Jason Woods, Justin Lee
Interactive Visualization
Barbara Cutler
05/05/2016

Introduction

For music lovers, finding new music can be tough, especially if they are only interested in very specific genres. Many music streaming services offer personalized suggestions based on a user's library, but these are usually based solely on other listeners' music history. A different way of showing connectedness between bands and how those bands are related would be to show connections between them based on performances: for example, bands often accompany each other on tours, and they may gather into festivals for certain musical styles.

Originally, we hoped that these performance connections would reveal information on unknown, or hard to define, subgenres, thus leading us to ask the following question: Can the performance relationships between bands reveal information about the genre they are grouped into?

However, as the project progressed, we could not acquire the appropriate data necessary to answer this question. As a result, we switched to an approach that would hopefully uncover music artists that typically tour with the artists that the user of the visualization would input. The question that we were hoping to answer with this new approach was the following: Would a graph of music artists connecting related artists based on performance history to a specific artist reveal new or unforeseen connections between certain bands or artists that the user could use to discover new music or make decisions on which shows to attend?

The project can provide answers to other questions as well; for example, for a more casual audience, will we find that people enjoy a band if it is adjacent to another band that they already know? Will other correlations emerge from band connectedness? And, finally, do bands across genres, as known by the user, have performance correlations?

The audience for the visualization consists of people who are interested in music and take curiosity on how categories can objectively be drawn; however, this would also draw in those who would like to find other artists they might want to listen to or see at a show, based upon a certain artists they connect to.

Before we began the project, we had produced a hypothesis for the data, since we intended to come into the experiment without any notions about the data beforehand.

Null Hypothesis - Band connections will not reveal anything meaningful about music genres, or anything concerning listeners' habits.
Hypothesis - Band connections will reveal subgenres in music, and therefore how people tend toward certain styles of music.

Once we pivoted, we came up with a new hypothesis to test, so as to guide our visualization and test if we actually accomplished what we set out to do.

Null Hypothesis - Performance connections will not reveal anything meaningful about music artists and their relatability, thus providing no new information to the user.
Hypothesis - Performance connections will allow new relations to be seen between bands that were previously unknown, allowing users to find new music or make decisions on concerts they would or would not like to go to in the future.

Related Works and Data Sources

We took inspiration from one of the earlier papers we read this semester, called "Social Network Clustering and Visualization using Hierarchical Edge Bundles"; we wanted to create a graph wherein the nodes have multiple connections, so that it allowed analysis later of where isolated communities would be, which would represent subgenres. After changing our visualization though, the ideas in this paper became less important.

Another paper we read this semester that we took inspiration from was "Representing Uncertainty in Graph Edges: An Evaluation of Paired Visual Variables"; we wanted a graph where the visual variables and the interactions of such would communicate the edge strengths easily, which is why we added Carpendale's "Considering Visual Variables as a Basis for Information Visualization" to our cited paper's list, as the former paper referenced it.

"Graph visualization and navigation in information visualization: A survey," was used to inform us how to format our graph overall.

We expected to use a part of the “Finding and evaluating community structure in networks” paper to understand how to perceive communities in graphs visually, but since our visualization dramatically changed we could not use it.

The paper "The emergence of complex network patterns in music artist networks" covers the usage of music information networks and the visualizations of those. While much of the information was not directly applicable, it did serve as a sort of starting point though for what we wanted to achieve and gave us some ideas of what we did and did not want to do.

Our movement to D3 was facilitated by two pieces of code, both released under the GNU public license v3, and by the same author: Mike Bostock’s Block 2062045 (http://bl.ocks.org/mbostock/4062045) and Block 1062288 (https://bl.ocks.org/mbostock/1062288); the former was used to show a simple graph initially, and the second was to implement the collapsible nodes layout.

Credits to Songkick for providing all of the data for this project via their API. Their extensive database of past events for artists was invaluable and drove the progress of the visualization.

Visualization

The visualization is a graph of nodes connected by edges where each node represents a music artist and each edge represents a connection created by the two connected nodes performing together. Originally, the graph was created using GraphViz. However, as we progressed through the project, we switched over to D3.js for Javascript because of its flexibility, large number of additional features, and increased control over graph creation.

Initially, the graph was a network of all the connections to a single, user-inputted music artist, otherwise known as a source artist. We used color and length to denote edge strength. Certain colors were attributed to stronger connections and a longer edge meant a weaker connection. Using edge length was originally supposed to keep closely-related artists together, while separating artists that were less related. Node size also varied depending upon the strength of the connection to the source artist: the stronger the connection, the larger the node.

