Visualizing Streaming Text Data with Dynamic Graphs and Maps

Emden R. Gansner, Yifan Hu, and Stephen North

AT&T Labs - Research, Florham Park, NJ, USA
{erg,yifanhu,north}@research.att.com

Abstract. The many endless rivers of text now available present a serious challenge in the task of gleaning, analyzing and discovering useful information. In this paper, we describe a methodology for visualizing text streams in real-time modeled as a dynamic graph and its derived map. The approach automatically groups similar messages into "countries," with keyword summaries, using semantic analysis, graph clustering and map generation techniques. It handles the need for visual stability across time by dynamic graph layout and Procrustes projection techniques, enhanced with a novel stable component packing algorithm. The result provides a continuous, succinct view of evolving topics of interest. To make these ideas concrete, we describe their application to an online service called TwitterScope.

1 Introduction

With increased use of social media (Facebook, Twitter, and non-English equivalents such as Weibo), the age of big data is upon us. Some problems concerning big data are characterized solely by size, *e.g.*, draw a Facebook friendship graph. Other data is better modeled as a stream of small packets or messages, *e.g.*, Facebook posts, cell phone messages, or Twitter tweets. Here the problem of size relates to the rate at which these events occur. Taking Twitter as an example, one estimate gave, on average, three thousand tweets per second in February 2012, a six-fold increase over the same month two years earlier. Creating models, algorithms and tools to make sense of massive volumes of streaming data is an active area of research.

One approach is to provide a methodology for a real-time visualization and summation of the data that will provide an overview of the data and how it is changing. For this, we must take several factors into account. First, and most obvious, the velocity of the data means that it arrives in an almost continuous fashion. We need to handle this data stream efficiently, merging each packet with existing data and updating the visualization in a way that preserves the user's mental map.

Second, messages that are close in time often exhibit similar characteristics. For example, a discussion may arise in Twitter generating considerable cross-copying, or rewriting, of similar content. Our understanding can be greatly improved if related packets are appropriately grouped, categorized and summarized.

Finally, while a visual and semantic summary can provide an overview of the data stream, the user may also need tools to explore details once a topic of interest is identified.

In this paper, we propose a technique for visualizing and analyzing streaming packets viewed as a dynamic graph, and its instantiation in the form of TwitterScope, a system for visualizing Twitter messages. Our approach has the following features and novel algorithmic components:

- A succinct visual classification and textual summary, with details on demand. Our system automatically groups tweets into topic areas, thus helping the user to discover emerging and important events. Individual tweets can be revealed by rollovers, and the original source and link can be obtained by a mouse click.
- Real-time monitoring and dynamic visualization. We continuously monitor the Twitter message stream, pushing new items to the visual foreground. In doing so, our approach uses a dynamic graph layout algorithm and Procrustes projection to ensure that the layout is stable.
- A new stable graph packing algorithm for disconnected components. When unrelated clusters of tweets form separate components, we apply the packing algorithm to ensure that the visualization is stable even on a component level.
- Our system also allows the user to shift the display to an earlier point in time to explore what was discussed in the past.

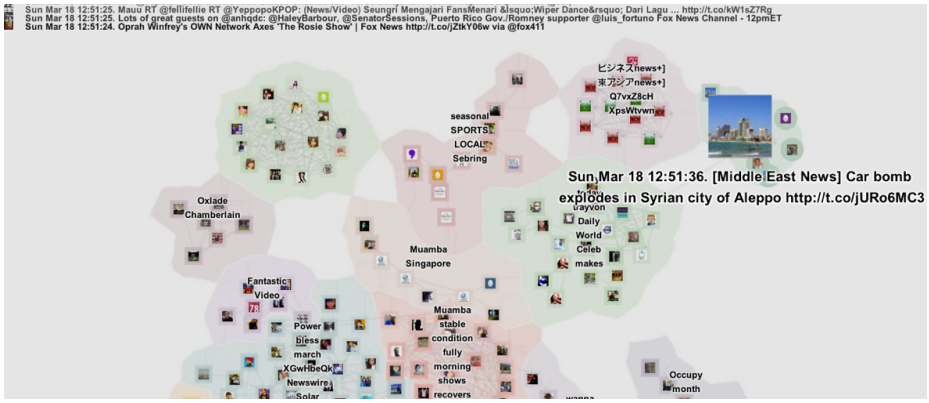


Fig. 1. Screen shot of the TwitterScope visualization system on March 18, 2012.

