Visualizing Dynamic Data with Maps

Daisuke Mashima* Georgia Institute of Technology
Stephen G. Kobourov† University of Arizona
Yifan Hu‡ AT&T Labs Research

ABSTRACT

Maps offer a familiar way to present geographic data (continents, countries), and additional information (topography, geology),can be displayed with the help of contours and heat-map overlays. In this paper we consider visualizing large-scale dynamic relational data by taking advantage of the geographic map metaphor. We describe a system that visualizes user traffic on the Internet radio station last.fm and address challenges in mental map preservation, as well as issues in animated map-based visualization.[1]

Index Terms: H.5.1 [Information Interface and Presentation(I.7)]: Multimedia Information Systems—Animation; I.3.6 [Computer Graphics]: Methodology and Techniques

1 INTRODUCTION

The use of a geographic map metaphor for visualizing relational data was originally described in the context of visualizing recommendations, with TV shows and the similarity between them as the underlying data [12]. This approach combines graph layout and graph clustering, together with appropriate coloring of the clusters and creating boundaries based on clusters and connectivity in the original graph. A companion paper describes the algorithmic details of this map generation approach, and how it can be generalized to any relational data set [13]. But in both cases the underlying data is static. The problem becomes harder when we would also like to visualize some underlying process. For example, instead of showing a static map of popular TV shows, we would like to see the evolution of this data over the course of one year, and discover which shows become more (or less) popular over that time period.

In this paper we explore a new way to visualize dynamic relational data with the help of the geographic map metaphor. Some of the challenges identified along the way include: preservation of the viewer's mental map under the dynamics in the data, readability of each individual layout, and effective visualization of the changes happening on the map. We describe one way to address these challenges and present the implementation that visualizes music trends collected from the Internet radio station last.fm (`http://www.last.fm`).

The paper is organized as follows. In Section 2, we overview GMap algorithm we rely on. Then, in Section 3, we discuss the algorithmic pipeline to address challenges discussed above. Section 4 presents our prototype implementation as well as implementational efforts to make the visualization better. Section 5 concludes the paper with future work.

*e-mail: d.mashima@gatech.edu
†e-mail:kobourov@cs.arizona.edu
‡e-mail:yifanhu@research.att.com

[1]All map images and movies presented in this paper are available in high-resolution at `http://www2.research.att.com/~yifanhu/TrendMap/figures`.

1.1 Related Work

In dynamic graph drawing the goal is to maintain a nice layout of a graph that is modified via operations such as inserting/deleting edges and inserting/deleting vertices; see the survey paper by Branke [4]. Brandes and Wagner adapt the force-directed model to dynamic graphs using a Bayesian framework [3]. Diehl and Görg [8] consider graphs in a sequence to create smoother transitions. Brandes and Corman [2] present a system for visualizing network evolution in which each modification is shown in a separate layer of a 3D representation with vertices common to two layers represented as columns connecting the layers. Thus, mental map preservation is achieved by pre-computing good locations for the vertices and fixing the position throughout the layers. Animations as a means to convey an evolving underlying graph have also been used in the context of software evolution [6] and scientific literature visualization [9].

There have been several earlier efforts to visualize the Internet radio station last.fm. Graph-based representations have been used [1], with each artist as a node, similarity relationships denoted by edges, and tags used for grouping and coloring. Even though this visualization contains a good amount of information, such as popularity, similarity, tags, and so on, it suffers in readability due to significant node overlapping and fragmentation of groups. Another last.fm visualization [20] uses self-organizing maps which leads to a 2D grid layout in which similar bands are close to each other. but this approach has high computational complexity and does not scale well to large data sets.

Using maps to visualize non-cartographic data has been considered in the context of spatialization [23]. Map-like visualization using layers and terrains to represent text document corpora dates back to 1995 [24]. The problem of effectively conveying change over time using a map-based visualization was studied by Harrower [15].

Also related is work on visualizing subsets of a set of items. Areas of interest in a UML diagram can be highlighted using a deformed convex hull [5]. Isocontours-based bubblesets can be used to depict multiple relations defined on a set of items [7]. Automatic Euler diagrams, which show the grouping of subsets of items by drawing contiguous regions around them have also been considered [22]. Apart from differences in the algorithms used to generate regions, all of these approaches differ from ours in that they create regions that overlap with each other, whereas we take the map metaphor strictly and assume that regions do not overlap.

Robertson *et al.* [21] evaluate the effectiveness of three trend visualization techniques. The results indicate that animation is not well suited to data analysis, but it is often enjoyable and exciting. Since one of the main goals of our work is to create an appealing and informative visualization for the general public, and not a precise data analysis tool, we believe our use of animation for trend visualization is justified. We attempt to address some of the main shortcomings of animations with the help of strong mental map preservation and a familiar geographic map metaphor.

2 CREATING MAPS FROM GRAPH DATA

We begin with a summary of the GMap algorithm for generating maps from static graphs [13]. The input to the algorithm is a relational data set from which a graph $G = (V,E)$ is extracted. The set

of vertices V corresponds to the objects in the data (e.g., artists) and the set of edges E corresponds to the relationship between pairs of objects (e.g., the similarity between a pair of artists). In its full generality, the graph is vertex-weighted and edge-weighted, with vertex weights corresponding to some notion of the importance of a vertex and edge weights corresponding to some notion of the closeness between a pair of vertices. In the case of music, the importance of a vertex can be determined by the popularity of an artist as derived from the total number of listeners or by the total number of songs played in a given time period. The weight of an edge can be defined by the strength of the similarity between a pair of artists.

