Clustering, Visualizing, and Navigating for Large Dynamic Graphs

Arnaud Sallaberry[1], Chris Muelder[1], and Kwan-Liu Ma[1]

University of California at Davis, U.S.A.
asallaberry@ucdavis.edu, {muelder,ma}@cs.ucdavis.edu

Abstract. In this paper, we present a new approach to exploring dynamic graphs. We have developed a new clustering algorithm for dynamic graphs which finds an ideal clustering for each time-step and links the clusters together. The resulting time-varying clusters are then used to define two visual representations. The first view is an overview that shows how clusters evolve over time and provides an interface to find and select interesting time-steps. The second view consists of a node link diagram of a selected time-step which uses the clustering to efficiently define the layout. By using the time-dependant clustering, we ensure the stability of our visualization and preserve user mental map by minimizing node motion, while simultaneously producing an ideal layout for each time step. Also, as the clustering is computed ahead of time, the second view updates in linear time which allows for interactivity even for graphs with upwards of tens of thousands of nodes.

1 Introduction

In recent years, the domain of network visualization has yielded many techniques for exploring large graphs. Most of these deal with static networks and are based on graph drawing algorithms (see for example [14] or [21]), clustering techniques (see [30] for an introduction), or exploratory methods (see for example [15], [1] or [2]). In contrast, many fewer works have been devoted to the exploration of dynamic graphs. A dynamic graph is an evolving graph where vertices and edges are added and removed over time. Examples of such graphs include social networks, dependency graphs in software engineering, website hyperlinks, router networks, collaboration networks, *etc.*

When creating a node-link diagram for a dynamic graph, not only does the layout need to consider graph topology, but also the stability between time-steps. This generally forces a trade-off between layout quality and stability, as a perfectly stable layout would sacrifice layout quality, and naively calculating ideal layouts would not offer stability. While there are a number of existing methods for creating these layouts, they generally do not scale well to large scale graphs.

In this paper, we describe a visualization approach for dynamic graphs that provides an overview of the entire dynamic graph over time, yields high quality layouts for every time-step, minimizes node motion between time steps to provide

W. Didimo and M. Patrignani (Eds.): GD 2012, LNCS 7704, pp. 487–498, 2013.

stability and preserve the user's mental map, and which is also efficient enough to allow for interactive exploration even under random access patterns. Our approach consists of first clustering each time-step independently to guarantee good locality for that time-step, associating the clusters between time-steps, and then arranging the clusters and nodes such that nodes that are consistant are stationary and transitional node motion is minimized. This produces a temporal arrangement which we directly visualize as a timeline, and which we use to define layouts for a node link diagram for each time-step which both meets general layout criteria (namely cluster co-location and short average edge length) and where node motion is minimized between time-steps.

The main contributions of this paper are the novel algorithms for clustering and ordering dynamic graphs, the overview visualization, the stable and efficient layout approach, and the scalability and interactivity of the overall system. The clustering and ordering algorithms are useful for the discovery of trends among time-varying graphs, and we think it will be a strong basis for future works in this area. Our approach is the first method that we are aware of that creates an overview to show the evolution of clusters, which is an important contribution in the process of making hypothesis about trends among the evolution of communities in a graph. Also, to the best of our knowledge, it is the first approach that allows for interactive visualization and exploration of dynamic graphs with tens of thousands of nodes and edges.

2 Related Works

A common method for visualizing dynamic graphs is to animate the transitions between time-steps [24,9,10,13,5,11]. This approach yields dynamic visualization with nodes appearing, disappearing and moving to produce readable layout for each time-step. Alternatively, multiple time-steps can be staticly placed next to each other using "Small Multiples" [32]. This eases the comparison of distant time-steps but the area devoted for each time-step is small and this reduces the readability of each graph. An empirical study to compare the advantages and drawbacks of these approaches ("Animation" vs. "Small Multiples") has been performed by Archambault *et al.* [3]. A major issue for both methods is to ensure the stability of the layout [20,11,7,18]. A stable layout helps preserve the user's mental map as there is less movement between time-steps, but sacrifices quality in terms of readability for later time-steps as their layout depends on previous time-steps. Many experiments has been proposed to examine the effect of preserving the mental map in dynamic graphs visualization [27,29,28]. The results of [28] were quite surprising because the most effective visualizations were the extreme ones, *i.e.* the ones with very low or high mental map preservation: visualizations with medium preservation performed less well. The approach described in this paper aims to achieve high mental map preservation.