We describe our approach as instantiated in a system called TwitterScope. TwitterScope runs in any HTLM5 client, and is thus accessible to a broad audience. It can be run either in passive mode for monitoring topics of interest, or as an active tool for exploring Twitter discussions on any topic. Fig. 1 shows a typical snapshot. TwitterScope can be run at `http://bit.ly/HA6KIR`.

2 Related Work

Visualizing streams of text is an active area of research, especially with the growth of social media and related interest in gaining insight about this data. Šilić [26] gives a

recent survey. Much of this work has a different focus from ours, dealing with the statistical analysis and presentation of topics or terms, focusing on the emergence of topic events [7], and often relying on off-line processing [23]. Our approach focuses on the visualization of the text messages themselves, arriving in real-time. Topic information emerges from the display of the messages.

TopicNets [19] analyzes large text corpora off-line. The main view is of a document-topic graph, though the system also allows aggregate nodes. Topic similarity is determined using Latent Dirichlet Allocation (LDA), with topics then positioned using multidimensional scaling. After the topics are fixed, documents are positioned using force-directed placement. The time dimension is handled by a separate visualization, with documents placed chronologically around a (broken) circle, and connected to related topic nodes placed inside the circle.

Alsakran *et al.* offer another approach in STREAMIT [1]. This system displays messages or documents positioned using a spring-based force-directed placement, where the ideal spring length is based on similarity of keyword vectors of documents. Scalability is achieved by parallelization on a GPU. Clustering is based on geometry, a second-order attribute, rather than directly on content. Clusters are labeled with the title of the most recent document joining the cluster.

Our work differs from the above in that we use a map metaphor to visualize clusters. We do not assume that the underlying content graph is connected. Our proposal supports monitoring text streams in real-time.

Focusing on the underlying layout algorithms, we note the large body of prior research on dynamic graph layout, in which the goal is to maintain a stable layout of a graph as it is modified via operations, typically insertion or deletion of nodes and edges. Brandes and Wagner adapt a force-directed model to dynamic graphs using a Bayesian framework [5], while Herman *et al.* [20] rely on good static layouts. Diehl and Görg [8] consider graphs in a sequence to create smoother transitions. Brandes and Corman [3] present a system for visualizing network evolution in which each modification is shown in a separate layer of a 3D representation, with nodes common to two layers represented as columns connecting the layers. Most recently, Brandes and Mader [4] studied offline dynamic graph drawings and compared various techniques such as anchoring node positions or linking nodes from successive time steps. While the issue of mental map preservation adopted in such approaches remains an important research issue, we note that in our application we want to avoid distortion of the layout for the purpose of visual stability. This is because such distortions introduce biases that can persist over time. In addition we require a fast, online procedure for dynamic layout.

Although there does not appear to have been much previous work on stably packing complex objects (Section 4), there has been much work on the related problem of stably removing node overlap (*e.g.*, [9, 13, 17]). This has been limited to removing overlaps of simple shapes, and not aimed at packing them. On the other hand, there has also been work on optimal unrestricted placement of complex shapes. Freivalds *et al.* [12] proposed a polyomino-based algorithm for packing disconnected components.

3 Dynamic Visualization

Our aim is to provide a visual search engine, presenting a collection of messages matching a search expression. Given a collection of messages, we would like to make an overview that helps people to make sense of it – how many messages are there? How are they related? What are the groups or clusters, which messages and authors belong to each, and what topics are discussed? How does interest in topics change over time? Finally, we would like to allow the user to drill down to the individual messages after she has identified interesting clusters, and move the display back in time to explore how a topic emerged.