In the first step of GMap the graph is embedded in the plane using a scalable force-directed algorithm [10] or multidimensional scaling (MDS) [17]. In the second step, a cluster analysis is performed in order to group vertices into clusters, using a modularity-based clustering algorithm [18]. We use information from the clustering to guide the MDS-based layout. In the third step of GMap, the geographic map corresponding to the data set is created, based on a modified Voronoi diagram of the vertices, which in turn is determined by the embedding and clustering. Here "countries" are created from clusters, and "continents" and "islands" are created from groups of neighboring countries. Borders between countries and at the periphery of continents and islands are created in fractal-like fashion. Finally, colors are assigned with the goal that no two adjacent countries have colors that are too similar. In the context of visualizing dynamic data where the relative change of popularity of important, we also use a heatmap overlay to highlight the "hot" regions. Further geographic components can be added to strengthen the map metaphor. For instance, edges can be made semi-transparent or even modified to resemble road networks. In places where there are large empty spaces between vertices in neighboring clusters, lakes, rivers, or mountains can be added, in order to emphasize the separation.

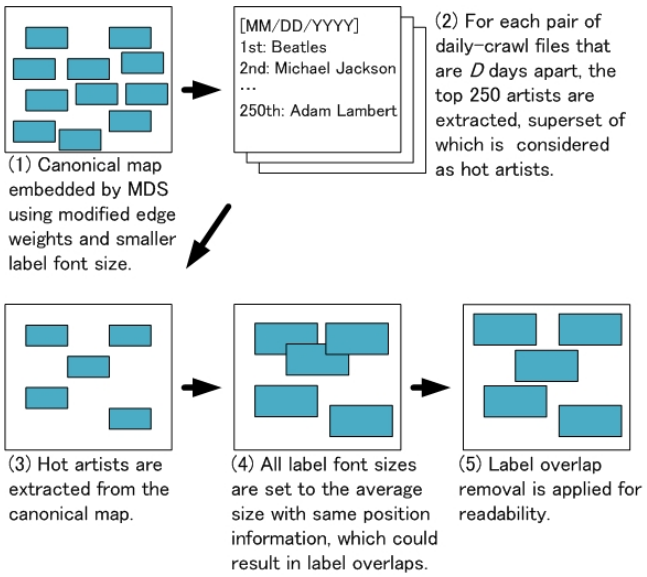


Figure 1: Algorithmic pipeline to create a base map from a canonical map. (1) Canonical map is created with all crawled artists, by embedding them using MDS and smaller-sized label fonts. (2) From each pair of daily-crawled files that are D days apart, 250 artists with the highest playcount increase are extracted. (3) Position data of the hot artists are extracted from the canonical map. (4) All label font sizes are set to the average size. (5) Overlap removal is applied, and the resulting layout is used as a base map.

3 Maps of Dynamic Data

Static maps of relational data lead to visually appealing representations, which show more than just the underlying vertices and edges. Specifically, by explicitly grouping vertices into different colored regions, viewers of the data can quickly identify clusters and relations between clusters. Moreover, this explicit grouping leads to easy identification of central and peripheral vertices within each cluster.

Extending traditional graph drawing algorithms from static to dynamic graphs is a difficult problem. In most proposed solutions, the typical challenges are those of preserving the mental map of a viewer and ensuring readability of each drawing. Changes are visualized by animation, which can be generated by concatenating static maps, thus providing continuity from one layout to the next. Whereas in dynamic graph drawing it is perfectly reasonable to have vertices move from one moment in time to the next, moving "countries" and "cities" within the countries on a map can be confusing and counter-intuitive. Also, if the layout from one time to the next is significantly different, it is likely that viewers will quickly get lost. A common way to deal with this problem is anchoring some vertices that appear in two or more subsequent drawings. Additionally, the way to encode metrics and changes into the map metaphor needs to be considered. Next we describe how we address some of these challenges. The algorithmic pipeline discussed in Section 3.2 to 3.4 is summarized in Figure 1.

3.1 Last.fm Data

As an Internet radio and music community website, last.fm has over 30 million users. Using a music recommender system, last.fm recommends music based on user profiles. Over several years the recommender system has collected information about how one musician is related to another in terms of how many listeners of one also enjoy the other. For each musician, the last.fm website lists related (similar) musicians. For example, Beethoven is considered to have "super similarity" to Mozart, Bach, Brahms, "very high similarity" to Mendelssohn, Schumann, Vivaldi, and so on. The website also provides the number of listeners of each musician. Using this data and daily crawls using the provided API, we create the underlying graph with artists as vertices and with edges determined by the strength of the similarity between the artists at the two endpoints.