An interesting visualization dealing with dynamic large directed graphs has been proposed by Burch *et al.* [8]. Vertices are ordered and positioned on several vertical parallel lines, and directed edges connect these vertices from left to right.

Each time-step's graph is thus displayed between two consecutive vertical axes. Hu *et al.* [18] proposed a method based on a geographical metaphor to visualize clustered dynamic graphs. However, their approach requires one global clustering over time, while ours allows nodes to be transferred in order to create better local clusterings and to capture the evolutions of the communities.

3 Clustering

A dynamic graph can be defined formally as an agglomerate graph $G = (V, E)$ and an ordered sequence of subgraphs $S = \{G_1 = (V_1, E_1), G_2 = (V_2, E_2), ..., G_k = (V_k, E_k)\}$ where each G_t is the subgraph of G at time t. $V, V_1, V_2, ..., V_k$ are finite and non-disjointed sets of nodes, $E, E_1, E_2, ..., E_k$ are finite and non-disjointed sets of edges such that $V = V_1 \cup V_2 \cup ... \cup V_k$ and $E = E_1 \cup E_2 \cup ... \cup E_k$. What we need is to create a time-varying clustering, *i.e.* a set of clusters evolving over time. The clustering method we describe here is a two step algorithm. The first step consists of partitioning the nodes for each time step independently. Then, we associate these clusters through time to derive time-varying clusters.

3.1 Time-Step Clusterings

Finding a partition of the nodes of a static graph according to its structure is a well studied problem. Schaeffer has published a good overview of graph clustering methods[30]. For our approach, we need to cluster a dynamic graph, which is a less studied problem. We do this by first finding a partition for each time step, *i.e.* a set of clusterings $C = \{C_1, C_2, ..., C_k\}$ where $C_t = \{c_1^t, c_2^t, ..., c_{lt}^t\}$ is a partition of the nodes V_t of G_t. In this paper, we call each C_t a "time-step clustering" where c_i^t is the "time-step cluster" i at time t, and $c_i^t \subseteq V_t$ for each $i \in [\![1, lt]\!]$, $V_t = c_1^t \cup c_2^t \cup ... c_l^t$ and $c_i^t \cap c_j^t = \emptyset$ for each pair $(i, j) \in [\![1, lt]\!]^2$.

Our algorithm is based on the so-called modularity function [22]. It represents the sum of the number of edges linking nodes of the same clusters minus the expected such sum if edges were distributed at random. For a graph $G_t = (V_t, E_t)$ and a partition C_t of its nodes, the modularity $Q(C_t)$ is defined by:

$$Q(C_t) = \frac{1}{2|E_t|} \sum_{u,v \in V_t} \left[A_{uv} - \frac{k_u k_v}{2|E_t|} \right] \delta(c^t(u), c^t(v))$$

where $|E_t|$ is the number of edges, A_{uv} is 1 if there is an edge between u and v and 0 otherwise, $k_u = \sum_v A_{uv}$ is the number of edges attached to u, $c^t(u)$ is the time-step cluster of C_t containing u, $\delta(c^t(u), c^t(v))$ is 1 if $c^t(u) = c^t(v)$ and 0 otherwise.

A partition that maximizes this function helps to discover clusters of densely connected communities. Moreover, as shown by Noack [23], optimizing the modularity is the same as optimizing an energy function in graph layout. This equivalence implies that our layout based on such a clustering algorithm yields a good representation of the graph.