3.1 Semantic Analysis

The first step is to cluster the messages into sub-topics using semantic analysis. One of the most effective methods for the semantic analysis of textual data is Latent Dirichlet Allocation (LDA) [2]. A simpler and computationally cheaper alternative is to calculate similarity using word counts using term frequency-inverse document frequency (tf-idf).

Before computing similarity using either LDA or tf-idf, we first clean the text of the messages, removing markup notations and stop words, and leaving only plain content-carrying words. Details of our approach with Twitter data are given in [14].

Comparing LDA and tf-idf, the former is sophisticated, and potentially more accurate. But in our application, we found LDA not any better than td-idf for identifying meaningful clusters. Sometimes LDA clusters messages that have no words in common, because LDA treats them as belonging to the same topic. The problem is that with short messages such as tweets, these assignments are not always meaningful. With tf-idf, messages in the same cluster must share at least some words, making the cluster easier to interpret, even if it is not always semantically "correct." For example, we observed that tf-idf assigned high similarity to the tweets, "Everything I know comes through visualization. Third eye senses" and "Many eyes: create visualization," among a data set of many tweets containing the word "visualization," because only those two tweets also contained the word "eye." Overall, like other researchers, we found that semantic analysis of very short text packets, such as tweets, is a challenging task [22], and will likely be studied for years to come.

After the similarity between messages is calculated, we apply a threshold (default 0.2) to the similarity matrix to ignore links between messages with low similarity. While this is not essential to the remaining steps described in this section, we found that applying this threshold can turn completely unrelated clusters into disconnected components in the graph of messages. This helps in making each of the clusters more rounded, and the map representation more aesthetic, but also introduces the complication of packing components in a stable manner, on top of the need for stability within components. We will discuss these issues in the rest of this and the following section.

3.2 Clustering and Mapping

After the similarities between messages are calculated, we treat the collection of messages as a graph – each message is a node, and every pair of messages is connected by

an edge whose strength is given by the similarity between the two messages. A strength of zero corresponds to no edge being present. We draw the graph by multidimensional scaling (MDS), so that similar messages are placed close to each other. For visualization, each node has an associated glyph or icon, such as thumbnail pictures for tweets. We then apply the PRISM algorithm [13], which removes node overlaps while maintaining their relative proximity in the input layout.

We limit the number of messages displayed depending on the available display area. Typically in a browser we show up to 500 tweets. Because we filter out similarities below a threshold, the graph may be very sparse or disconnected. We therefore also filter out singleton components.

We apply modularity clustering [24] to identify clusters of messages, using edge weights proportional to message similarities. After finding clusters, one possibility is to highlight them by assigning colors. We go one step further, and use a map metaphor [11] to depict clusters: messages are treated as towns and cities, where similar messages are close. Clusters of highly related messages then form countries, each enclosed by a boundary, and each country is assigned a color as different as possible from its neighbors within a defined palette [15].

As an example, in Fig. 1, we see part of a map, a screen shot of TwitterScope taken on Sunday March 18, 2012, related to the keyword "news." Each "city" represents a tweet, and is shown as a thumbnail picture of the person who posted the tweet. There is a small country on the right of the figure relating to news events about Syria, with one tweet highlighted. There is also a country with a Japanese keyword summary to the west. To the south of the Japanese topic is one with the keyword "Trayvon," relating to the tragedy of February 26 in which a 17-year-old unarmed African-American was shot by neighborhood watcher George Zimmerman. Three weeks later, the alleged shooter remained free, and this caused much discussion, and would soon make national news, as discussed further in [14]. A small gray country in the southeast of the map with summary "Occupy month" contain tweets about the 6-month anniversary of the "Occupy Wall Street" movement on that day. There are also two countries connected by edges in the center with summary words "Muamba." They are related to the incident in which the English football player Fabrice Muamba suffered a cardiac arrest during a FA Cup quarter-final in London. This event took place on the previous evening, therefore there are many tweets about it. (Messages on these topics continued to form large clusters for several more days.) Thus, at a glance, we immediately see the themes of discussions in Twitter space about "news."