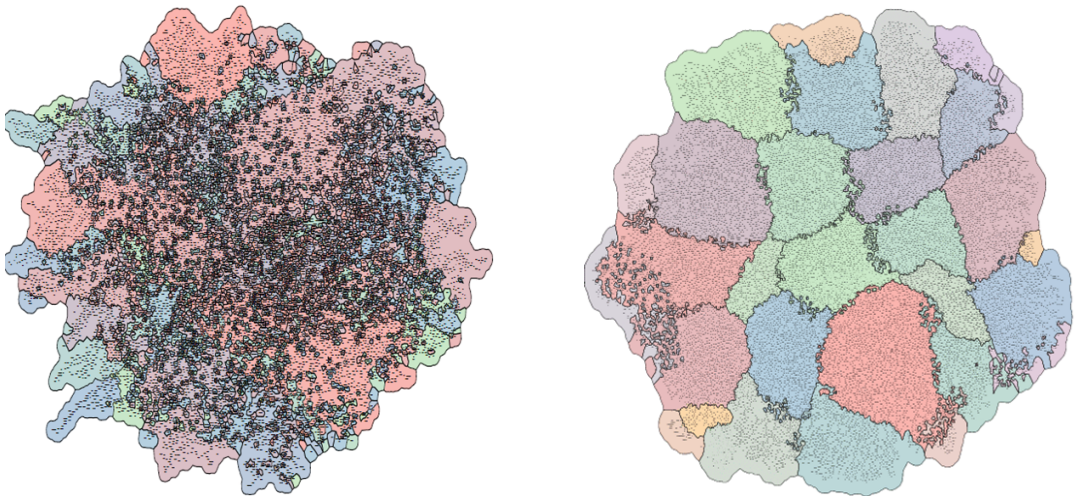
3.2 Mental Map Preservation

Mental map preservation is important when visualizing dynamic data. In general, vertices and edges may appear and disappear over time. If a vertex appears, then disappears, and appears again, it would be desirable to use the same location in the layout. Specifically, in the last.fm data artists that were not in the previous map may suddenly become popular while others may drop off from the top.

To address this problem, we create a "canonical map" that stores the position information of a much larger graph than the subgraph that is actually shown. Then, when displaying a specific subgraph consisting of top artists at a given time, we use the pre-computed position information from the canonical map (Figure 1(1)). In this way, as long as the same canonical map is used, the same artists appear in the same position, thereby helping preserve a viewer's mental map. Updating a canonical map is anyway mandatory to keep up with trend changes, but it can be done less frequently, for instance once a month, as long as the number of artists included is large enough, which most likely contains all artists that could appear in the visualization before the next update.

3.3 Map Readability

Our initial attempt at obtaining a canonical map with GMap of the 18,000 artists crawled from the top artists in last.fm immediately



(a) (b)

Figure 2: (a) Map of 18,000 artists (b) Map of 18,000 artists using adjusted edge lengths

exposed a problem with this approach. We used modularity and MDS for clustering and embedding, respectively. The (embedding, clustering) pairing seemed applicable, given that in the underlying graph the strength of an edge corresponds to the measure of similarity between the two artists it connects. Since the inverse of similarity can be naturally interpreted as a distance, MDS can determine a layout that matches the underlying clustering.

However, the resulting map was far from ideal; see Figure 2 (a). The most conspicuous problem is the fragmentation of countries into disjoint regions. We found that, on average, one cluster (country) is divided into over 100 regions. Even though this canonical map is never intended to be seen by viewers, such fragmentation will negatively affect the readability of resulting visualization. In fact, the placement of the vertices determined by this canonical map led to significant fragmentation even in a map created for the top 500 artists (in terms of the number of listeners). Using a force-directed layout [10] or a LinLog layout [19] in place of MDS resulted in even more fragmentation.

One possibility is that the fragmentation problem is to some extent caused by the independent nature of the clustering and the embedding steps. Therefore, we combined the two steps by using the clustering results as additional input parameters of the embedding process. In other words, based on the clustering results, we increase the edge lengths between artists that belong to different clusters, leading to a much better canonical map; see Figure 2 (b). In this map, fragmentation is significantly reduced although there are irregularities near some country boundaries. In addition, we did not see any fragmentation in a map of top 500 artists created based on a canonical map generated in this way.

It is worth mentioning that GMap uses a label overlap-removal routine [11] to ensure that vertex labels are readable. This is accomplished by moving apart vertices with overlapping labels, but can potentially lead to a vertex near a border between two countries "jumping" into the wrong country. By strengthening the edges between vertices in the same cluster, we help such vertices stay in their own countries. Even though such edge length modification distorts the underlying raw similarity information, most of the resulting layout changes are local.

Additionally, since smaller number of hot artists will be extracted out of a canonical map in the later step, we need to determine node positions densely in order to prevent the resulting map from being too sparse. To do embedding in that way, we chose to use smaller label font sizes that are proportional to the popularity of artists; see Figure 1 (1).

3.4 Mental Map Preserving Node and Label Placement

A general issue regarding the visualization is the number of artists displayed in a single map. As we target a regular computer screen, the number of artists that can be shown depends on how many non-overlapping and readable labels can fit on it. Not surprisingly maps with 1000 labeled artists turned out to be unreadable. Cutting down the number of artists to the top 250 leads to better results.