The problem of finding a partition that maximizes the modularity is hard, and the corresponding decision problem is NP-complete [6]. We use the heuristic proposed by Blondel *et al.* [4], which works well in terms of both the quality of the results and the computation time. Initially, each node belongs to its own cluster. Then pairs of clusters are recursively merged such that the modularity of the partitioning increases. If two possible merges involve the same cluster, the merge that improves the modularity the most is performed.

3.2 Time-Varying Clustering

We define a time-varying clustering of a dynamic graph G as a set of time-varying clusters $VC = \{VC_1, VC_2, ..., VC_l\}$. Each of these time-varying clusters is an ordered sequence $VC_i = \{vc_i^1, vc_i^2, ..., vc_i^k\}$ where k is the number of time steps and each vc_i^t is a subset of the vertices V_t at time t. That is, each time-varying cluster VC_i is a cluster whose membership can evolve over time, where vc_i^t represents the set of nodes in the cluster i at time t. As the number of clusters can change between timesteps, not every cluster exists at every timestep, so the total number of time-varying clusters l can be larger than the number of time step clusters at any time step.

Our overall approach is to compare the time-step clusters pairwise between neighboring time-steps and putting the most similar time-step clusters into the same time-varying cluster. We start from an empty set VC of time-varying clusters and we create a time-varying cluster VC_i for each time-step cluster c_i^1 of the first time-step clustering C_1. The set of nodes of these time-varying clusters VC_i at time 1 are initialized with the time-step clusters c_i^1: $vc_i^1 \leftarrow c_i^1$.

Then, starting with this partition of the graph at time 1, for each vc_i^2 we search for the time-step cluster of C_2 that is the most similar to vc_i^1. Let c_a^2 be such a cluster, then $vc_i^2 \leftarrow c_a^2$. If no similar cluster can be found in C_2, then vc_i^2 is an empty set (i.e. the time-varying cluster has disappeared at time-step 2). If there is a time-step cluster c_a^2 in C_2 that cannot be associated with a vc_i^2, then a new time-varying cluster VC_b is created with $vc_b^1 \leftarrow \emptyset$ and $vc_b^2 \leftarrow c_a^2$. We iterate this process for each time-step.

The crux of this algorithm is how to decide which time-step cluster of C_t is the most similar to a cluster of C_{t-1}. The solution we use is based on a similarity function between time-step clusters of C_{t-1} and clusters of C_t. Results of this function can be stored in a matrix M such that M_{ij} denotes the similarity between c_i^{t-1} and c_j^t. Starting from the highest value of this matrix and the corresponding clusters c_i^{t-1} and c_j^t, we assign c_j^t to the time t of the dynamic cluster VC_i that contains the cluster c_i^{t-1} at the time $t-1$. Then we do the same for the second highest value of the matrix and so on. Any values corresponding to time-step clusters that have already been assigned to a dynamic cluster are skipped. This process ends when there are no more time-step clusters or when the highest similarity value is under a given *threshold*. Finally, any remaining time-step clusters become new dynamic clusters.

In our implementation, we use the Jaccard index to compute the similarities. For two clusters c_i^{t-1} and c_j^t, this is defined by the equation $|c_i^{t-1} \cap c_j^t| / |c_i^{t-1} \cup c_j^t|$. There are two main advantages in using this metric. First it takes into account the number of shared nodes as well as the total number of nodes, which guarantees homogeneity between consecutive steps of a time-varying cluster. Secondly it returns a value normalized between 0 and 1 which is helpful for empirically defining a *threshold*.

4 Ordering

Our visualization is based not just on a clustering, but on an ordering of the time-varying clusters (in the next sections, we use the word "cluster" instead of "time-varying cluster" to simplify the notation). Then, nodes are also ordered within each cluster. A node that moves from a cluster VC_a to a cluster VC_b is involved in the node ordering of both VC_a and VC_b, *e.g.* it can be the 6th node of VC_a and the 3rd node of VC_b.