3.3 Dynamic Layout

The above steps describe a static layout method. Our goal, however, is to handle a dynamic graph of text message streams, with the visualization updated frequently, say, every minute. To put this in context, consider the rate of messages in a Twitter stream. The rate clearly will depend on the initial filtering. We found that, for example, there were on average around 400 new tweets containing the keyword "news" each minute, sometimes spiking to 1200 tweets. On the other hand, for the keyword "tech," there were about 40 tweets per minute, and for "visualization," only about one new tweet per minute. Since we limit the number of messages shown, active streams, such as "news"

in Twitter, require the complete replacement of every tweet by new ones at each update, and often the clusters are about completely different topics. In these situations, it may not be as important (or possible) to maintain a stable mental map. In smaller streams, for example, "visualization," only a portion of the messages will be replaced, so it is important for the remaining messages to appear at approximately the same positions, to preserve a user's mental map.

The problem of laying out dynamic graphs to preserve a user's mental map has been studied previously. One approach is to "anchor" some, or all, of the nodes, or to limit node movement by adding artificial edges linking graphs across successive time frames [10]. However, such approaches can introduce biases not in the underlying data, and that persist over time.

In our approach, we preserve the mental map via a two-step approach. Before laying out a new graph, we initialize each node in the new graph with its previous position, if it had one, otherwise at the average of its neighbors in the previous graph, and then apply MDS by local optimization. We found that MDS tends to be quite stable in terms of maintaining relative node positions, though the graph may rotate, or expand or shrink. We therefore apply a Procrustes transformation [25] to the node coordinates after MDS layout. The Procrustes transformation is a well-known technique [6], which attempts to find the best alignment of two inputs via rotation, translation, and scaling.

In our application, we found that when the common set of nodes are close in the first layout, but are further apart in the second one (due to the insertion of other nodes between them), a Procrustes transformation tends to scale down the new layout. In the context of abstract graph layout where nodes are represented as points, this is not a problem. However, in our setting, nodes are represented by figures having a fixed width and height, so shrinking the layout can cause node overlaps. For that reason, we limit the transformation to rotations and translations, and use a scale factor of 1.

4 An Algorithm for Stable Packing of Disconnected Components

The dynamic layout algorithm described in the previous section ensures that the positions of nodes are stable in successive time frames as long as the graph is connected. On the other hand, when there are multiple disconnected components the situation is more complex. While the dynamic layout algorithm aligns drawings of the same component in consecutive time frames, this alignment does not consider potential overlaps between components.

A standard technique for arranging disconnected components is to pack them to minimize total area and avoid wasted space, while avoiding overlaps. This problem has been studied in graph drawing [12,18], in VLSI design, and in space planning for building and manufacturing, where it is known as the floor-planning problem.In these cases the emphasis is on maximal utilization of space. A widely used algorithm for packing disconnected graph components is that of Freivalds *et al.* [12]. It represents each disconnected component by a collection of polyominoes (tiles made of squares joined at edges). The polyominoes are generated to cover node bounding boxes, as well as graph edges. The algorithm places components one at a time, from large to small. It attempts to place the center of polyominoes at consecutive discrete grid locations in increasing

distance from the origin, until it finds one that does not overlap any components already placed. The algorithm yields packings that tend to utilize space efficiently, even though the computational complexity scales quadratically with the number of components and average number of polyominoes per component.

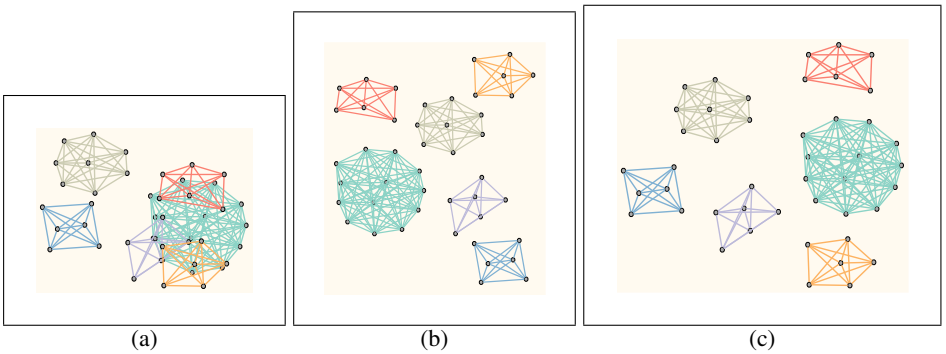


Fig. 2. Illustration of the stable packing problem. (a) Graph with overlapping components. (b) Standard packing algorithms ignore relative positions. (c) Result of proposed stable packing algorithm.

