Visualizing Dynamic Data with Maps

(Submission Type: Research Paper, Long)

Daisuke Mashima[1], Stephen G. Kobourov[2], and Yifan Hu[3]

[1] College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
mashima@cc.gatech.edu
[2] Department of Computer Science, University of Arizona, Tucson, AZ, USA
kobourov@cs.arizona.edu
[3] AT&T Labs Research, Florham Park, NJ, USA
yifanhu@research.att.com

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

1 Introduction

The use of a geographic map metaphor for visualizing relational data was originally described in the context of visualizing recommendations, where the underlying data is TV shows and the similarity between them [13]. 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 [12]. 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 are getting more (or less popular) over that period, in addition to displaying relational information among.

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 popular Internet radio station last.fm (`http://www.last.fm/`).

1.1 Related Work

The problem of drawing dynamic graphs is well studied; see the survey paper by Branke [4]. In dynamic graph drawing the goal is to maintain a nice layout of a graph that is modified via operations such as insert/delete edges and insert/delete vertices. Techniques based on static layouts have been used [16]. North [23] studies the incremental graph drawing problem in the DynaDAG system. 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 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. In [1] a graph-based representation is used, 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 [24] uses self-organizing maps (SOM) [17], an unsupervised learning algorithm that places objects on a two-dimensional grid so that similar objects are close to each other. Unlike our approach, in SOM the cells of the grid are colored based on a feature value; therefore it operates on a discrete grid space without a clear inner boundary between "countries". Furthermore, the grid tends to fit a rectangular box, so that the overall outline of the point set often follows that shape. In addition SOM has a high computational complexity and is not scalable.

Using maps to visualize non-cartographic data has been considered in the context of spatialization [26]. Map-like visualization using layers and terrains to represent text document corpora dates back to 1995 [27]. 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 [25]. 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.

2 Creating Maps from Graph Data

We begin with a summary of the GMap algorithm for generating maps from static graphs [12]. 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

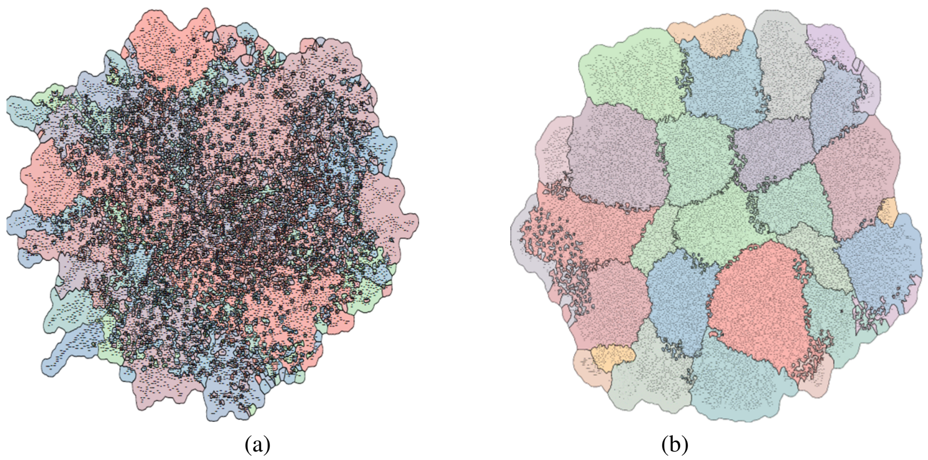


Fig. 1. (a) Map of 18,000 artists (b) Map of 18,000 artists using adjusted edge lengths

(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. The weight of an edge can be defined as similarity relationship 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) [18]. In the second step, a cluster analysis is performed in order to group vertices into clusters. GMap uses modularity-based clustering algorithm [20] or a geometric clustering algorithm such as k-means [19]. It is desirable that the embedding and clustering algorithms be related, which leads to vertices in the same cluster being geometrically close to each other in the embedding. For example, force-directed embedding algorithms rely on similar principles as modularity-based clustering, as pointed out by Noack *et al.* [22]. 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 so that no two adjacent countries have the same or similar shades. Further geographic components are 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 mountain ranges can be added, in order to emphasize the separation.

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. 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. Changes are visualized by animation, which can be generated by concatenating static maps, thus providing continuity from one layout to the next. Extending the map visualization to large-scale dynamic data poses additional challenges. 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 can be confusing and counter-intuitive. Animated cartographic maps pose four additional challenges: 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].