4.1 Ordering Clusters

The stability of the layout is one of the main goals of our method: we want to easily see the evolution of the clusters and also be able to follow nodes that move between clusters. As the layout depends on the ordering, clusters need to be ordered in such a way that two clusters exchanging many nodes are close to each other.

We do this by first creating a weighted quotient graph $QG = (V_{QG}, E_{QG}, \omega)$ defined by the relationships between the time-varying clusters VC of G. Each node of V_{QG} represents a cluster of VC, *i.e.* $V_{QG} \leftarrow VC$. There is an edge in E_{QG} between VC_i and VC_j if and only if there is at least one node in the sets of VC_i that is also in a set of VC_j. The weight function ω is a function $\omega : E_{QG} \to \mathbb{N}$ defined for each edge $e = (VC_i, VC_j)$ as the number of transferred of nodes between sets of VC_i and sets of VC_j.

Next we need to find an ordering of these clusters, *i.e.* a permutation $\phi : V_{QG} \to \{1, 2, ..., |V_{QG}|\}$ that minimizes the function:

$$LA_\phi(QG) = \sum_{\substack{uv \in E_{QG} \\ u,v \in V_{QG}}} \omega(uv) \cdot |\phi(u) - \phi(v)|$$

This function is called the *Linear arrangement function* (LA) and finding an ordering that minimizes it is known as the *Minimum Linear Arrangement Problem* (MinLA) [26]. MinLA is NP-hard and the corresponding decision problem is NP-complete [12]. Many heuristics have been proposed to find a satisfying solution. A list of these methods and an experiment has been proposed by Petit [26]. More recently, Koren and Harel have proposed a new heuristic [19] that is a good compromise between computation time and quality of the results.

4.2 Ordering Nodes

The second ordering step consists in finding a permutation of the nodes within each cluster VC_i of VC. Since we want to maximize stability, we calculate this permutation over all time, so that nodes will not move within a cluster, even if this leaves gaps at some time-steps. As with the clusters, this is another MinLA problem. Let vc_i be the set of nodes of VC_i: $vc_i = \bigcup_{1 \leq t \leq k} vc_i^t$. Then the permutation is defined as $\varphi_i : vc_i \rightarrow \{1, 2, ..., |vc_i|\}$. This ordering needs to take into account the ordering of the clusters computed previously: for example, if a node v moves only once from a cluster VC_a to a cluster VC_b and if $\phi(VC_a) < \phi(VC_b)$, then v should lie at the upper extremity of VC_a (high $\varphi_a(v)$) and at the lower extremity of VC_b (low $\varphi_b(v)$). To find the permutation φ_i, we first compute for each node v of vc_i the median of the clusters VC_i v belongs to:

$$median_i(v) = \frac{\sum_{VC_j; v \in vc_j} \phi(VC_j)}{|\{VC_j; v \in vc_j\}|}$$

Then, the permutation φ_i is the ordering obtained by sorting the nodes of vc_i according to their median value: $median_i(v) < median_i(u) \Leftrightarrow \varphi_i(v) < \varphi_i(u)$.

5 Visualization

The visualization methods we employ focus on representing the evolving clusters in dynamic graphs. inspired by [25,31] that provides an overview of the entire dynamic graph, and a more traditional node-link view for individual time-steps. Both of these views are derived from the clustering and ordering methods described earlier. Moreover, since the clustering and ordering are computed as a preprocessing step, the computation times of the visualizations are linear, which makes it possible to obtain real-time, interactive navigation of the dynamic graph.

5.1 Time-Line View

The time-line view depicts an overview of the nodes' arrangement into clusters and of the nodes motion between clusters. Each node is represented as a line where the x-position is time and the y-positions corresponds the cluster the nodes belong to at each given time and its position within the cluster.

Figure 1(a) shows an example. For reference purposes, in this diagram the time-steps are represented with vertical grey lines (here from from 1 to 5). There are 8 horizontal lines (including black, blue and green ones), which correspond to 8 nodes. There are four clusters on the y-axis, and a horizontal line is in front of one of them when the corresponding node belongs to it. Clusters are positioned according to the ordering ϕ computed by the pre-processing algorithm, from bottom to top (*e.g.* the cluster labelled 4 is the cluster VC_a with a such that $4 = \phi(VC_a)$).