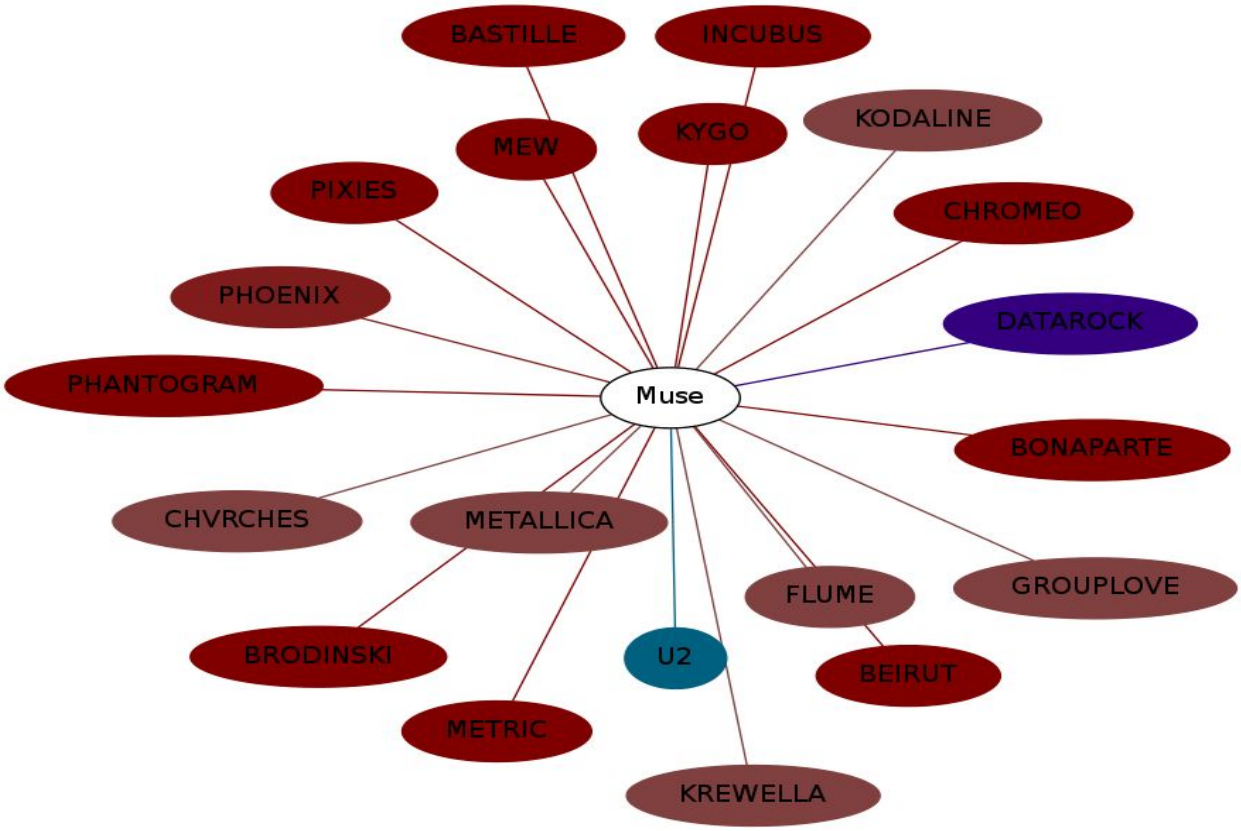

Fig 1. Initial GraphViz graph, using a combined text file created from the Songkick data and manually-selected data from Muse's fan website. The color of the branch nodes and the edges are identical and based on edge weight and type of edge weight (whether Muse was the supported band, the supporting band, or the other band had been with them in a festival, mapped to HSV format), and edge length was based on the strength of the edge.

As we progressed and took into consideration feedback from others, we decided to take a new approach and switch to D3 to handle the graph creation. We also changed the visual variables and added support for multiple source artists. This greatly changed the overall look of the graph, along with what information could be learned from it.

To start, we added support for multiple source artists to be specified. Instead of just showing the top related nodes to a single artist via edges, the graph can contain up to three source artists. Each node (whether source or non-source) is connected to a source node. This represents the performing relationship that the two music artists share. Non-source nodes are not connected to other non-source nodes, but can be connected to multiple source nodes. A connection to multiple source nodes would mean that that particular artist is in the top related artists list for both sources.

2a.

2b.