In Section 3 and 4, we describe how we addressed these challenges with the goal of visualizing the music trends of the Internet radio station last.fm.

3.1 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 precomputed position information from the canonical map. 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.

3.2 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 exposed a problem with this approach. We used modularity-based clustering 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 Fig. 1 (a). The most conspicuous problem is the fragmentation of countries into disjoint regions. We found that, on average, one cluster 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 LinLog layout [21] 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 the different clusters, leading to a much better canonical map; see Fig. 1 (b). In this map, fragmentation is significantly reduced although there are irregularities near some country boundaries.

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 vertices with overlapping labels in their neighborhood to resolve overlaps, 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.

3.3 Labels and Overlap Removal

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 250 artists, some artists are much more popular than others and this difference needs to be included in our visualization. One straightforward way is to represent the popularity by varying the font size of the labels, just like geographic maps, where the names of major cities are drawn with large fonts and smaller towns with smaller fonts. To modify the font sizes, we use the following conversion for each artist displayed.

$$ModFontSize_a = BaseSize + Variation * f(pop_a) \tag{1}$$

where

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

Here, the set A denotes that of 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

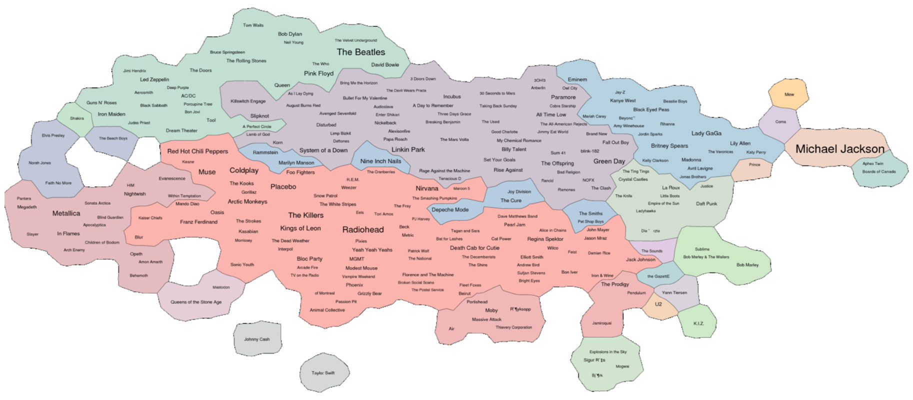


Fig. 2. Sample visualization of top-250 artists

(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 Fig. 2. This visualization shows us which artists attracted people through the font sizes while effectively showing the relations among artists as a map.

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 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. First, we create a list of artists that appear on any of the map frames to be included in the animation. Second, the system extracts the position information of these artists from the canonical map. Third, we set the font sizes of all labels on a map to the average size, i.e. $BaseSize$ in formula (1). After that, we apply an overlap removal step on the resulting map. We call this map a "base map" because it is used to create each map frame in the animation. In other words, the font resizing is applied to this base map to generate each map frame. 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, because all 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. 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 detailed a procedure for creating them in Section 4.

3.4 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. 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 of viewers. 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.

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. An overview of our implementation is shown in Fig. 3. As can be seen, the system consists of tasks conducted on a monthly basis and tasks performed daily. The crawling is done using a custom-made Java program and the last.fm API. In addition, we use modularity-based clustering [21] and `neato` in Graphviz [14] for the embedding process using MDS. To generate GIF animation, we utilize the `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 of artists that are similar to them, in a breadth-first order. The result is stored in DOT format used by Graphviz,

