Improved Cluster Tracking for Visualization of Large Dynamic Graphs

Chris Muelder, Arnaud Sallaberry, Kwan-Liu Ma

Improved Cluster Tracking for Visualization of Large Dynamic Graphs

Chris Muelder*, Arnaud Sallaberry**, Kwan-Liu Ma*

* VIDi group - University of California, Davis
One Shields Avenue
Davis, CA 95616-8562, USA
muelder@cs.ucdavis.edu, ma@cs.ucdavis.edu
http://vis.cs.ucdavis.edu/~muelder/, http://www.cs.ucdavis.edu/~ma/

**LIRMM - Université Montpellier 3
UMR 5506 - CC 477
161, rue Ada
34095 Montpellier Cedex 5, France
arnaud.sallaberry@lirmm.fr
http://www2.lirmm.fr/~sallaberry/

Abstract. Analysis and visualization of dynamic graphs is a challenging problem. Clustering can be applied to dynamic graphs in order to generate interactive visualizations with both high stability and good layout quality. However, the existing implementation is naïve and unoptimized. Here we present new algorithms to improve both the temporal clustering results and the efficiency of the cluster tracking calculation, and evaluate the results and performance.

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 Hachul and Jünger (2006) or Muelder and Ma (2008)), clustering techniques (see Schaeffer (2007) for an introduction), or exploratory methods (see for example van Ham and van Wijk (2004), Abello et al. (2006) or Archambault et al. (2008)). In contrast, 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 have not been shown to scale well to large dynamic graphs.

A dynamic clustering based approach for visualization of dynamic graphs can provide an overview of the entire dynamic graph over time, yield high quality layouts for every time-step, minimize node motion between time steps to provide stability and preserve the user's mental map, and allow for interactive exploration even under random access patterns. One existing approach consists of first clustering each time-step independently to guarantee good locality for every time-step, then tracking the clusters between time-steps, and finally arranging the clusters and nodes such that nodes that are consistant are stationary and transitional node motion is minimized Sallaberry et al. (2013). This produces a temporal arrangement which we directly visualize as a timeline, and which is used 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.

However, there are some limitations to this approach. First, by only considering pairwise timesteps, it is impossible to track clusters that disappear for some amount of time before reforming, which results in the formation of extraneous dynamic clusters. Second, the association between timesteps was previously cacluated very naïvely, with pairwise comparison and Jaccard calculation for each possible association.

Here, we describe improvements to this approach that resolve both of these issues. The first issue is addressed by an algorithm for preserving unused dynamic clusters from old time steps and allowing them to be reincorporated into new time steps. And the second issue is addressed through a heavy optimization of the association algorithm that not only reduces the number of clusters that have to be compared but which also enables direct calculation of the Jaccard index. The result of the combination of these improvements is a more compact, robust, dynamic clustering that can be computed much more efficiently.

2 Related Works

A common method for visualizing dynamic graphs is to animate the transitions between time-steps (North, 1996; Diehl and Görg, 2002; Erten et al., 2004; Görg et al., 2004; Boitmanis et al., 2008; Frishman and Tal, 2008). 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" Tufte (1990). 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. (2011). A major issue for both methods is to ensure the stability of the layout (Kumar and Garland, 2006; Frishman and Tal, 2008; Brandes and Mader, 1012; Hu et al., 2012). 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 (Purchase et al., 2007; Saffrey and Purchase, 2008; Purchase and Samra, 2008). The results of Purchase and Samra (2008) 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. (2011). 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. (2012) 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.

Finding a partition of the nodes of a static graph according to its structure is a well studied problem; Schaeffer (2007) has published a good overview of graph clustering methods. But clustering a dynamic graph is a less studied problem.

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

To calculate a dynamic clustering, we first find 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 of Newman and Girvan (2004). 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 (2008), 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 (Brandes et al., 2006). We use the heuristic proposed by Blondel et al. (2008), 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

3.2.1 Overview

The previous approach (Sallaberry et al., 2013) was to compare each time-step cluster in the current time-step pairwise with the time-step clusters of the previous time-step according to the Jaccard index, and then to itratively and greedily associate the time-step clusters that most closely match into the same time-varying cluster, halting when all clusters are assigned or the similarity falls below a user-defined threshold. If there are any remaining new clusters that do not have a match, they are considered new clusters, and so they start new time-varying clusters. And any remaining time-varying clusters present in the previous time-step that were not assigned a cluster in the current time-step were discarded, as there was no match.