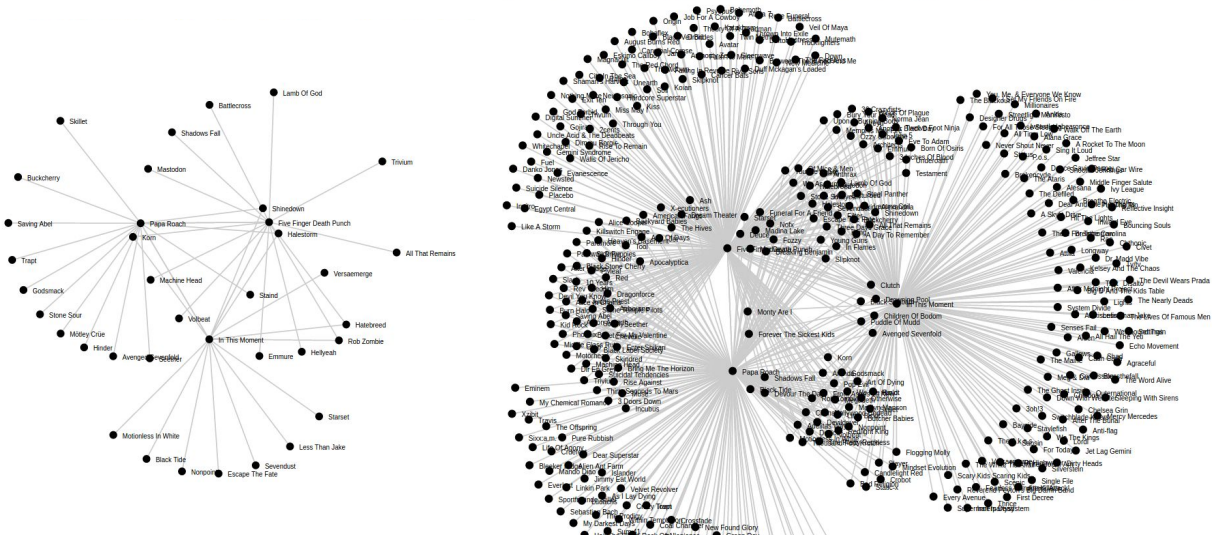

Fig 2a and 2b. First graphs created with D3. Visual variables are equal for every node and edge. The left graph (2a) has three source artists and plots the top ten related artists for each source. The right graph (2b) has the same three source artists but plots the top one hundred related artists.

After adding multiple source nodes, we decided to use node size to encode a new piece of information. The size of the node represents the corresponding artist's performance popularity. The larger the node, the more times that particular artist has been a headliner, or one of the top artists to play, at a musical event. Smaller nodes represent artists that are more commonly openers for more popular bands. This seemed like a natural way of representing performance popularity because the larger nodes would be more visible, thus gaining more attention than the smaller nodes.

Fig 3. Graph using the performance popularity variable to change node size. Larger nodes represent artists that tend to be headliners more than the smaller node artists.

Next, we changed the way that edge strength was represented. Edge strength is calculated by: $(concertsTotal * 30) + (festivalsTotal * 20)$. While using length initially seemed like a good idea, it inevitably led to very large graphs that had widespread nodes. This made it difficult to keep in perspective the entire visualization because one had to scroll around it to see the entire thing. Therefore, we switched to an opacity and width model. Basically, the stronger the connection, the wider and more opaque the edge became. This reduced the size of the visualization greatly and used variables that were more common and therefore easier to understand. The calculation for the width places the value from a range of 0 - 15: $\frac{(edgevalue - minEdge)}{(maxEdge - minEdge)} * 15$. The calculation for opacity is $(\ log(edgeValue)/10\ ) - 0.45$, the constant 0.45 value to ensure not every edge looks solid; more often than not the value is greater than 1 without the constant.