As an example of reading this plot, consider the blue line. The corresponding node v belongs to the cluster 4 at times 1 and 2, and it belongs to the cluster 3 at times 3 and 4, since the blue line moves from cluster 4 to cluster 3 at time 3. We also see that the blue node is no longer in the graph at the time-step 5 and that the green node appears in the graph at the time-step 2.

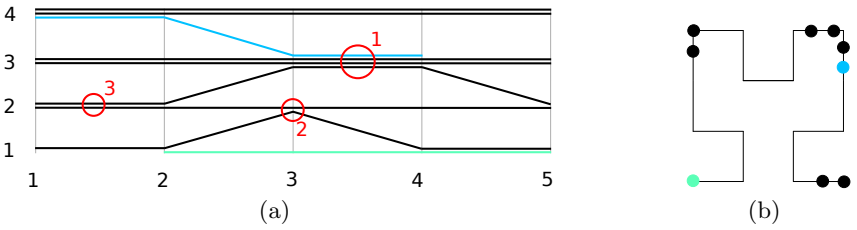


Fig. 1. (a) The time-line gives an overview of the clusters and of the nodes moving from clusters to clusters. Each horizontal or bent line is a node. Vertical grey lines represent time-steps, from 1 to 5. Y-axis represents clusters, *e.g.* the blue line near the cluster 4 at time-steps 1 and 2 stands for a node v that belongs to VC_a with a such that $4 = \phi(VC_a)$, at the time 1 and at the time 2 (it belongs to vc_a^1 and vc_a^2 but not to vc_a^3, vc_a^4 and vc_a^5). (b) Time-step view of the graph used in the example of Figure 1(a). It shows the graph at the 3rd time-step. We don't display the edges here. Nodes represented as disks are positioned along a Hilbert's curve, represented by the black bended line.

Lines in the clusters are positioned according to the orderings φ_i computed during the pre-processing step. In this way, a node v that moves from a cluster VC_a to a cluster VC_b with $\phi(VC_a) < \phi(VC_b)$ is likely to be positioned at the upper extremity of VC_a (high $\varphi_a(v)$) and at the lower extremity of VC_b (low $\varphi_b(v)$). This technique reduces edge crossings and improves the readability of the view.

Clusters are separated by a constant gap to clarify their distinctions. The height of a cluster VC_i corresponds to the size the set vc_i of all the nodes that belong to it at least for one time-step. As an example, $|vc_b| = 4$ (see the red circle 1) and $|vc_c| = 3$ with c such that $2 = \phi(VC_c)$ (see the red circles 2 and 3). Thanks to this, a node has always the same position in the same cluster, so there will be no bends when a node remains in the same cluster and the area devoted to a cluster remains the same.

5.2 Time-Step View

The second view is a node-link diagram that shows the graph at any selected time-step. The layout is based on the technique of Muelder and Ma [21], which maps a 1-D ordering of nodes to a space filling curve to define the layout.

Since we have already computed a stable ordering of nodes, it is sensible to map this same ordering onto the space filling curve. In the timeline, the height

of each line at any time step corresponds to the pre-computed ordering. So, we can reuse these y-positions as a 1-D layout for that time-step, then map the nodes directly to a space-filling curve by placing the nodes at the corresponding distance along the curve. This is done by normalizing both the 1-D layout and the length of the curve, then calculating the position of each node by recursively mapping it to the curve in constant time, as in the original paper [21]. Figure 1(b) shows an example of this node positioning on the same example as the one presented in Figure 1(a) for time-step 3. In this diagram, we use a Hilbert curve, but we also use Peano curve, a Gosper curve, and an H-curve, and the user can switch between these curves as desired (see [16] for a summary of well-known space-filling curves).