Even within the top 250 artists, some are much more popular than others. One straightforward way is to represent the popularity of an artist by varying the font size of the labels, as in geographic maps where the names of major cities are drawn with larger fonts than those of smaller towns. To modify the font sizes, we use the following conversion for each artist displayed.

$$ModFontSize_a = BaseSize + Variation * f(pop_a) \qquad (1)$$

where

$$f(pop_a) = \frac{pop_a - AVERAGE_{i \in A}(pop_i)}{MAX_{i \in A}(pop_i) - AVERAGE_{i \in A}(pop_i)} \qquad (2)$$

Here, the set A denotes all artists to be shown on a map, and pop_a indicates a popularity metric (e.g., the number of listeners) of an artist $a \in A$. Note that $f(pop_a)$ in (2) is scaled to be within $[-1,1]$. Therefore the resulting font size $ModFontSize_a$ is in $[BaseSize - Variation, BaseSize + Variation]$, with a mean font size of $BaseSize$. A sample map created under this configuration is shown in Figure 3. Related to font size modification is the timing of the label overlap-removal step [11]. Because we modify the font sizes after the layout of the nodes in the canonical map has been determined, the resized labels could lead to new overlaps in crowded areas, once again making the maps difficult to read. Applying another overlap-removal step, once the labels have been resized, makes the maps readable but at the expense of modifications in the positions of labels from one time frame to the next. Although this process could be effective for the sake of better presentation, the negative side effects (inconsistent label positions between consecutive frames) seem to outweigh the advantages. While such movements of labels over time would draw viewers' attention to an area

Figure 3: The top 250 artists: showing artist popularity through font sizes, while also displaying similarity using the geographic map metaphor.

where there are changes, such movements do not fit our general approach for maintaining a viewer's mental map by having a fixed geographic map as a reference.

In order to benefit from overlap removal without moving labels from frame to frame, we adopted the following approach, summarized in Figure 1. First, we create the canonical map. Second, we form the superset of artists that appear on any of the map frames to be included in the animation. Third, we extract the position information for these artists from the canonical map. Fourth, we set the font sizes of all labels on a map to the average size, i.e. *BaseSize* in formula (1). Fifth, we perform an overlap-removal step and call the final result the "base map" because it is used to create each map frame in the animation. As a result of the pre-processing, the positions of artists and shapes of countries remain unchanged within one animation.

Note, however, that since this base map is generated every time we create an animation, the node positions are not exactly consistent among animations created for different time periods. For example, today's animation and an animation created one week later could have slightly different node positions and country boundaries owing to both the overlap removal and difference in artists to be displayed. But, when those base maps use the same underlying canonical map, such differences are minimal.

When we evaluated the animations created by the above procedure, we found that the lack of easily recognizable differences among maps can be a problem. Too much of a good thing (mental map preservation) can be bad. Specifically, it is difficult to spot the differences, as only the font sizes of some artists are changing between map frames, while other components (e.g., size and shape of countries) remain exactly the same (a sample animation can be found in `http://www2.research.att.com/~yifanhu/TrendMap/figures`). As we would like to keep the mental map of viewers unchanged from one frame to the next, and just changing the font sizes does not convey the changes in the data, we employ another visual cue that is well suited to maps, namely heat-map overlays. We discuss the metric used to create such heat-maps next, and give the detailed procedure for creating them in Section 4.

3.5 Metric for Visualization

A challenge in visualization of dynamic data is defining a suitable metric which allows us to extract "hot" objects (e.g., artists) out of the canonical map and visualizing them meaningfully. Since the suitable metric is highly context-specific, our discussion here focuses on last.fm data and their API. For example, the number of listeners for each artist and the number of times each artist's songs are played (also called playcounts) both seem to be useful. However, these numbers are all cumulative. In other words, artists that have been around for a long time tend to have higher values than newer artists who only recently attracted attention. While such numbers are useful to see long-term popularity, it implies that these values largely depend on the past data, and are not significantly affected by recent and short-term dynamics, which are often of interest to the viewers.

Ideally, both long-term and short-term metrics should be incorporated in the visualization. Thus, while using the *cumulative number of listeners as a long-term metric*, we also consider the short-term one, which is more sensitive to abrupt changes. To prevent the bias by past data, we focus on the difference in these values over a fixed time interval. The ideal interval varies depending on the settings and nature of the target data set. In the case of last.fm data, these numbers are updated weekly, so 7-day or longer interval is appropriate. Our preliminary analysis indicates that playcounts capture the dynamics of the moment well, so in our implementation we use *differences in playcounts as a short-term popularity metric*.

4 Implementation

Following the approach discussed in Section 3, we describe our visualization system applied to last.fm data. In addition, we will discuss how we addressed the four additional challenges posed on animated cartographic maps, namely disappearance (blink and you'll miss it), attention (where to look as the animation is playing), complexity (animated maps try to do too much and end up saying very little), and confidence (viewers of animations are less confident of the knowledge they acquire from animated data than from static data) [15].

A system implementation overview is shown in Figure 4. The system contains both monthly tasks and daily tasks. The crawling is done using a custom-made Java program and the last.fm API. We use modularity-based clustering [19] and `neato` in Graphviz [14] for the MDS-based embedding. To generate animations, we use `ImageMagick` (`http://www.imagemagick.org/`).