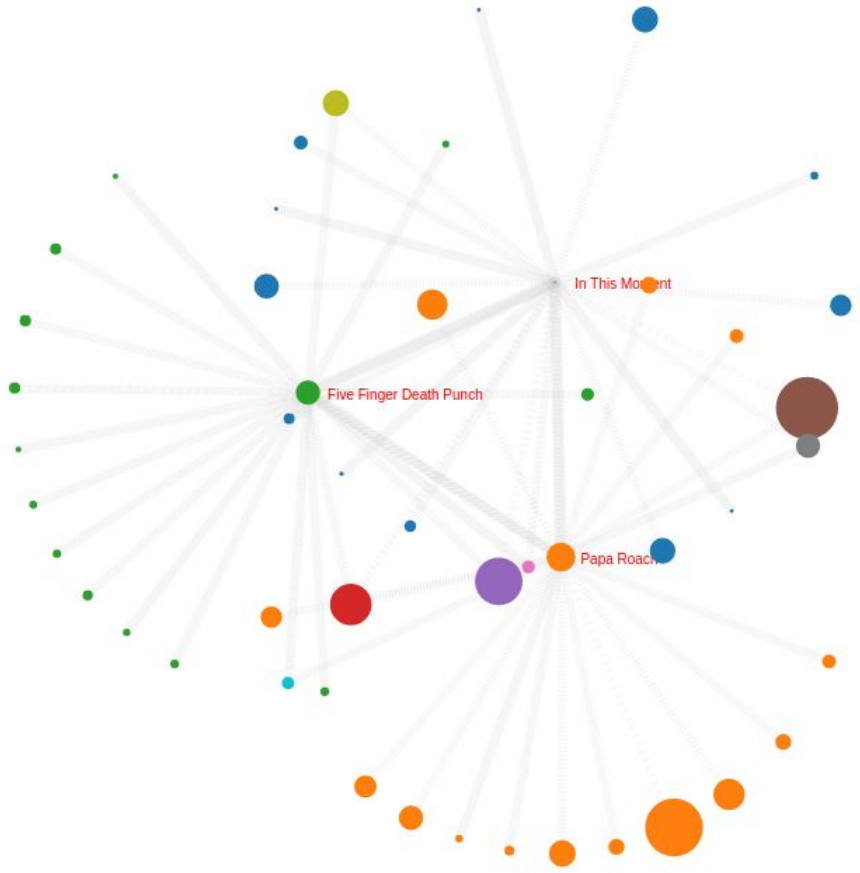

Fig 4. Graph showing the new method of encoding edge strength. Thicker and more opaque edges represent stronger connections between nodes than thin, transparent edges.

The last visual variable changed was color. We decided that color would be a good way to group nodes. This grouping is based on the source nodes that each node is connected to. Each source node receives its own unique group, and each non-source node is assigned a group based on every possible combination of connections with the original source nodes. For example, non-source nodes only connected to a single source node would be in the same group as the connected source node. However, non-source nodes connected to multiple source nodes would be placed in their own group calculated by the exact source nodes they are connected to. Group was then directly transformed into color, which helped to easily identify which nodes belonged to which source nodes.