One interesting property of a space-filling curve is known as the *Worst-Case Locality* [16]. This property guarantees that the euclidean distances between nodes in the layout are bounded by the distances of the same nodes in the one-dimensional layout. So, the proximities of elements (nodes/clusters) depend directly on the ordering. As the ordering is based primarily on the connectivity of the networks, this guarantees layout quality metrics, such as tightly connected groups of nodes being placed close together with a good aspect ratio, and short average edge lengths.

Since a node has always the same position in the time-line when it is in the same cluster and the area devoted to a cluster remains the same, its placement in the layout will also be constant. This ensures the stability of the layout. Even the distance that moving nodes is minimized, as the ordering is computed such that clusters that exchange many nodes will be placed closer together.

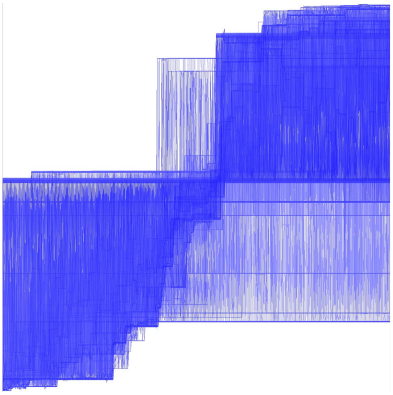
As the layout itself runs in linear time (see the section 7 for more detailed explanation), the visualization can be updated interactively by the user and we can even easily play the sequence of graphs and animate the transitions with graphs of tens of thousands nodes/edges (see section 5.3 for more details).

As we use a clustering hierarchy, we can also employ the hierarchical edge bundling technique [17] which improves the readability of the graph. Control points of the spline linking a node v and a node u are defined by the path through the clustering hierarchy, and placed according to the clusters' centroids.

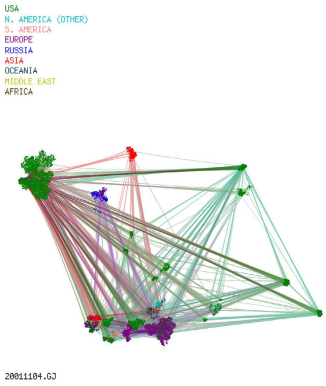
Figure 2 shows a real-world example of both views.

5.3 Interaction and Navigation

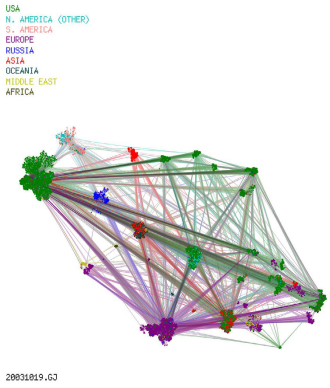
One of the most useful features of our approach is that any time-step can be laid out quickly and directly, without needing to iterate over the other time-steps. The benefit of this is that it enables random access. That is, users can find interesting time-steps in the time-line and skip between them directly. We enable this form of interaction by letting the user simply click in the time-line on the time-step that they want to load. We also include the more traditional approach of simply animating over the entire dynamic graph. In either case, the positions of nodes that move are interpolated between time-steps so that the user can follow their motion. Within the node-link diagram itself, we can also allow for traditional graph interaction, such as selection, or focus+context zooming.