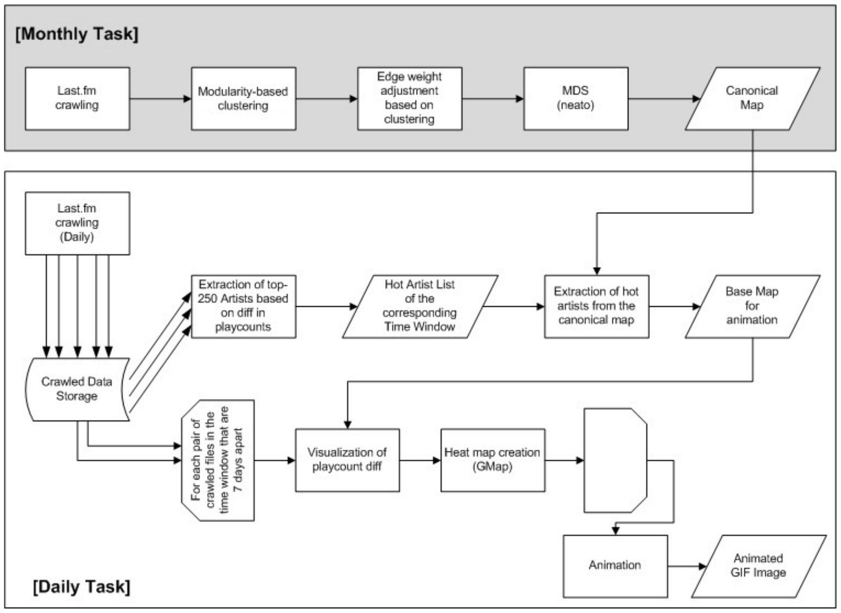

[Monthly Task]
Last.fm crawling
Modularity-based clustering
Edge weight adjustment based on clustering
MDS (neato)
Canonical Map
Last.fm crawling (Daily)
Crawled Data Storage
Extraction of top-250 Artists based on diff in playcounts
Hot Artist List of the corresponding Time Window
Extraction of hot artists from the canonical map
Base Map for animation
For each pair of crawled files in the time window that are 7 days apart
Visualization of playcount diff
Heat map creation (GMap)
Animation
Animated GIF Image
[Daily Task]

Fig. 3. Overview of Implementation

with “similarity” values provided by last.fm used as edge weights. 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 $L = 75$).

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 the canonical map.

Daily tasks: crawling, base map creation, and animated heat-map creation. Among them, crawling is done independently and is done in the same way as in the monthly task. The results of daily crawls are kept in a storage on our server as separate DOT files. It may 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 only 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 computing the differences in playcounts of all the artists over the given time window with timestamps that are *D* days apart. *D* can also be arbitrarily adjusted based on a characteristic of the target dataset 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. The superset of these artists is saved as a list of “hot” artists. Note that this list could contain more than 250 artists. However, since the total number was usually less than 300 and the resulting readability was good enough for a normal computer screen, we decided to include all of them in a base map. (As shown later in an example, artists not in the top 250 in each frame are hidden.) After that, by extracting position data of the hot artists from the canonical map, our system creates a base map. As discussed in Section 3.3, we apply overlap removal in this step as well.

Animation: the next step is to create 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 artists to determine the font sizes with formula (1). This allows both the overall popularity and the local ups and downs of each artist to be visualized in the map.

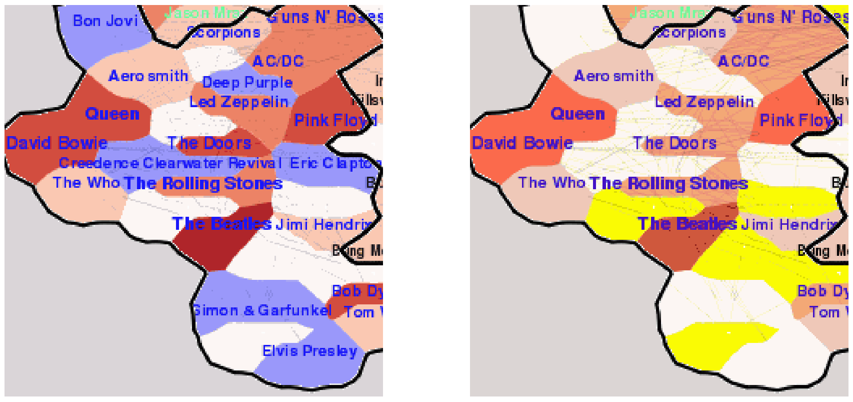
Heat-maps: To draw a heat-map, we need to establish groups of artists that have similar degree of changes. There are a couple of ways to do this. For example, we can use playcount differences to classify artists, perhaps with suitable log scaling, and map the scaled differences to a color palette. Alternatively, 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. We 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 work to some extent, additional visual cues are needed to help define the countries. We use the original clustering information based on similarity to define label colors of each artists. Artists in the same country have the same label color. Furthermore, each country is automatically "named"; these names are created by taking the top two most frequent tags assigned to artists in each country. The tagging information is acquired from last.fm API and included in crawled data. This way we obtain a labaled heat-map image (`base heat-map`). This is repeated for each pair of daily-crawl results that are *D* days apart.