Our approach here operates fairly similarly, with two key differences. The first improvement is that rather than discarding time-varying clusters after they fail to find a match, we retain them in the matching algorithm so that they can return if that portion of the network returns to a similar enough state, even if there are many intervening time-steps. This reduces the overall number of time-varying clusters, and can aid in identifying certain patterns (such as periodicity, split/merges, clustering instability, etc...). The second improvement is to reduce the number of pairwise clusters to evaluate, by only comparing clusters that share at least one vertex. This optimization method has the added bonus that it can be used to calculate the Jaccard index very efficiently.

3.2.2 Details

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 time-step clusters can change between timesteps, not every time-varying cluster is populated at every timestep, and the total number of time-varying clusters l can be larger than the number of time-step clusters at any time step.

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, for each subsequent timestep t, we want to compute similarities between each time-step cluster $c_i^t \in C_t$ and potential time-varying clusters VC_i. 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 $threshold$.

In the original algorithm, the association step was performed by comparing each time-step cluster $c_i^t \in C_t$ to every time-step cluster $c_j^{t-1} \in C_{t-1}$ to create a similarity matrix, which costs $O(|C_t| * |C_{t-1}|)$ times the cost of the similarity calculation. Associations are then performed greedily, starting with the largest matrix value, and stopping when either one set of clusters is exhausted or the remaining matrix values are less than the threshold. Then, any remaining clusters in C_t were assigned new time-varying clusters in VC. And any remaining clusters in C_{t-1} were discarded, and their corresponding time-varying cluster in VC was terminated for the remainder of the execution.

However, we found that this process led to the creation of many new time-varying clusters, because sometimes the network might revert to a clustering previously encountered where the corresponding time-varying cluster was already terminated, so the method would create a new time-varying cluster. To resolve this, we preserve the most recent time-step cluster c_k^u for each time-varing clusters in VC that would have been terminated, and compare against those as well, where $0 < u < t - 1$. If an older c_k^u is more similar than any time-step cluster $c_j^{t-1} \in C_{t-1}$, then the system will select and revive the time-varying cluster of c_k^u instead of that of one of active time-varying clusters VC_i. This adds additional complexity to the algorithm, but produces more robust and compact results, as time-varying clusters are allowed to reform instead being terminated.

Another issue is that many of the time-step clusters are disjoint sets of nodes, and thus we do not need to compute their similarity. In our improved version, rather than computing the entire matrix, we consider only the clusters that share at least one node. We do this by computing a list of candidate clusters CC_i^t for each time-step cluster c_i^t, which we define as $CC_i^t = \{(vc_a^*, |c_i^t \cap vc_a^*|), (vc_b^*, |c_i^t \cap vc_b^*|), ...\}$, where each vc_j^* is the most recent timestep of VC_j. This can be computed relatively efficiently by iterating over each node $n \in c_i^t$: for each n we take any vc_j^* that contains n, then either add vc_j^* to CC_i^t with a paired value of 1, or if it is already in the list, we simply increment its paired value. To make this process even more efficient, we use a lookup table to map each node n to its existing vc_j^* clusters. In the original process, this would be trivial to do, as each node n could only exist in one previous cluster vc_j^*. However, since we are now preserving terminated clusters, it is possible for n to have multiple previous clusters. So we build a hash to map each node n to its prior clusters by iterating over each vc_j^* and inserting a pointer to vc_j^* for each node $m \in vc_j^*$. This elimates the need for any searching. While this process appears to add some computational overhead as it iterates over every node instead of working with the clusters, this is entirely offset in the calculation of the Jaccard index, as we have already computed the size of each intersection $|c_i^t \cap vc_j^*|$. From this, we can calculate the Jaccard index in $O(1)$ time as $J(c_i^t, vc_j^*) = \frac{|c_i^t \cap vc_j^*|}{|c_i^t| + |vc_j^*| - |c_i^t \cap vc_j^*|}$. So while this optimization does add additional memory overhead, the computation is much more efficient.

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 (Petit, 2001). MinLA is NP-hard and the corresponding decision problem is NP-complete (Garey and Johnson, 1979). Many heuristics have been proposed to find a satisfying solution. A list of these methods and an experiment has been proposed by Petit (2001). More recently, Koren and Harel (2002) have proposed a new heuristic 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 $\varphi_i : vc_i \to \{1, 2, ..., |vc_i|\}$. 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. We employ two views: a time-line inspired by Ogawa and Ma (2010); Tanahashi and Ma (2012) 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.

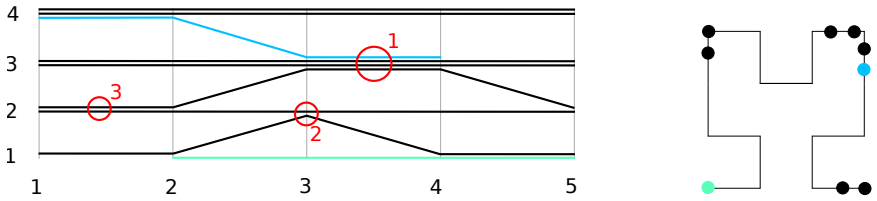
5.1 Time-line view

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 (from t=1 to 5) positioned along the x-axis that represents time. There are 8 plotted 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. Also, 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.