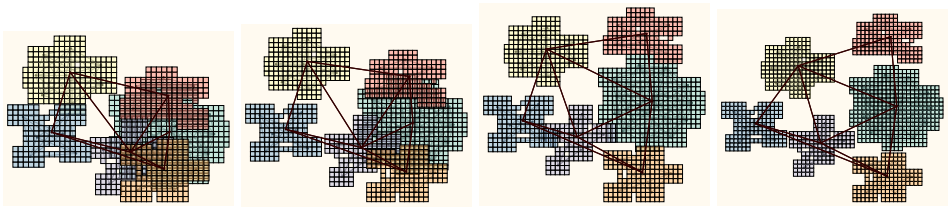
In the context of dynamic graph visualization, no graph packing algorithms that we are aware of were designed with maintaining layout stability in mind. For example, Fig. 2 (a) shows six disconnected components, some of them overlapping each other. If we feed this graph to a standard packing algorithm (in this case, `gvpack` in `Graphviz` [16]), we obtain Fig. 2 (b). The proximity relations are completely lost. For example, the orange component, which was below the large green component, is now to the upper right of it.

In this section we describe a packing algorithm that attempts to maintain layout stability. The algorithm extends the label overlap removal algorithm of Gansner and Hu [13], by making it work on graph components of arbitrary shape, not just rectangular node labels. In addition, the new algorithm also attempts to remove "underlap" – unnecessary space between components; thus it is a packing algorithm, not only an overlap removal method.

We begin by adapting the polyomino algorithm for component packing. For each component, polyominoes are used to cover nodes and its labels, as well as all the edges. Fig. 3 (a) shows the overlapped components of Fig. 2 (a), now covered by polyominoes.

We use the polyominoes to help detect collisions between components. We set up an underlying mesh (not shown). The mesh cell size may or may not be the same as the polyomino size. When polyominoes move due to adjustment to the position of the corresponding components, we hash the mesh cells that intersect with the polyominoes of one component, and detect a collision of this component with another component by checking the array hash table against polyominoes of the second component.

To achieve our goal of removing overlaps and underlaps, while preserving the proximity relations of the initial component configuration, we first set up a rigid "scaffolding" structure so that while components can move around, their relative positions are maintained as much as possible. This scaffolding is constructed using an approximate *proximity graph* that is rigid, in the form of a Delaunay triangulation (DT). The

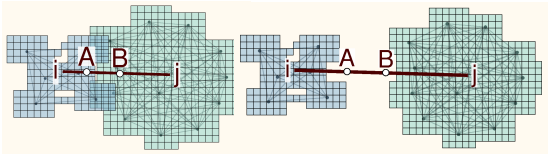


(a) (b) (c) (d)

Fig. 3. Illustration of our stable packing algorithm. (a) Polyominoes are generated to cover nodes and edges of the components in Fig. 2. A triangulation is carried out over the centers of the components. (b), (c) and (d) After 3 iterations, the overlap has been eliminated while the relative position of components remains stable. The resulting graph is shown in Fig. 2(c)

triangulation is carried out using the center of each component. Fig. 3 (a) shows the triangulations as thick dark lines, connecting the centers of each component.

