Hip-Hop to Deep House: Navigating the Music Graph using Decentralized Search

Group 37: Alexander Embiricos, Salman Quazi, Prasanth Veerina
{embirico, salmanq, pveerina}@stanford.edu

December 10, 2013

1 Introduction

Music is considered by many an integral part of the human experience. Indeed, most cultures and subcultures around the world are associated with specific musical tastes. In this project we are interested in exploring the properties of music graphs constructed using attributes collected from several music datasets. In particular the problem we are trying to explore is something most of us have faced: how do we pick the next song to listen to. We examine how we can generate music graphs, what the important parameters are, and whether these music graphs are navigable using decentralized search. Ultimately we hope to gain some insight that can help us build interesting applications like a system that can smoothly interpolate playlists between pairs or lists of user defined songs.

2 Relevant Work

In our review of prior work we found that there were fundamentally three unique approaches to solving this problem: one that looks at the acoustic properties of a song, another that looks at the metadata about a song, as it relates to its neighbors, and lastly, one that looks at the metadata about a song in addition to listening behaviors and other social graph constructs. The problem with the first approach is that there's a fundamental gap in the semantic property between the low-level acoustic features of a song and the high level music concepts Bu et al. (2010) that is important to the listener.

There are hybrid solutions to the problem by combining acoustic based and collaborative filtering music recommendations to improve overall accuracy of predictions. One such approach was taken by Bu et al. (2010) in their "Music Recommendation by Unified Hypergraph: Combining Social Media Information and Music Content". In this paper, the authors produce a hyper graph which includes: various types of objects and relations in the social music community from Last.fm. The relations include friendship relations, membership relations, listening relations, tagging relations, inclusion relations among resources for instances tracks, and albums and similarity relations between music tracks. All of this together resulted in a single hypergraph.

Thinking about our ultimate project goal of building a music playlist interpolator, we wanted to have a searchable graph that reflected similarity in more than one dimension (i.e. not just acoustic similarity, not just co-listens between different users), but we wanted to avoid building a complicated hypergraph as done by Bu et al. (2010). The motivations for these objectives were to 1) make good recommendations and 2) have a simple graph that could be quickly searched in the context of a web application.

This leads us to the most important reference for our work: Watts et al. (2002). In short Watts et al. propose a way to generate and then search social networks using various hierarchical identities. The resulting graph is a simple undirected edge list of individuals that know each other, where the likelihood of individuals being connected is related to how similar they are. The way Watts et al. describe similarity is also interesting in that they consider several axes of similarity (e.g. area of study and location for students) and take the social similarity to be the minimum of those similarities. They then propose an efficient and 'realistic' search algorithm where each node forwards to the neighbor which is most socially similar to the target node.

As we will discuss below, we have adapted the model and search algorithm from Watts et al. (2002): our nodes are artists and identity is measured along axes such as musical similarity or number of co-listen occurrences etc. (Ultimately however, due to poor searchability of the colistens graph, to be discussed below, our final graphs measure identity only along a single axis, such as acoustic similarity.) One important modification to note is that since metrics such as musical similarity are non-hierarchical but still intuitively make sense to us as a form of identity, we are not constraining ourselves to hierarchical relationships. Our non-hierarchical metrics maintain the same interface with the model by providing a score where lower values indicate higher similarity.

3 Data Collection

3.1 Data sources

In order to actually start building music graphs we collected data from a number of sources. Here we describe our datasets.

3.1.1 Last.fm Track colistening data

The first dataset we found was a collection of listening histories for 1000 Last.fm (social music website) users. (las, 2013) (Celma, 2010) The dataset contained a list of anonymous users and a list of the tracks that each user listened to over a period of time. From this dataset we created our first co-listening graph: if user A listened to track x and also track y, then x and y are connected by an edge. However for reasons described in section 3.1.2 we chose not to use this dataset for now.

3.1.2 Last.fm Artist co-listening data

Similar to the track co-listening data, we found a large Last.fm dataset which lists the most listened artists for 360k users. (las, 2013) (Celma, 2010) This dataset contains anonymized users, the name

of the artists they listened to and unique artist identifiers. While this data is less granular and perhaps less useful for ultimately creating track recommendations, we chose to use this dataset over the track co-listening dataset for the following reasons:

- Richer data set: the artist co-listening set describes the listening history of 360k users, compared to 1k users for the track set.
- Smaller graph: Artists and tracks have a one-to-many relationship so a graph where nodes are artists will be smaller and more manageable. Tracks from a given artist are highly clustered to one another in graphs where edges are more likely when tracks are more similar. As such you can look at artists as hypernodes.
- Cross-referenceability of data: assessing node similarity along multiple axes is an important part of our project, meaning that we need a way to map nodes from different graphs (from different sources) to each other. As will be described below, it is much easier to find information on artists from other sources than it is to find information on tracks.