Lines in the clusters are positioned according to the orderings φ_i computed during the preprocessing 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.



(a) (b)

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.

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 (2008), 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 (Muelder and Ma, 2008). 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 Haverkort and van Walderveen (2010) for a summary of well-known space-filling curves).

One interesting property of a space-filling curve is known as the *Worst-Case Locality* (Haverkort and van Walderveen, 2010). 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 nodes move is minimized, as the ordering is such that clusters that exchange many nodes are placed closer together.

As the layout itself runs in linear time, 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 our previous paper (Sallaberry et al., 2013) for more details).

As we use a clustering hierarchy, we can also employ the hierarchical edge bundling technique of Holten (2006) 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.

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.

6 Discussion

We have described two algorithmic improvements over the previous approach. First, our new approach is tuned to reduce the total number of persistant time-varying clusters, which will improve the space utilization of the resulting visualization. Second, we have substantially optimized the cluster association step, which greatly reduces the time it takes to process the data. Here, we present some case studies and quantitatively evaluate both improvements.

To test our improvements, we ran both improvements as well as the original approach on several datasets. First, we ran them on a social network dataset collected from the Rimzu social networking site, as used by Frishman and Tal (2008). While quite small and straightforward, this dataset makes a good baseline for comparison against existing works. Next, we evaluated the MIT Reality dataset (Eagle and Pentland, 2005). In this dataset, there is a small, strong core of about 80 people, but hundreds of periferal nodes that do persist through the data and which generally only have one connection. Due to their transitivity, they do not contriubte much to the overall structure of the network, so we evaluate both the entire network and just the core network without the external nodes. Finally, we evaluate the improvement on a dataset of the autonomous systems of the internet, derived from data collected by the Oregon Route Views project and as used in several existing works (Muelder, 2011; Sallaberry et al., 2013).

In each case, the results followed our expectation, as is shown in Table 1. The inclusion of additional comparisons against dead clusters succeeds in reducing the total number of dynamic time-varying clusters, but requires more comparisons, and hence takes more computation time. However, this is entirely offset by the optimization, which reduces the computation time by around a factor of 20 or more.

	Original		Improved		
	N Clusters	Time (ms)	N Clusters	Time (ms)	Opt. time (ms)
Social network	392	3050	391	3150	160
MIT reality	518	120	457	518	80
MIT reality (core)	136	30	78	50	<1
Internet (16 steps)	162	7220	86	9640	400

TAB. 1: Results of our improvements. We found that our methods reduce the number of time-varying clusters by up to half, and reduce computation time by a factor of about 20 or more.

7 Conclusion and Future Work

This paper presents an incremental improvement to a previous work. While the visual representation remains the same, the underlying clustering algorithm has been greatly improved and optimized. The evaluation follows accordingly, and demonstrates our improvements quantitatively.

While the results described in this paper are promising, they are still further ways they can be improved. Most importantly, the cluster and node ordering steps are still extremely slow and we are investigating ways to improve upon this. One such method is to assign more localized orderings, where we find the actual time range used by each dynamic cluster, and more compactly arrange the clusters accordingly, similar to the work of Tanahashi and Ma (2012). However, this algorithm is also very time consuming, and we are investigating heuristic improvements.

References

Abello, J., F. van Ham, and N. Krishnan (2006). ASK-GraphView: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics 12*(5), 669–676.

Archambault, D., T. Munzner, and D. Auber (2008). GrouseFlocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics 14*(4), 900–913.

Archambault, D., H. C. Purchase, and B. Pinaud (2011). Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics 17*(4), 539–552.

Blondel, V., J. Guillaume, R. Lambiotte, and E. Lefebvre (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*.

Boitmanis, K., U. Brandes, and C. Pich (2008). Visualizing internet evolution on the autonomous systems level. In *Proceedings of the International Symposium on Graph Drawing (GD'07)*, Volume 4875 of *LNCS*, pp. 365–376. Springer.

Brandes, U., D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner (2006). Maximizing modularity is hard. `arxiv.org/abs/physics/0608255`.

Brandes, U. and M. Mader (1012). A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Proceedings of the International Symposium on Graph Drawing (GD'11)*, Volume 7034 of *LNCS*, pp. 99–110. Springer.

Burch, M., C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf (2011). Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics 17*(12), 2344–2353.