Once we find a DT, we search every edge in it and test for component overlaps or underlaps along that edge. If there is an overlap, we compute how far the components should move apart along the direction of the edge to remove the overlap. If there is an underlap, we compute how far the components should move toward each other along the edge to close the gap. The distance of the move can be calculated using an iterative algorithm, where two overlapping components are first separated enough so that they do not overlap, then a simple bisection algorithm is applied until a sufficiently accurate measure of the distance of travel is achieved. In an implementation, though, we can use a cheaper though less accurate measure. Suppose for one of these edges, the two endpoint components are i and j. Let the edge be $\{i, j\}$, and the coordinates of the two end points be x_i^0 and x_j^0. We project the corners of each polyomino onto the line $x_i^0 \rightarrow x_j^0$. If the two components overlap, it is certain that the projection of the components will overlap. The amount of overlap on line $x_i^0 \rightarrow x_j^0$ is denoted by $\delta_{ij} > 0$. On the other hand, if the two components are separated, it is likely (but not always true) that the projection of the components are separated by a gap. We denote this underlap (negative overlap) as $\delta_{ij} < 0$. If $\delta_{ij} > 0$ but no collision is detected, we set $\delta_{ij} = 0$. We define the *ideal length factor* to be $t_{ij} = 1 + \frac{\delta_{ij}}{\|x_i^0 - x_j^0\|}$.



(a) (b)

Fig. 4. Illustration of ideal length factors. (a) Components i and j overlap. Projection of polyominoes of i on the line $i \rightarrow j$ extends as far right as A; projection of polyominoes of j on the line $i \rightarrow j$ extends as far left as B. Since $|AB|$ is $1/3$ of $|ij|$, the ideal length factor is $1 + 1/3 = 1.33$. (b) Components i and j underlap. Since $|AB|$ is $1/4$ of $|ij|$, the ideal length factor is $1 - 1/4 = 0.75$.

Component overlap and underlap can be removed if we expand or shrink the edge between these components; see Fig. 4. Therefore we want to generate a layout such that an edge in the proximity graph has the ideal edge length close to $t_{\text{ij}}\|x_i^0 - x_j^0\|$. In other words, we want to minimize the following stress function

$$\sum_{(i,j)\in E_P} w_{\text{ij}} \left(\|x_i - x_j\| - d_{\text{ij}}\right)^2. \tag{1}$$

Here $d_{\text{ij}} = s_{\text{ij}}\|x_i^0 - x_j^0\|$ is the ideal distance for the edge $\{i, j\}$, s_{ij} is a scaling factor related to the overlap factor t_{ij} (see below), $w_{\text{ij}} = 1/\|d_{\text{ij}}\|^2$ is a weighting factor, and E_P is the set of edges of the proximity graph.

Given that the DT is a planar graph with at most $3k-6$ edges, where k is the number of components, the stress function (1) has no more than $3k - 6$ terms. Furthermore, the rigidity of the triangulated graph provides a good scaffolding that constrains the relative position of the components and helps to preserve the global structure of the original layout.

Trying to removing overlaps and underlaps too quickly with the above model with $s_{\text{ij}} = t_{\text{ij}}$ can be counter-productive. Imagine the situation where components form a regular mesh configuration. Initially, the components do not overlap. Then assume that component i in the center grows quickly, so that it overlaps severely with its nearby components, while the other components remain unchanged. Suppose components i and j form an edge in the proximity graph, and they overlap. If we try to make the length of the edge equal to $t_{\text{ij}}\|x_i^0 - x_j^0\|$, we will find that t_{ij} is much larger than 1, and an optimal solution to the stress model keeps all the other vertices at or close to their current positions, but moves the large component i outside the mesh entirely, to a position that avoids any overlap. This is not desirable because it destroys the relative position information. Therefore we damp the overlap and underlap factor by setting $s_{\text{ij}} = \max(s_{\min},\ \min(t_{\text{ij}}, s_{\max}))$ and try to remove overlaps and underlaps gradually. Here $s_{\max} > 1$ is a parameter limiting the amount of overlap, and $s_{\min}$ a parameter limiting the amount of underlap that we are allowed to remove in one iteration. In experiments, we found that $s_{\max} = 1.5$ and $s_{\min} = 0.8$ work well.

After minimizing (1), we arrive at a layout that may still have component overlaps and underlaps. We then regenerate the proximity graph using DT, calculate the ideal length factor along the edges of this graph, and rerun the minimization. This defines an iterative process that terminates when there are no more overlaps or underlaps along the edges of the proximity graph. In addition, to avoid having components oscillate between overlap and underlap, we set $s_{min} = 1$ after a fixed number N of iterations. Based on our experiences, we use $N = 10$.