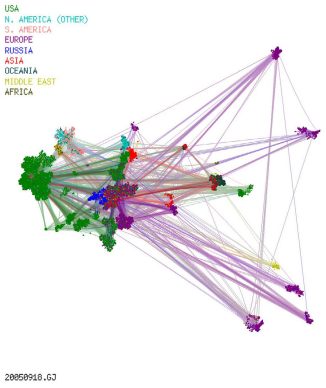
(a) Timeline



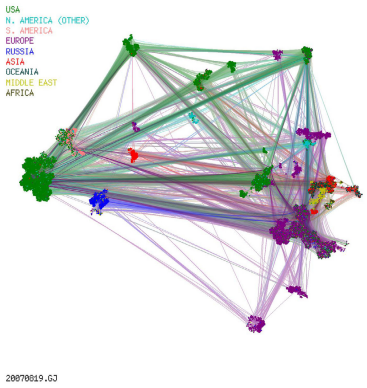
(b) 2001-11-04



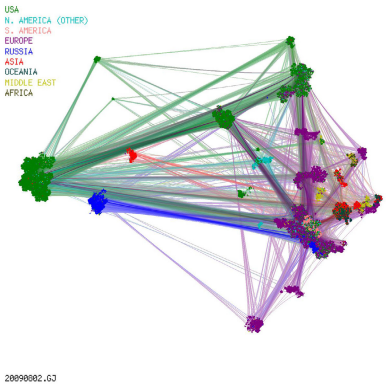
(c) 2003-10-19



(d) 2005-09-18



(e) 2007-08-19



(f) 2009-08-02

Fig. 2. The autonomous systems of the Internet

6 Case Study

The Internet dataset, shown in Figure 2, is a very large and complex dynamic network. It is composed of 41,928 nodes and 218,080 edges over 421 time-steps. But even at this scale, it still renders fast enough to allow for smooth animation or interactivity. The timeline is dense and chaotic (Figure 2(a)). However, this timeline does reveal some interesting patterns. Firstly, there are several dense horizontal sections that never change much, which would be core autonomous systems. Secondly, there is a gradual shift over time, from the bottom of the plot to the top. This could be due to gradual improvements to the underlying connectivity (e.g. when a newer, faster fiber-optic cable is developed, a new sub-network could form utilizing it). These patterns are also visible in the node-link diagrams. In every time-step, the large green USA cluster at the left of the plot (most likely the west coast) is fairly constant, other than the density increasing. Russia (in blue) and the asian cluster just left of the center of the plot are also quite stable (although Russia seems to split near the end). The rest of America is rather chaotic until near the end of the time range where they coalesce in the upper right region of the plot. Europe (in purple) has a very interesting pattern: there is a large cluster that is fairly constant for the first part of the time range, and a large cluster that is constant near the end of the time range, but in the middle it jumps around the plot a lot. Also, there are several smaller European clusters distributed throughout the plot. South America also has an interesting pattern, as it is right by the large USA cluster except for the very beginning and end of the run. One pattern that is not apparant in the still images, but is very interesting to see in the animation, is a trend where a single node or small group of nodes starts a new cluster, then several nodes leave their clusters to join the new cluster, and then the cluster immediately dissolves, all within a few time-steps. We posit that this could be due to a misconfiguration where a router was temporarily issuing bad BGP routes, and then was subsequently fixed.

7 Discussion

One of the strongest points of our approach is its scalability and interactivity. Nearly all the computation is done in the pre-processing step, so both visual representations can be generated extremely quickly. The time-line takes $\Theta(k*n)$ where k is the number of time-steps and n is the number of nodes. The graph layout is even faster, taking only $\Theta(n)$ time to compute. It actually takes more computation to render the network then to lay it out, as the rendering is $\Theta(n+m)$ where m is the number of edges.

The downside to this is that the pre-processing step can take a long time. Clustering each time step is relatively fast (see the original paper [4] for timing details). Correlating clusters between time-steps and ordering the clusters are both fast, since there are much fewer clusters than nodes. The limiting factor is the ordering of the nodes within each cluster, which can take hours on larger datasets.

8 Conclusion

We have presented a new, highly scalable approach for exploration of large dynamic graphs. Since the layout and rendering is performed in linear time, the user can interactively navigate dynamic graphs of upwards of tens of thousands of nodes and hundreds of thousands of edges. The layout method performs exceptionally well both in quality since each time-step is clustered locally, as well as in stability since the nodes' placements are fixed. As a future work, we intend to further explore faster and better algorithms for deriving time-varying clusters.

Acknowledgements. This research is sponsored in part by the National Science Foundation through grants CCF 1025269 and CCF 0811422.