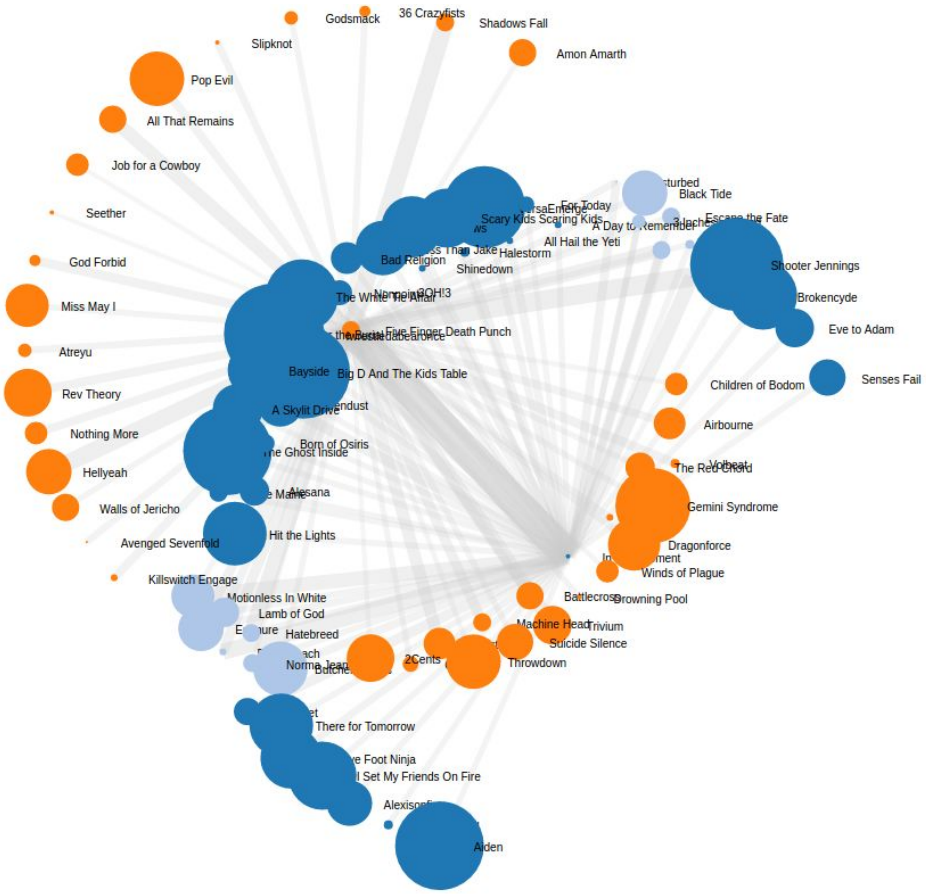

Fig 5. All of the orange nodes belong to a single source node and all of the dark blue nodes belong to another source node. The light blue nodes have connections to both source nodes and are therefore in their own group.

We also used this grouping to color the edges. Before, the edges tended to all just blend together on larger graphs. However, coloring the edges based on the color of the source node greatly aided in determining what source node each non-source node is actually connected to. It especially helps with non-source nodes that are connected to multiple source nodes, for one can clearly see the multiple edge colors to determine which sources the nodes are connected to.