For many disconnected graphs, the above algorithm yields a drawing that is free of component overlaps, and utilizes the space well. For some graphs, however, especially those with components having extreme aspect ratios, overlaps may still occur, even though no overlaps are detected along the edges of the proximity graph.

To overcome this situation, once the above process has converged so that no more overlaps are detected over the proximity graph edges, we hash the components one at a time in the background mesh to detect any overlaps, and augment the proximity graph with additional edges, where each edge consists of a pair of components that

overlap. We then re-solve (1). This process is repeated until no more component overlaps are found. We call this the proximity graph-based STAble PACKing (STAPACK) algorithm. Algorithm 1 gives a detailed description of this algorithm. The complexity of the algorithm is discussed in [14].While we do not have a proof that this procedure converges, in practice it always does so, often very quickly. We impose a limit of 1000 iterations, although the algorithm has not been observed to ever take this many iterations. Overall the algorithm runs quickly if the underlying mesh for the polyominoes is of dimension $m \times n$, with m and n in the range of $100 - 500$.

Algorithm 1 Proximity graph-based stable packing algorithm (STAPACK)

Input: Disconnected components, and their centers x_i^0,
repeat
  Form a proximity graph G_P of x^0 by Delaunay triangulation.
  Find the ideal distance factors along all edges in G_P.
  Solve the stress model (1) for x. Set $x^0 = x$.
until (no more overlaps along edges of G_P)
repeat
  Form a proximity graph G_P of x^0 by Delaunay triangulation.
  Find all component overlaps using background mesh hashing. Augment G_P with edges from component pairs that overlap.
  Find the ideal length factor along all edges of G_P.
  Solve the stress model (1) for x. Set $x^0 = x$.
until (no more overlaps)

Fig. 3 (b-d) shows the result of 3 iterations on the overlapping components in Fig. 3 (a). As can be seen, in this case the process converges quickly and the result in Fig. 3 (d) packs the components with no overlaps, while proximity is well-preserved.

5 TwitterScope System Overview

We have implemented a prototype system, called TwitterScope, that employs the visualization technique described above. Its aim is to provide a dynamic view of tweets related to a given keyword or set of keywords. Fig. 5 shows an overview of TwitterScope. It collects data using Twitter's streaming API methods. Data is collected continuously and stored permanently. This supports queries over any time period after the collector started running, which is crucial for repeatable experiments and historical review.

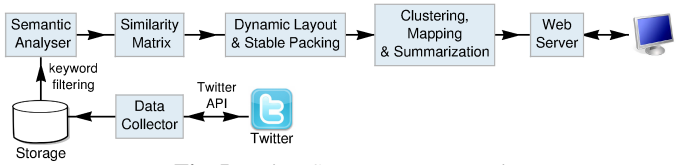


Fig. 5. TwitterScope system overview

In parallel with the data collector, a semantic analyzer finds stored tweets matching a specific keyword, and outputs a similarity matrix. The system uses the techniques described above to produce a map of the tweets. In addition, the system produces a

list of the top keywords in the tweets of a cluster which can be used to summarize the cluster. The final results are pushed to the web server as JSON files, and show up on the user's browser continuously.

From the user's point of view, TwitterScope has two modes. In passive mode, it runs in the background and makes overviews for monitoring the given keywords. In active mode, the interface allows the user to interactively explore the tweet landscape, drilling in for specific details or exploring the changes over time. Further details of the system are given in [14].

6 Conclusion

We have presented a general technique for visualizing in real-time a dynamic graph of text packet streams. It clusters packets into subtopics based on semantic analysis and, using a geographic map metaphor, renders subtopics as countries, with an attached textual precis. The approach relies on placement algorithms that promote the stability of the drawing to aid a user's comprehension. In particular, we introduced a new packing algorithm to address component stability. The resulting system for Twitter messages, TwitterScope, presents a real-time streaming visualization of tweets with messages grouped into clusters and grouped into countries. Users can drill down into individual tweets, or inspect past tweets by using the time series chart.