Monthly tasks: creating and updating a canonical map that contains position information of 18,000 artists. Currently, crawling starts with the top-10 artists from the top-50 popular tags on last.fm, and recursively collects information about artists that are similar to them, in a breadth-first fashion. The result is stored in the DOT for-

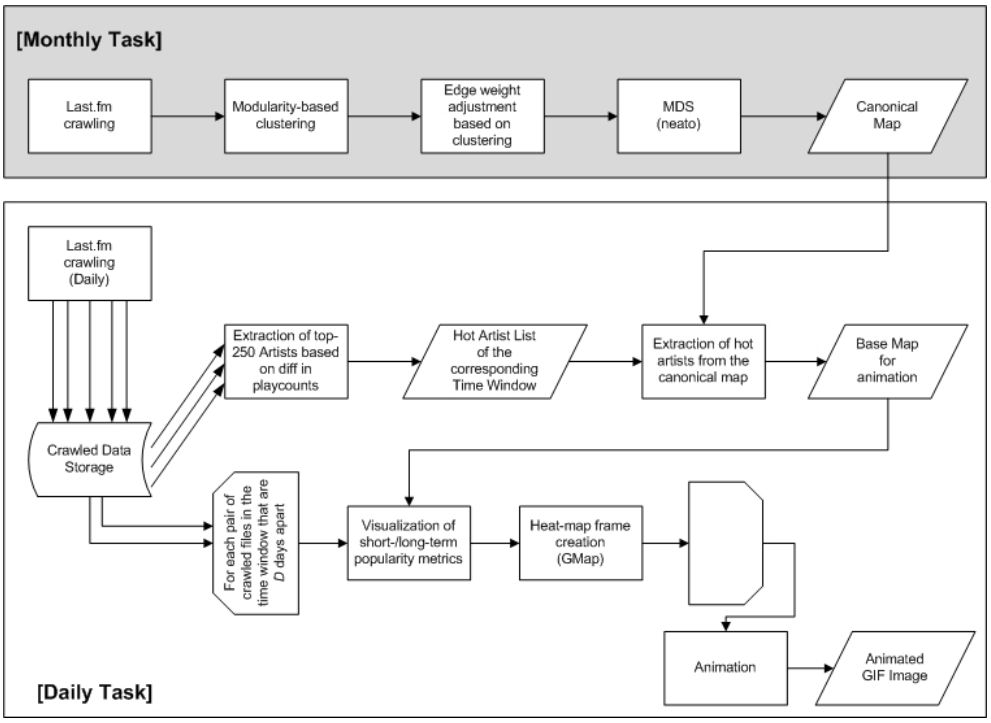


Figure 4: Overview of Implementation

(a) (b) (c)

Figure 5: (a) Highlighting in blue areas where artists are about to disappear: Bon Jovi, Deep Purple, Elvis, Simon & Garfunkel, CCR, and Eric Clapton. (b) Highlighting in yellow the areas where new artists are about to appear. (c) An image after the appearing/increasing phase showing newcomers Bruce Springsteen, Neil Young, The Kinks, and The Beach Boys.

mat used by Graphviz, with edge weights defined by the "similarity" values provided by last.fm. This relational data set is fed into the clustering module. Based on this clustering result, we adjust edge lengths as follows in order to reinforce the edges connecting nodes in the same cluster.

1. The length of intra-cluster edges is set to 1.
2. The length of inter-cluster edges is set to a constant $L > 1$ (currently $L = 75$ is used based on the results of our preliminary trials).

Following this step, the graph with adjusted edge lengths is passed on to `neato`, which then computes the node positions for the 18,000 artists. The output from `neato` is used for vertex placement in the canonical map.

Daily tasks: crawling, base map creation, and animated heat-map creation. Crawling is done independently and in much the same way as in the monthly task. The results of daily crawls are kept as separate DOT files. It should be possible to make the size of the daily crawl much smaller than the monthly one without missing new and important artists, but we have not explored this as the crawling of 18,000 artists usually completes in under 12 hours using one PC.

Base map: selecting daily-crawl results that are in the given time window (one of the configurable parameters in the current implementation) and, for each pair of daily-crawl results with timestamps that are D days apart, computing the differences in playcounts of all the artists. D can also be arbitrarily adjusted based on a characteristic of the target data set or a preferred degree of "sensitivity to changes". Based on these differences in playcounts, the top-250 artists are extracted for each pair of crawled data files, which are D days apart, in the time window. The superset of these artists is saved as a list of "hot" artists. Note that this list could contain more than 250 artists. Since the total number was usually less than 300, the