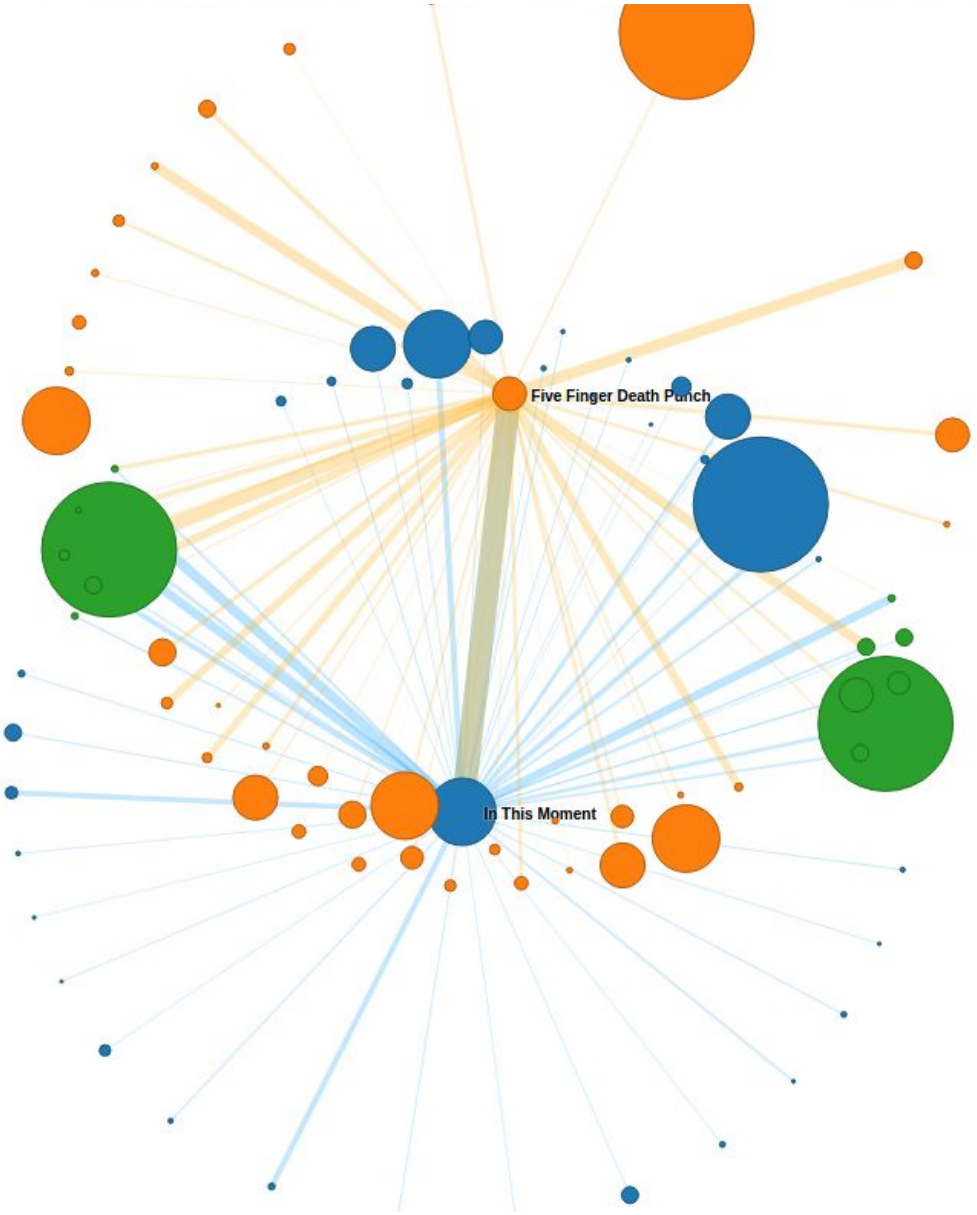

Fig 6. Edges are colored to depict which source node the edge is connected to. For this example, it is now easier to tell that the green nodes are connected to both the blue and orange source nodes. The color is blended together for source nodes that are connected to each other.

We also added interactions to the visualization to make it easier for the audience to navigate and understand the data. First, we decided that having the artist's name for each node displayed all the time made the graph very cluttered and obscured a lot of edges. To fix this, only the source nodes have ever-present artist names (which also was used to easily identify source nodes) and non-source nodes only display their artist name when hovered on. We also added the functionality to click and drag nodes around the graph. The graph is force-directed,

so it does a pretty nice job of separating nodes. However, sometimes nodes are overlapping or certain areas are too crowded. Being able to drag nodes out of the way fixes some of these problems. Source nodes are also collapsible within the graph. When a source node is clicked, every node of one degree connected to that particular source node disappears. Nodes connected to the collapsed source node and other source nodes are still visible though. This allows for a sort of filtering mechanism that ignores nodes with only one source connection and focuses on nodes with relationships to multiple sources.

Feedback

On April 26, 2016 a demo was held, where we demonstrated a version of our graph, made via GraphViz, where the edge lengths and colors were based upon the edge strength; node size was also based on edge strength. Generally people did not like these variables, and suggested an interaction of width and fuzziness to show edge strength.

We received a multitude of suggestions. One mentioned that node size could be dependent on whether the band the source is related to is a headliner; the more often the node is a headline, the larger the node is. This struck us as a good metric to record for node size, so we implemented this. Another mentioned that the color of the edges should be on a discrete scale, which we had decided not to implement since we had not received enough feedback that color was good for showing edge strength; the color issue was raised again when another person mentioned that people have different perceptions of color, and that saturation and hue were superior. Someone recommended using dotted or dashed lines for the edges to show edge weakness. Someone suggested coloring nodes based on genre, but Songkick does not include genre information, so this was not practical to implement.

We therefore underwent some strong changes based on this feedback. To begin with, we decided node size would be dependent on the number of times the artist was a headliner. We decided edge strength should be depicted by edge width, opacity, and number of dashed lines (more dashes, the weaker the edge is). However, we found that three separate visual variables to encode the same information was too much, so we eliminated the dash-lined variable. We also colored nodes and edges on a discrete color scheme determined by the source nodes.

It was also suggested that we move away from GraphViz and start using D3 instead to create more interactive and aesthetically pleasing graphs. We already planned on doing this, but it was encouraging to hear others suggesting it.

Technical Implementation

Initially, many data sets were considered for the visualization. We tried using the Spotify API to obtain information about artists, but found that the information provided was fairly sparse. There was no touring history provided and the most useful information that could be gathered was related artists based on listener history. However, this project was all about visualizing music artist connections based on performances, not listener history.

Wikipedia also proved to be largely unreliable. While most artists have pages, the layout and content of the pages varies greatly, and is near impossible to parse information from. We also had the problem of determining what words or sets of words on a page were band names and what were not, along with identifying tours and concerts that bands partook in.

The source that we found most fit for our target information and provided the most useful data was Songkick. All of the data for the visualization was pulled from the Songkick database via the Songkick API. This provided access to an artist's entire "gigography", or touring history. Each event that the artist participated in included information such as location, date, other bands that were there, the billing order, etc. The code to issue the API calls and parse the data was all written in Python.

For each source artist provided, an API call to Songkick was made to get each artist's gigography. From there, the data was parsed to find the top related artists to each source artist. While calculating relatedness between artists, we took event type into consideration. Playing at a concert together increased the relatedness score more than playing at a music festival together. This is because festivals can feature many bands that may not be related at all and never even really interact. Playing at a concert while on tour together, though, usually means that the artists have a bit more in common.