We see two obvious algorithmic enhancements for our current approach, both dealing with visual stability. First, when we recreate the map, colors for countries are assigned without much consideration to stability. There has been recent work on achieving color assignment stability that we would like to incorporate [21]. Second, for certain topics, when we perform a periodic packing refresh, the map may change significantly. It would be good to avoid this discontinuity, either algorithmically or visually.

References

1. J. Alsakran, Y. Chen, D. Luo, Y. Zhao, J. Yang, W. Dou, and S. Liu. Real-time visualization of streaming text with a force-based dynamic system. *IEEE Computer Graphics and Applications*, 32(1):34–45, 2012.
2. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.
3. U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. In *IEEE INFOVIS'02*, pages 145–151, 2002.
4. U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In M. J. van Kreveld and B. Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2011.
5. U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In G. D. Battista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 1997.
6. T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Chapman and Hall/CRC, 2000.
7. W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. Gao, H. Qu, and X. Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2412–2421, 2011.

8. S. Diehl and C. Görg. Graphs, they are changing. In S. G. Kobourov and M. T. Goodrich, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–30. Springer, 2002.
9. T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *Proc. 13th Intl. Symp. Graph Drawing (GD '05)*, volume 3843 of *LNCS*, pages 153–164. Springer, 2006.
10. C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. Graphael: Graph animations with evolving layouts. In G. Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 98–110. Springer, 2003.
11. S. I. Fabrikant, D. R. Montello, and D. M. Mark. The distance-similarity metaphor in region-display spatializations. *IEEE Computer Graphics & Application*, 26:34–44, 2006.
12. K. Freivalds, U. Dogrusöz, and P. Kikusts. Disconnected graph layout and the polyomino packing approach. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 378–391. Springer, 2001.
13. E. R. Gansner and Y. Hu. Efficient node overlap removal using a proximity stress model. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2008.
14. E. R. Gansner, Y. Hu, and S. C. North. Visualizing streaming text data with dynamic graphs and maps. `http://arxiv.org/abs/1206.3980`, 2012.
15. E. R. Gansner, Y. F. Hu, and S. G. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 201 – 208, 2010.
16. E. R. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice & Experience*, 30:1203–1233, 2000.
17. E. R. Gansner and S. C. North. Improved force-directed layouts. In *Proc. 6th Intl. Symp. Graph Drawing (GD '98)*, volume 1547 of *LNCS*, pages 364–373. Springer, 1998.
18. D. Goehlsdorf, M. Kaufmann, and M. Siebenhaller. Placing connected components of disconnected graphs. In S.-H. Hong and K.-L. Ma, editors, *APVIS: 6th International Asia-Pacific Symposium on Visualization 2007, Sydney, Australia, 5-7 February 2007*, pages 101–108. IEEE, 2007.
19. B. Gretarsson, J. O'Donovan, S. Bostandjiev, T. Höllerer, A. U. Asuncion, D. Newman, and P. Smyth. Topicnets: Visual analysis of large text corpora with topic modeling. *ACM TIST*, 3(2):23, 2012.
20. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.
21. Y. Hu, S. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In *Proceedings of IEEE Pacific Visualization Symposium*, 2012.
22. O. Jin, N. N. Liu, K. Zhao, Y. Yu, and Q. Yang. Transferring topical knowledge from auxiliary long texts for short text clustering. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 775–784, New York, NY, USA, 2011. ACM.
23. A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Twitinfo: aggregating and visualizing microblogs for event exploration. In D. S. Tan, S. Amershi, B. Begole, W. A. Kellogg, and M. Tungare, editors, *CHI*, pages 227–236. ACM, 2011.
24. M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103:8577–8582, 2006.
25. R. Sibson. Studies in the robustness of multidimensional scaling: Procrustes statistics. *Journal of the Royal Statistical Society, Series B (Methodological)*, 40:234–238, 1978.
26. A. Silic and B. D. Basic. Visualization of text streams: A survey. In R. Setchi, I. Jordanov, R. J. Howlett, and L. C. Jain, editors, *KES (2)*, volume 6277 of *Lecture Notes in Computer Science*, pages 31–43. Springer, 2010.