Diehl, S. and C. Görg (2002). Graphs, they are changing. In *Proceedings of the International Symposium on Graph Drawing (GD'02)*, Volume 2528 of *LNCS*, pp. 23–30. Springer.

Eagle, N. and A. S. Pentland (2005). CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality.

Erten, C., P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee (2004). GraphAEL: Graph animations with evolving layouts. In *Proceedings of the International Symposium on Graph Drawing (GD'03)*, Volume 2912 of *LNCS*, pp. 98–110. Springer.

Frishman, Y. and A. Tal (2008). Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics 14*(4), 727–740.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman.

Görg, C., P. Birke, M. Pohl, and S. Diehl (2004). Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'04)*, Volume 3383 of *LNCS*, pp. 228–238. Springer.

Hachul, S. and M. Jünger (2006). An experimental comparison of fast algorithms for drawing general large graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'05)*, Volume 3843 of *LNCS*, pp. 235–250. Springer.

Haverkort, H. J. and F. van Walderveen (2010). Locality and bounding-box quality of two-dimensional space-filling curves. *Computational Geometry, Theory and Applications 43*(2), 131–147.

Holten, D. (2006). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics 12*(5), 741–748.

Hu, Y., S. G. Kobourov, and S. Veeramoni (2012). Embedding, clustering and coloring for dynamic maps. In *Proceedings of the 5th IEEE Pacific Visualization Symposium (PacificVis 2012)*, pp. 33–40.

Koren, Y. and D. Harel (2002). A multi-scale algorithm for the linear arrangement problem. In *28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002)*, Volume 2573 of *LNCS*, pp. 296–309. Springer.

Kumar, G. and M. Garland (2006). Visual exploration of complex time-varying graphs. *IEEE Transactions on Visualization and Computer Graphics 12*(5), 805–812.

Muelder, C. (2011). *Advanced Visualization Techniques for Abstract Graphs and Computer Networks* . Dissertation, University of California, Davis.

Muelder, C. and K.-L. Ma (2008). Rapid graph layout using space filling curves. *IEEE Transactions on Visualization and Computer Graphics 14*(6), 1301–1308.

Newman, M. E. J. and M. Girvan (2004). Graph clustering. *Physical Review E 69*(026113).

Noack, A. (2008). Modularity clustering is force-directed layout. *CoRR abs/0807.4052*.

North, S. C. (1996). Incremental layout in DynaDAG. In *Proceedings of the International Symposium on Graph Drawing (GD'95)*, Volume 1027 of *LNCS*, pp. 409–418. Springer.

Ogawa, M. and K.-L. Ma (2010). Software evolution storylines. In *Proceedings of the ACM 2010 Symposium on Software Visualization (SoftVis'10)*, pp. 35–42.

Petit, J. (2001). Experiments on the minimum linear arrangement problem. Technical Report LSI-01-7-R, Universitat Politecnica de Catalunya, Departament de Llenguatges i Sistemes Informatics.

Purchase, H., E. Hoggan, and C. Görg (2007). How important is the "mental map"? - an empirical investigation of a dynamic graph layout algorithm. In *Proceedings of the International Symposium on Graph Drawing (GD'06)*, Volume 4372 of *LNCS*, pp. 184–195. Springer.

Purchase, H. and A. Samra (2008). Extremes are better: Investigating mental map preservation in dynamic graphs. In *Proceedings of the 5th International Conference on Diagrammatic Representation and Inference (Diagrams 2008)*, Volume 5223 of *LNCS*, pp. 60–73. Springer.

Saffrey, P. and H. Purchase (2008). The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Proceedings of the 9th Australasian User Interface Conference (AUIC2008)*, pp. 85–93.

Sallaberry, A., C. W. Muelder, and K.-L. Ma (2013). Clustering, visualizing, and navigating for large dynamic graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'12)*, LNCS. Springer (to appear).

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review 1*(1), 27–64.

Tanahashi, Y. and K.-L. Ma (2012). Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics 18*(12), 2679–2688.

Tufte, E. R. (1990). *Envisionning Information*. Graphics Press.

van Ham, F. and J. J. van Wijk (2004). Interactive visualization of small world graphs. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'04)*, pp. 199–206.

Résumé

L'analyse et la visualisation de graphes dynamiques est un problème difficile. Une méthode de clustering que nous avons développée lors d'un précédent travail peut être appliquée à de tels graphes afin de générer des visualisations interactives à la fois stables et de bonne qualité. Cependant, l'implémentation existante est naïve et non optimisée. Dans cet article, nous présentons de nouveaux algorithmes pour améliorer à la fois les résultats du clustering dynamique et la rapidité des calculs. Nous comparons les résultats et le rendement par rapport à la méthode précédente.