While acquiring this information, the billing index for each artist was recorded. This was used to determine the size of the node for each artist in the visualization. The lower the billing index, the more of a headliner that artist was for that particular event. The average of all the billing indices for each artist was recorded, and the artists with the lowest average billing index are represented as the largest nodes on the graph, to display their higher level of popularity.

Groups were used to link each artist on the graph to source artists. Since an artist may be related to multiple source artists, grouping is not as simple as just seeing what source artist it is related to and placing it in that group. In order to take this into an account, all of the possible combinations of source artists were assigned groups, and the links that each related artist had to each source artist was recorded and used to group it in one of these assigned groups. Due to the fact that the number of combinations of source artists would quickly grow to a large number, a max of three was used for the number of total source artists. This means that there are at most six different groups of artists, which is small enough to use color to encode the groups. If there were four source artists specified, the number of combinations would jump to 19, which is too many different groups for color to be effectively used to display the information.

After assembling all of this data, the Python script outputs it into a JSON file, which is passed to a Javascript file. The Javascript file uses D3 to handle the graphical part of the visualization. It creates nodes and edges between those nodes based on the input JSON file. The nodes are resized based on their value, which comes from the previously mentioned average billing index. Edges are also edited to reflect the strength of the relation between artists. Lastly, groups are used to assign color to each node. Source nodes show their text constantly, and branch nodes show their text only when the user's mouse is over them.

Forces were applied to nodes to help separate the graph and keep it spread out. Otherwise a lot of the nodes were overlapping and the structure was random. After adding forces, nodes tend to separate naturally into their groups, especially the ones only connected to a single source node. This greatly improved the usefulness and readability of the graph.

One final feature implemented for the graph, which proved useful since the thickness of edges sometimes obscured graphs, was that to collapse the branches of source nodes when the source nodes were clicked. This proved the most challenging aspect of the task, since it meant the colors had to be reassigned to every node after these branches were removed, and there had to be surety for what order D3 redrew the nodes and the edges afterwards.

Challenges

One of the most significant challenges of creating this visualization was wrangling all of the data. Songkick has an incredibly extensive gigography stored for each artist, resulting in thousands of events that have to be parsed for each source node that is specified. This made efficient algorithms very important to reduce the overall runtime of generating the JSON file for

the graph. Also, testing and debugging was tough because it was hard to make sense of the data sometimes. In order to verify results, we chose source bands in genres that we were very familiar with, and used that to check to make sure that the resulting visualization made sense.

Calculating the edge weight and opacity was another tough challenge, and still does not work consistently well across all graphs. Since we wanted these two properties to be relative to the present graph, we are using algorithms to determine an appropriate value for each of these. If we had more time, we could check over a large variety of artists' networks to see which algorithm would work best for a general purpose, however it is unknown if there is a one-size-fits-all algorithm for determining these two variables.