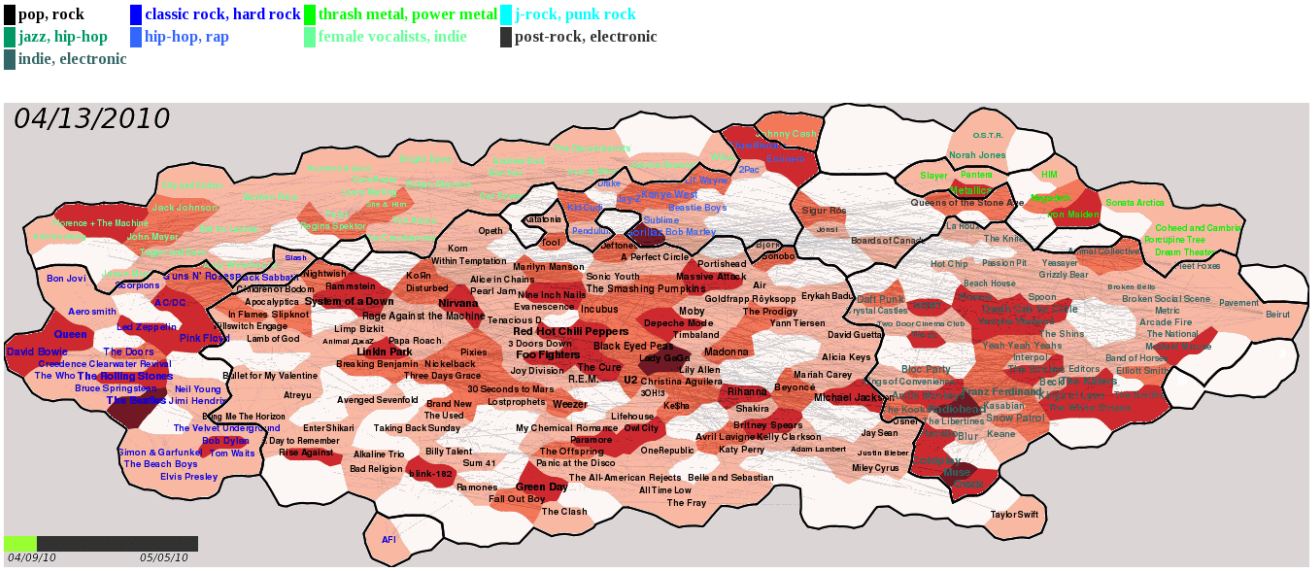


Figure 6: A snapshot of trend visualization of last.fm. On the heat-map, darkness of the color indicates the degree of increase in short-term popularity while font sizes of artists correspond to long-term popularity. Labels of artists who are out of the top 250 are hidden and colored white. Clusters based on similarity are represented with black boundary lines as well as label colors, which are mapped with country names shown on the top of the map. The date label located on the top-left corner indicates the timestamp of the heat-map displayed, and the progress bar on the bottom-left shows the position of the map in the entire animation.

base map remains readable and we include all of them. (As shown later in an example, artists that are not in the top 250 in each frame are hidden.) The position information for nodes in the base map is extracted from the canonical map. As discussed in Section 3.4, we also apply a label overlap-removal step here.

Metrics Visualization: creating an animation that visualizes the changes. Each heat-map frame to be included in an animation is created by modifying font sizes of the base map as well as categorizing artists in the base map into heat-map clusters. While the latter is done based on the magnitude of difference in playcounts, we use the cumulative number of listeners for each artist to determine the font sizes with formula (1). Our implementation uses $0.5 \times BaseSize$ as $Variation$. This allows both the overall popularity and the temporal ups and downs of each artist to be visualized in the map. We also need to establish groups of artists that have similar degree of change in order to draw a heat-map. There are a couple of ways to do this. For example, we can use playcount differences to classify artists, perhaps with suitable log scaling, and map the scaled differences to a color palette. Alternatively, we can utilize the ranking of artists in terms of the degree of change in playcounts to bin artists, and map bin indices to a color palette. We choose the latter option and use a single-hue color scheme so that artists with larger increase in popularity are assigned darker red colors.

Countries: when using heat-map overlay, each country can no longer be colored with a uniform color. Even though country boundaries help define the countries, additional visual cues are needed. Otherwise, viewers could have difficulty in identifying similarity relationships when countries are fragmented. We use the original clustering information based on similarity to define a label color for each artist so that artists in the same country have the same color. Currently we simply select a label color from a static palette, but this can be improved, for example by using a maximal differential color scheme [16], which is part of our future work. Within a country, different shades of the background color indicate variations in popularity. Each country is also automatically "named"; these names are created by taking the top two most frequent tags assigned to artists in the corresponding country. A list of country names written with the color associated with that country is presented above the animation; see Figure 6. In this way we obtain a labeled heat-map image, or the "base heat-map", and this process is repeated for each pair of daily-crawl results in the target time window that are D days apart.

Attention, (Dis)appearance, Complexity, and Confidence: The base heat-map images are concatenated in chronological order to generate a single animated GIF file. Note that we can arbitrarily change the number of frames in an animation by adjusting the size of the time window accordingly. However, naive concatenation of image files would create an animation that is difficult for a viewer to follow because there are too many changes happening at the same time: artists disappearing/appearing in the top 250, in addition to heat-map color changes representing artists remaining in the top 250 but whose popularity has changed. We break down these changes in a few intermediate steps. To emphasize appearance and disappearance of artists from one frame to another, we create intermediate frames for each pair of base heat-maps as follows.

1. A frame that highlights in blue all disappearing artists; see Figure 5(a).
2. A frame that hides all disappearing artists and updates heat-map colors for artists decreasing their popularity.
3. A frame that highlights in yellow all appearing artists; see Figure 5(b).
4. A frame that shows the artists joining the top 250 and updates heat-map colors for artists increasing their popularity, which is also the next base heat-map; see Figure 5(c).