Attention, Disapperance, Complexity: The heat-map images are concatenated 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 from the top 250, in addition to heat-map color changes representing artists remaining in the top 250 but with popularity 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 all disappearing artists with blue color; see Fig. 4(a).
2. A frame that hides all disappearing artists and updates heat-map colors for artists decreasing their popularity.
3. A frame that highlights all appearing artists with yellow color; see Fig. 4(b).

The frame 1 and 2 correspond to "disappearing/decreasing" phase and the frame 3 and the next base heat-map establish "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) (b)

Fig. 4. (a) Highlighting disappearing artists (b) Highlighting appearing artists

Static images given by our system are shown in Fig. 5 and in the Appendix, an animated version is available online at `http://lastfmmap.web.fc2.com/`. 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 (Fig. 4), and artists out of the top 250 are hidden. In addition, changes between frames are two-phased and are not jumpy, which helps viewers identify and keep track of differences. Detailed evaluation about the "complexity" of these maps requires a user study, which is planned for our future work. Besides changes in playcounts, viewers can grasp long-term popularity through label font sizes. Country boundaries and label colors help viewers understand similarity relationship at a glance. More detailed similarity information is also conveyed by semi-transparent edges, which come from an underlying graph data. Regarding "confidence," we provide a movie at the website which allows a replay of the animation, this offers better control over the animation (pause, forward, backward, etc.).

From the animated visualization, we can see some important trends: a number of artists are always very popular, such as Beatles (southern part of the Classic Rock/Hard Rock country), Lady Gaga (middle of the Pop/Rock country) and Radiohead (south west of the Indie/Electronic country). On the other hand, some artists, such as Adam Lambert from the American Idols (south east of the Classic Rock/Hard Rock country) goes in and out of the top 250. We can also see that event in music industry, for example, Bullet For My Valentine (western border of the Pop/Rock country) released a new album "Fever" on April 27 and we can clearly see that the color for the band, which stayed pink up through out much of April, suddenly surged to dark red on April 27 and remained so.

Finally, our system takes 3-4 hours to create a canonical map from a crawled data of 18,000 artists. This is mainly due to the embedding step using MDS. However, this process is done infrequently (monthly in our implementation) and is still short enough to allow for daily updates. The daily tasks, including creation of a base map and a heat-map animation, require less than 15 minutes for the creation of a 1-month animation.

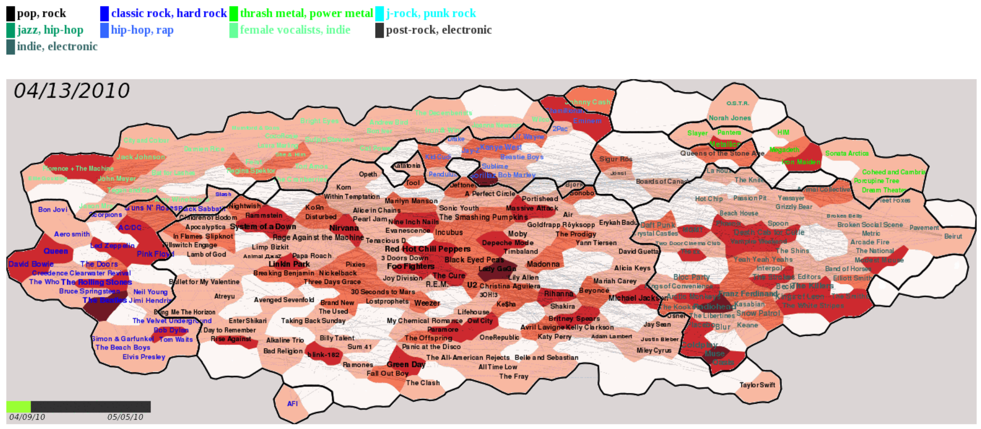


Fig. 5. A snapshot of trend visualization of last.fm

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 in 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 websites, 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 include finding the optimal selection of parameters (duration of animation, interval for difference calculation, etc.). We are also considering to implement an interactive interface. Google-Maps like interface that allow viewers to zoom in and out by means of intuitive mouse operation would be effective. Finally, offering several metrics for visualization would provide more power to the system. For instance, while our system uses a difference in the number of times each artist's songs are played (playcounts), utilizing the second-order difference in playcounts (difference in differences in playcounts) will enable us to further visualize the degree of each artist's momentum.