References

1. Abello, J., van Ham, F., Krishnan, N.: ASK-GraphView: A large scale graph visualization system. IEEE TVCG 12(5), 669–676 (2006)
2. Archambault, D., Munzner, T., Auber, D.: GrouseFlocks: Steerable exploration of graph hierarchy space. IEEE TVCG 14(4), 900–913 (2008)
3. Archambault, D., Purchase, H.C., Pinaud, B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE TVCG 17(4), 539–552 (2011)
4. Blondel, V., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment (2008)
5. 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)
6. Brandes, U., Delling, D., Gaertler, M., Goerke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: Maximizing modularity is hard (2006), `http://arxiv.org/abs/physics/0608255`
7. Brandes, U., Mader, M.: A Quantitative Comparison of Stress-Minimization Approaches for Offline Dynamic Graph Drawing. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 99–110. Springer, Heidelberg (2011)
8. Burch, M., Vehlow, C., Beck, F., Diehl, S., Weiskopf, D.: Parallel edge splatting for scalable dynamic graph visualization. IEEE TVCG 17(12), 2344–2353 (2011)
9. Diehl, S., Görg, C.: Graphs, They are Changing: Dynamic Graph Drawing for a Sequence of Graphs. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 23–30. Springer, Heidelberg (2002)
10. 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)
11. Frishman, Y., Tal, A.: Online dynamic graph drawing. IEEE TVCG 14(4), 727–740 (2008)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
13. Görg, C., Birke, P., Pohl, M., Diehl, S.: Dynamic Graph Drawing of Sequences of Orthogonal and Hierarchical Graphs. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 228–238. Springer, Heidelberg (2004)

14. 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)
15. van Ham, F., van Wijk, J.J.: Interactive visualization of small world graphs. In: Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2004), pp. 199–206 (2004)
16. Haverkort, H.J., van Walderveen, F.: Locality and bounding-box quality of two-dimensional space-filling curves. Computational Geometry, Theory and Applications 43(2), 131–147 (2010)
17. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE TVCG 12(5), 741–748 (2006)
18. Hu, Y., Kobourov, S.G., Veeramoni, S.: Embedding, clustering and coloring for dynamic maps. In: Proceedings of the 5th IEEE Pacific Visualization Symposium (PacificVis 2012), pp. 33–40 (2012)
19. Koren, Y., Harel, D.: A Multi-scale Algorithm for the Linear Arrangement Problem. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 296–309. Springer, Heidelberg (2002)
20. Kumar, G., Garland, M.: Visual exploration of complex time-varying graphs. IEEE TVCG 12(5), 805–812 (2006)
21. Muelder, C., Ma, K.L.: Rapid graph layout using space filling curves. IEEE TVCG 14(6), 1301–1308 (2008)
22. Newman, M.E.J., Girvan, M.: Graph clustering. Physical Review E 69(026113) (2004)
23. Noack, A.: Modularity clustering is force-directed layout. CoRR abs/0807.4052 (2008)
24. North, S.C.: Incremental Layout in DynaDAG. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 409–418. Springer, Heidelberg (1996)
25. Ogawa, M., Ma, K.L.: Software evolution storylines. In: Proceedings of the ACM 2010 Symposium on Software Visualization (SoftVis 2010), pp. 35–42 (2010)
26. Petit, J.: Experiments on the minimum linear arrangement problem. Tech. Rep. LSI-01-7-R, Universitat Politecnica de Catalunya, Departament de Llenguatges i Sistemes Informatics (2001)
27. 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)
28. 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)
29. Saffrey, P., Purchase, H.: The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In: Proceedings of the 9th Australasian User Interface Conference (AUIC 2008), pp. 85–93 (2008)
30. Schaeffer, S.E.: Graph clustering. Computer Science Review 1(1), 27–64 (2007)
31. Tanahashi, Y., Ma, K.L.: Design considerations for optimizing storyline visualizations. To appear in IEEE TVCG (2012)
32. Tufte, E.R.: Envisionning Information. Graphics Press (1990)