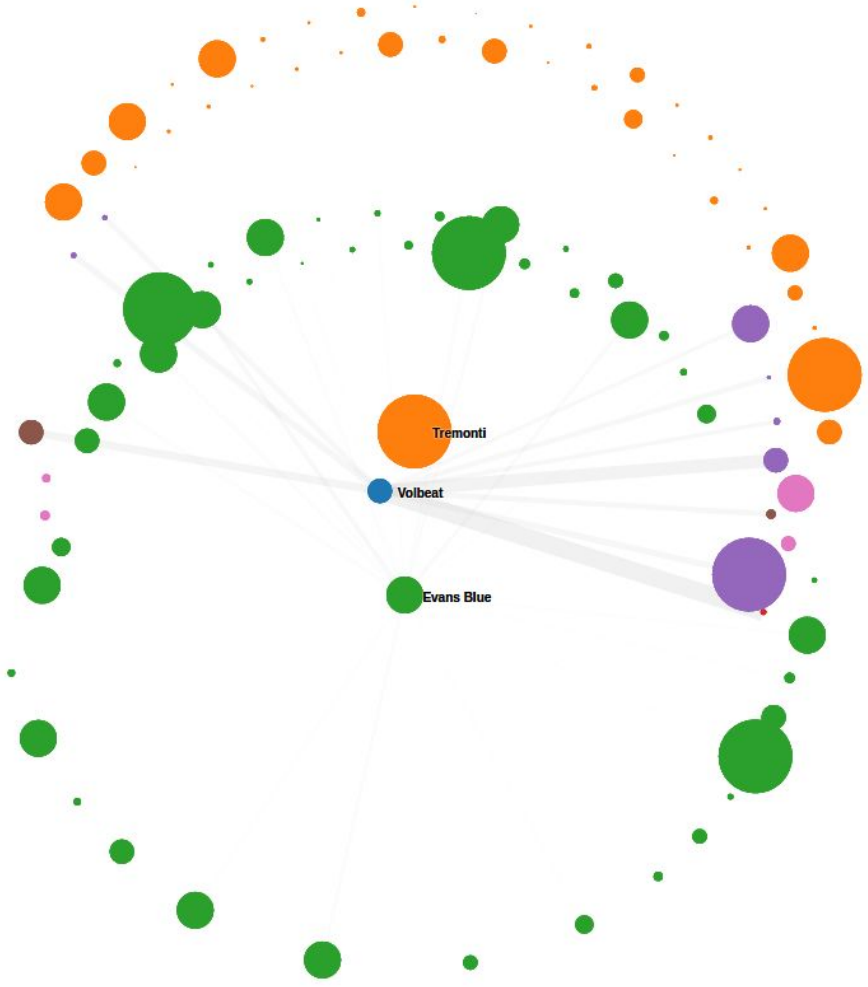

Fig 7. Many of the edges linking nodes to Tremonti and Evans Blue are extremely difficult to see, however the edges connected to Volbeat look fine. Edges values vary greatly depending upon the source artists, and therefore are difficult to consistently use to calculate an optimal edge width and opacity.

Conclusion and Future Works

While the graph did not reveal any information about subgenre clusters within music artist communities as originally intended, it does do a pretty good job of displaying related artists across source nodes. It would have been nice to make larger graphs and find even more connections, though the cost of issuing API calls and lack of color schemes would have extended much further the size of the already limited graph. Therefore, we are satisfied with the results: we have provided information about artists' connectedness based on performance history for a relatively small number of artists, but are disappointed that it could not be scaled up to view more complex relationships.

There are some problems with the visualization that we would like to see fixed in future versions. As mentioned before, the algorithm for calculating edge width and opacity could definitely be improved. There was also a problem with the text hover feature, where sometimes the text used to display an artist's name would be displayed underneath surrounding nodes, thus obscuring the name. We tried to fix it, but were unable to remove this problem. The layout algorithm could be improved as well, as right now nodes can overlap each other, especially when they are larger in size. Lastly, the speed of data collection and conversion to a JSON format for the actual graph could always be improved to create a quicker experience.

Something to keep in mind are the other relationships which artists may have between one another; one set of data, which is not really relevant for concerts, is the set of covers and collaborations artists play; a cover usually shows what an artist's influences are, and collaborations possibly show a physical connection.

Team

Jason accessed the Songkick data, and from that formatted the data into a .CSV file using a Python script; later, he took the same Python script and used it to convert the Songkick data into a modified JSON file, which contained the nodes and links. The nodes were in the

format of { source: {true, false}, group: {integer}, ‘id’: {string}, ‘value’: {integer} } and the edges were in { source: {string}, target: {string}, id: {randomized id}, value: {integer} } he also helped debug the Javascript code and suggested the node size, opacity and stroke-width calculations.

Justin collected the initial Muse data from a fan website while we were still waiting for the Songkick API key, and using a C program converted the information from a .CSV file into Graphviz lines for the Tuesday demo. He then worked on the D3 Javascript file, mostly getting the collapse-node and text-label-appearing-on-mouseover functions to work.

Both team members worked on this report and the presentation slides together.

Finally, for personal use of the application, we have set up a GitHub page: https://github.com/justinlee0777/music-artists-graph.

Paper Citations

Herman, Ivan. “Graph visualization and navigation in information visualization: A survey,” *IEEE Transactions on Visualization and Computer Graphics,* Vol. 6, Iss. 1, pg. 24 - 43, Jan-Mar 2000. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=841119&tag=1

Newman, M. E. J. and Girvan, M. “Finding and evaluating community structure in networks.” Physical Review E, Vol. 69, Iss. 2, February 2004. http://journals.aps.org/pre/abstract/10.1103/PhysRevE.69.026113

Carpendale, M.S.T. “Considering Visual Variables as a Basis for Information Visualization.” University of Calgary, Department of Computer Science, 2001-693-16, 2003. https://cdn.mprog.nl/dataviz/excerpts/w2/Carpendale_Considering_Visual_Variables.pdf

Cano, Pedro, and Markus Koppenberger. "The emergence of complex network patterns in music artist networks." Proceedings of the 5th international symposium on music information retrieval (ISMIR 2004). 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.3220&rep=rep1&type=pdf