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 Symp. on Information Visualization (INFOVIS '02)*, pages 145–151, 2002.
3. U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In *Proceedings of the 5th Symposium on Graph Drawing (GD)*, pages 236–247, 1998.
4. J. Branke. Dynamic graph drawing. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, pages 228–246, 1999.

5. H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *SoftVis'06: Proceedings of the 2006 ACM symposium on Software visualization*, 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 Symp. on Software Visualization (SoftVis)*, pages 77–86, 2003.
7. C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.
8. S. Diehl and C. Görg. Graphs, they are changing. In *Proc. of the 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*, pages 98–110, 2003.
10. T. M. J. Fruchterman and E. M. 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 Symposium on Graph Drawing*, pages 206–217, 2008.
12. E. R. Gansner, Y. F. Hu, and S. G. Kobourov. GMap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symposium (PacVis)*, pages 201–208, 2010.
13. E. R. Gansner, Y. F. Hu, S. G. Kobourov, and C. Volinsky. Putting recommendations on the map: visualizing clusters and relations. In *3rd ACM Conference on Recommender Systems (RecSys)*, pages 345–348, 2009.
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. 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.
17. T. Kohonen. *Self-Organizing Maps*. Springer, 2000.
18. J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Press, 1978.
19. S. Lloyd. Last square quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.
20. M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103:8577–8582, 2006.
21. A. Noack. Energy-based clustering of graphs with nonuniform degrees. In *Proceedings of the 13th International Symposium on Graph Drawing (GD 2005*, pages 309–320, 2005.
22. A. Noack. Modularity clustering is force-directed layout. *Physical Review E*, 79, 2009.
23. S. C. North. Incremental layout in DynaDAG. In *Proc. of the 4th Symp. on Graph Drawing (GD)*, pages 409–418, 1996.
24. 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.
25. P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28:967–974, 2009.
26. 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.
27. 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 Symposium on Information Visualization*, pages 51–58, 1995.

Appendix

A sequence of static images from May 1 to May 5, 2010.

