock, J., Garland, M., Hart, J.C.: Rapid multipole graph drawing on the GPU. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 90–101. Springer, Heidelberg (2009)
12. Gorochowski, T., di Bernardo, M., Grierson, C.: Using aging to visually uncover evolutionary processes on networks. IEEE Trans. Vis. Comput. Graph. **18**(8), 1343–1352 (2012)
13. Hachul, S., Jünger, M.: An experimental comparison of fast algorithms for drawing general large graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 235–250. Springer, Heidelberg (2006)
14. Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 183–196. Springer, Heidelberg (2001)
15. Hu, Y., Kobourov, S.G., Veeramoni, S.: Embedding, clustering and coloring for dynamic maps. In: Visualization Symposium (PacificVis), 2012 IEEE Pacific, pp. 33–40 (2012)
16. Khoury, M., Hu, Y., Krishnan, S., Scheidegger, C.: Drawing large graphs by low-rank stress majorization. Comput. Graph. Forum **31**(3pt1), 975–984 (2012)
17. Koren, Y., Carmel, L., Harel, D.: Drawing huge graphs by algebraic multigrid optimization. Multiscale Model. Simul. **1**, 645–673 (2003)
18. Kumar, G., Garland, M.: Visual exploration of complex time-varying graphs. IEEE Trans. Vis. Comput. Graph. **12**(5), 805–812 (2006)
19. Lee, Y.Y., Lin, C.C., Yen, H.C.: Mental map preserving graph drawing using simulated annealing. In: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation, APVis 2006, Vol. 60, pp. 179–188 (2006)
20. Mcfarland, D.: Student resistance: how the formal and informal organization of classrooms facilitate everyday forms of student defiance. Am. J. Sociol. **107**(3), 612–678 (2001)
21. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Vi. Lang. Comput. **6**(2), 183–210 (1995)
22. North, S.C.: Incremental layout in dynadag. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 409–418. Springer, Heidelberg (1996)
23. Purchase, H.C., Hoggan, E., Görg, C.: How important is the "Mental Map"? – an empirical investigation of a dynamic graph layout algorithm. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 184–195. Springer, Heidelberg (2007)

24. Purchase, H.C., Samra, A.: Extremes are better: investigating mental map preservation in dynamic graphs. In: Stapleton, G., Howse, J., Lee, J. (eds.) Diagrams 2008. LNCS (LNAI), vol. 5223, pp. 60–73. Springer, Heidelberg (2008)
25. Tufte, E.R.: Envisioning Information. Graphic Press, Cheshire (1990)
26. Yao, Y.: Collection and streaming of graph datasets. http://www.eecs.wsu.edu/yyao/StreamingGraphs.html