Thus frames 1 and 2 correspond to a "disappearing/decreasing" phase and frame 3 and 4 establish an "appearing/increasing" phase. To help viewers understand which part of the animation they are watching, we also include a date label and a progress bar.

A snapshot of our last.fm visualization, which incorporates all the components discussed so far, is shown in Figure 6. Figure 7

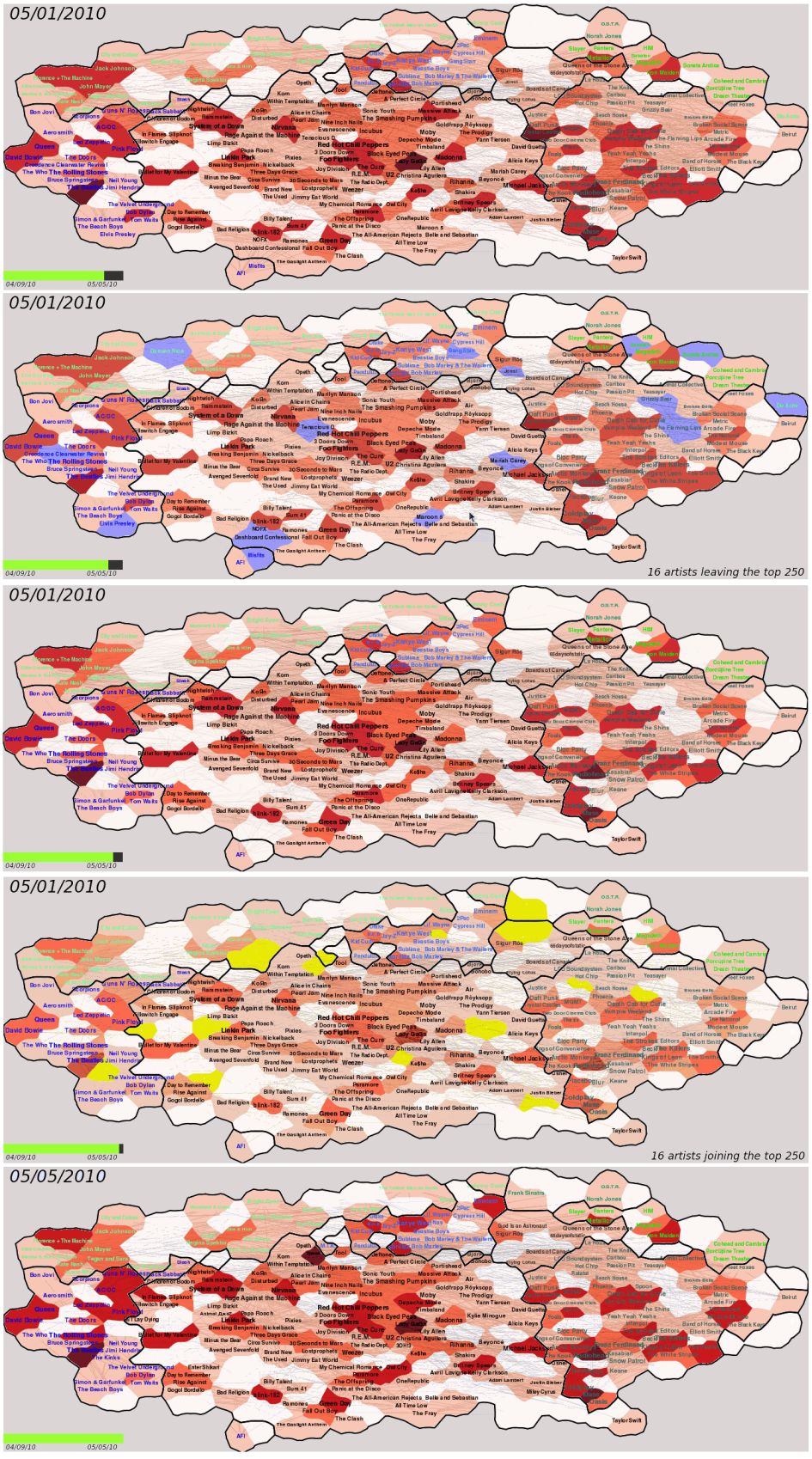


Figure 7: A sequence of animation frames between two consecutive base heat-maps including blue and yellow highlights.

shows a sequence of animation frames between two consecutive base heat-maps (for 01/05/2010 and 05/05/2010). An animated version is also available online at http://www2.research.att.com/~yifanhu/TrendMap/. The current implementation of our system accepts the entire duration of the animation and the number of heat-map frames as configurable parameters, and the interval between frames (4 days in this example) is determined from this information. As can be seen in this example, ups and downs in short-term popularity can be easily recognized by comparing darkness of colors. Artists joining or dropping off from the top 250 are highlighted, and artists out of the top 250 are hidden. In addition, changes between frames are two-phased which helps viewers identify and keep track of differences. Detailed evaluation of the "complexity" of these maps requires a user study. However, we believe that the metrics are reasonably encoded in familiar map components and that the geographic map metaphor helps viewers intuitively understand. For example, short-term changes in playcounts are encoded with color changes, while long-term popularity changes are encoded with label font sizes. The combination of these long-term and short-term metrics allows the viewer to derive additional information, such as spotting a rising star that suddenly draws public attention. Country boundaries and label colors help the viewer understand the relationships between similar artists at a glance. More detailed similarity information is also conveyed by semi-transparent edges, which come from the underlying graph.