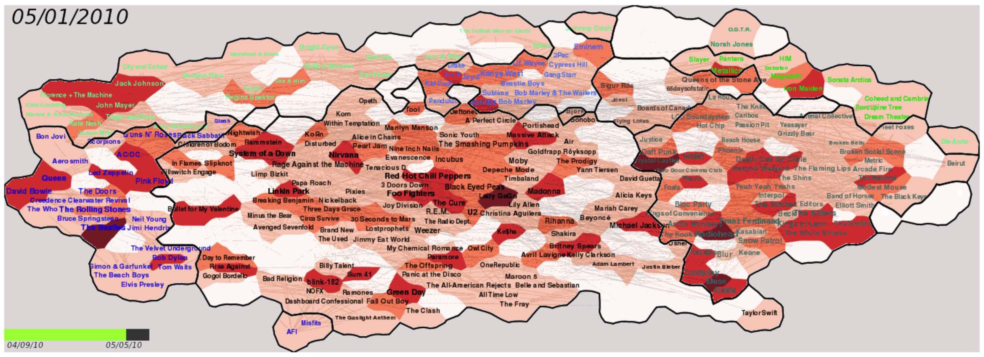


Fig. 6. Base heat map of 05/01/2010

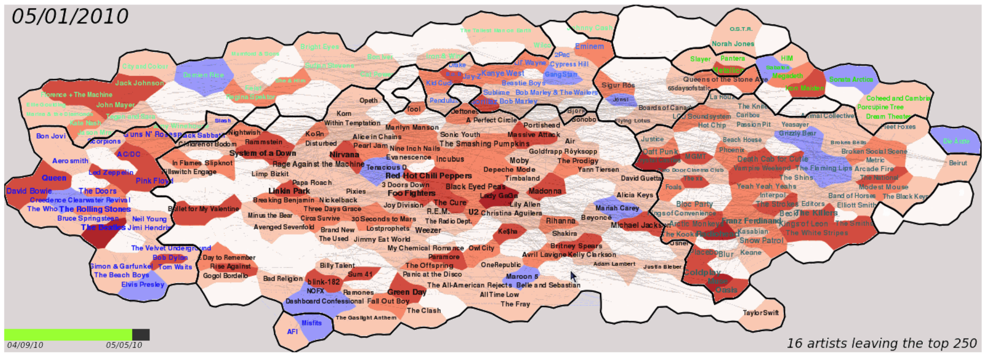


Fig. 7. Highlighting disappearing artists

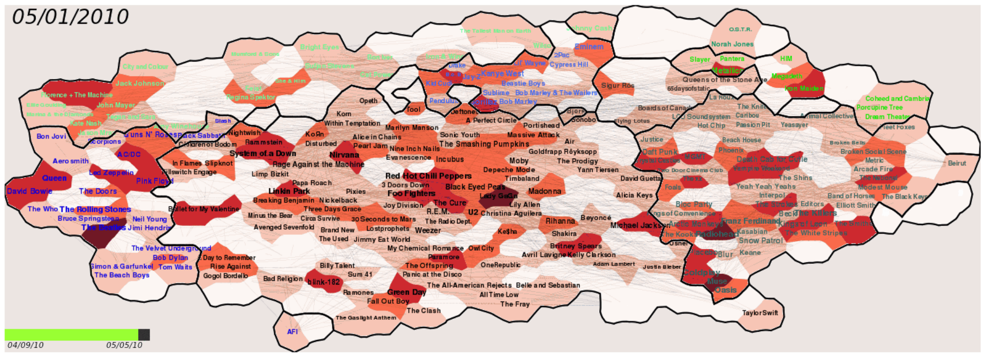


Fig. 8. A snapshot after disappearing/decreasing phase

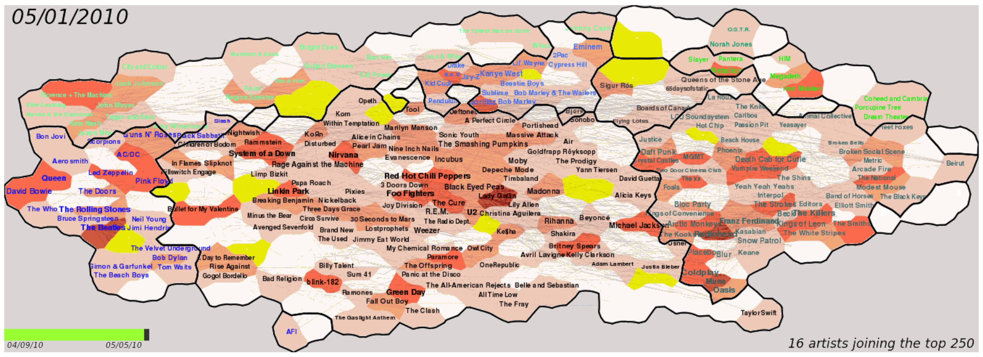


Fig. 9. Highlighting appearing artists

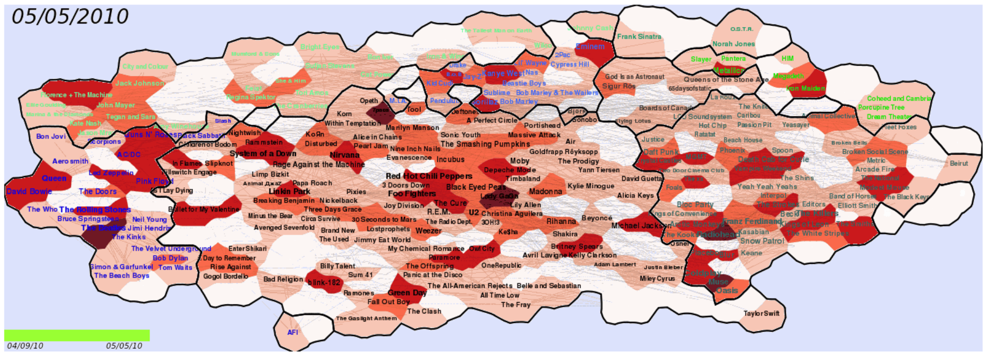


Fig. 10. Base heat map of 05/05/2010