One of the best ways to increase viewer "confidence", when looking at animated maps, is to give the viewer the ability to pause, rewind, and replay the animation. As this is difficult to achieve with a GIF animation, we also provide movies which offer better control (http://www2.research.att.com/~yifanhu/TrendMap/).

After a brief review of the animation used to illustrate this paper, it is easy to identify some trends and patterns. Several artists, such as the Beatles, Lady Gaga, and Radiohead, are popular throughout the time period. Others, such as Michael Jackson, fluctuate in popularity but remain in the top 250. Still others, such as Adam Lambert from American Idol go in and out of the top 250. We can also spot some events in music industry. For example, the release of the new album "Fever" by Bullet For My Valentine on April 27 is accompanied by a characteristic color change: the band was pink through much of April, but suddenly surged to dark red on April 27 and remained so thereafter.

5 Conclusions and Future Work

In this paper we explored a way to visualize large-scale dynamic relational data with the help of the geographic map metaphor. We addressed some challenges created by the dynamics in the data and presented a system that visualizes the user traffic on the Internet radio station last.fm with a heat-map animation. We believe the applicability of our approach is not limited to the last.fm data. For example, our scheme can be used, with minor modification in the data collection module, to visualize trends in the popularity of web sites, TV shows, etc., where similarity and popularity information are easy to define.

The major component of our future work is the evaluation of the effectiveness of our visualization through the user study. This would also include the calibration of parameters (duration of animation, interval for difference calculation, etc.). We are also working on implementing a functional interactive interface. As the underlying data is a map, we are exploring pan-and-zoom Google Maps-like interactions. The resulting system can be further enhanced by allowing access to external online contents (e.g., accessing the last.fm artist web pages, or a wikipedia page) by clicking on a node label. Although our prototype uses animated GIF, exploring other file-size efficient ways of creating the animation is part of our future work. Finally, offering several metrics for visualization would result in a more powerful system; for instance, our system uses a difference in the number of times each artist's songs are played (playcounts), while the second-order difference in playcounts will allow for more precise view of the momentum of an artist.

References

[1] Reconstructing the structure of the world-wide music scene with last.fm. http://sixdegrees.hu/last.fm/index.html.

[2] 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.

[3] U. Brandes and D. Wagner. A Bayesian paradigm for dynamic graph layout. In *5th Symp. on Graph Drawing (GD)*, pages 236–247, 1998.

[4] J. Branke. Dynamic graph drawing. *Drawing graphs*, 2025:228–246, 2001.

[5] H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *ACM SoftVis'06*, pages 105–114, 2006.

[6] C. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *ACM SoftVis'03*, pages 77–86, 2003.

[7] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE TVCG*, 15(6):1009–1016, 2009.

[8] S. Diehl and C. Görg. Graphs, they are changing. In *10th Symp. on Graph Drawing (GD)*, pages 23–30, 2002.

[9] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph animations with evolving layouts. In *11th Symp. on Graph Drawing (GD)*, pages 98–110, 2003.

[10] T. Fruchterman and E. Reingold. Graph drawing by force directed placement. *Software-Practice and Experience*, 21:1129–1164, 1991.

[11] E. R. Gansner and Y. F. Hu. Efficient node overlap removal using a proximity stress model. In *16th Symp. on Graph Drawing (GD)*, volume 5417, pages 206–217, 2008.

[12] E. R. Gansner, Y. F. Hu, S. Kobourov, and C. Volinsky. Putting recommendations on the map: visualizing clusters and relations. In *3rd ACM Conf. on Recommender Systems (RecSys)*, pages 345–348, 2009.

[13] E. R. Gansner, Y. F. Hu, and S. G. Kobourov. GMap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symp. (PacVis)*, pages 201–208, 2010.

[14] E. R. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice & Experience*, 30:1203–1233, 2000.

[15] M. Harrower. Tips for designing effective animated maps. *Cartographic Perspectives*, 44:63–65, 2003.

[16] Y. F. Hu, S. Kobourov, and S. Veeramoni. On maximum differential graph coloring. In *18th Symp. on Graph Drawing (GD)*, 2010.

[17] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Press, 1978.

[18] M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103:8577–8582, 2006.

[19] A. Noack. Energy-based clustering of graphs with nonuniform degrees. In *13th Symp. on Graph Drawing (GD)*, pages 309–320, 2005.

[20] E. Pampalk. Islands of music - analysis, organization, and visualization of music archives. *Journal of the Austrian Society for Artificial Intelligence*, 22(4):20–23, 2003.

[21] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1325–1332, 2008.

[22] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28:967–974, 2009.

[23] A. Skupin and S. I. Fabrikant. Spatialization methods: a cartographic research agenda for non-geographic information visualization. *Cartography and Geographic Information Science*, 30:95–119, 2003.

[24] J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow. Visualizing the non-visual: spatial analysis and interaction with information from text documents. In *IEEE Symp. on Information Visualization*, pages 51–58, 1995.