We used this dataset to compute the number of times every pair of artists in our target artist set were listened to together. (Selecting the set of target artists is described in section 3.2)

3.1.3 MusicBrainz unique identifiers

MusicBrainz is a publicly maintained music metadata database. We use MusicBrainzIDs as the identifying glue between the various components of our project.(mus, 2013) These MBIDs are unique identifiers for the objects we are working with, be they artists or songs, and artist MBIDs tend to be directly provided for in the various datasets or APIs we are working with.

3.1.4 Echo NEST musical profile data

Echo NEST is a music intelligence platform that can provide a variety of information about tracks, artists etc. Of particular interest to us are the Echo NEST Acoustic Attributes for tracks. These can be used to compute the similarity between tracks and thus indirectly, artists.

We used the Echo NEST API to create artist profiles: These profiles are feature vectors representing the average acoustic values of an artist's hottest 5 songs. We use this vector as a sort of musical identity. (See Table 1 for details). We allow Echo Nest to select the 'hottest' songs. While the calculation is opaque, they describe it as "what songs are trending or hot right now". Because the various musical features have different ranges, we normalized their values to lie mostly between 0.0 and 1.0.

We collected our artist profiles by querying against the EchoNest API for the target artist MBIDs collected in the Last.fm artist co-listening dataset. Because the Echo NEST API is rate limited, we store the artist profiles of the artists in our graph in advance.

Artist Profile	API Value	Normalized Value
Danceability	0-1	0-1
Energy	0-1	0-1
Loudness	-30-0 *	0-1
Speechiness	0-1	0-1
Tempo	60-200 *	0-1
Acousticness	0-1	0-l

Table 1: * The Echonest API value is not strictly limited to this range, but the range shown is reasonable and is supported by observing values for various types of music.

3.1.5 Last.fm API

In order to evaluate the search results from our graphs we collected data from the Last.FM API. Using the `Artist.getSimilar` call we gathered the similar artists (with a normalized similarity score) for our target artists. Given that LastFM returns a locally normalized value between o - 1 (o meaning not similar at all and 1 meaning the same) we normalized this value by $(1 - similarityScore)/2$. In other words, in our network a similarity score of 0 means artists are closest meaning the most similar. We describe how this data is used for evaluation in the results section.

3.2 Data Selection

Before collecting data points from the above sources, we made a sub selection of artists that would serve as nodes in our graphs.

First, due to computational constraints we limited our dataset to only 10,000 artists.

Since all of our sources other than the co-listening data are queryable APIs, we used the artists in the Last.fm artist co-listening data as our base set of artists. From within this set, we selected the 10,000 artists with the most overall plays. While we also considered choosing the 10,000 artists who have had the greatest number of users listen to them, the two sets differ only by around 5% of elements.

4 Models and Algorithms

4.1 Graph generation & distance metrics

Using the data collected above, we created music graphs where the nodes are artists and the probability of an edge between two artists is defined by the following binomial probability distribution function: $p(e) = ce^{-\alpha\delta}$, where c is some constant (taken to be $c = 1$ for now), α is a tunable constant, and δ is some measure of distance between the two artists. This model allows us to modify the structural properties of the network by simply modifying the value for α. Our generated graphs are described in section 5.2.

4.1.1 Echonest Distance

The first distance metric we defined is the Euclidean distance between the normalized music feature vectors found by querying the Echo NEST API. (See Table 1 for a reminder.) In other words, given the musical attribute vectors $\mathbf{m_a}$, $\mathbf{m_b}$ for artists a and b we have:

$$EchonestDistance(a, b) = \frac{||\mathbf{m_a} - \mathbf{m_b}||}{C}$$

Where C is a normalizing constant defined to be the maximum distance between any two artists in our dataset (found empirically to be roughly 1.87).

4.1.2 Colisten Distance

Next, we implemented a distance metric based on the Last.fm artist co-listening data from Section 3.1.2. We have defined this metric as one minus the normalized number of co-listening occurrences between the two artists a and b.

$$ColistenDistance(a, b) = max(0, 1 - \frac{\text{\# colistens for (a, b)}}{C})$$

Again C is a normalizing constant. Initially we let C be the maximum number of co-listens in our dataset however this did not work since our dataset contained some outlier pair of artists that had over 16000 co-listens (See section 5.1). This normalization caused the distance metric to be poorly distributed. Instead we choose to use $C = 35$ as we determined that 35 is number of co-listens at the 95th percentile of the data. In order to handle cases in which the number of co-listens was greater than 35 we simply used max to clip the distance to 0.

4.2 Search Algorithm

The main algorithm used in this project is decentralized search. The objective is given a starting node representing artist a find a path to artist b in the graph. The search is a simple greedy algorithm that works with no global knowledge of the graph other than the musical identity of the target artist. We simply start at a given node, and using only the local knowledge of neighbors (and their musical identities) we select a neighbor according to our selection strategy until we reach the target or surpass a maximum number of steps. We keep track of nodes we already visited in order to prevent infinite loops.

There are several different selection strategies one might use. The simplest strategy is greedily selecting the next node based on highest degree. A more intelligent strategy might select the next node based on the artist's musical distance to the target node (as defined in the previous section). We can also use a search strategy that uses the min distance over all our distance metrics. In our experiments below we use the strategy of selecting the neighbor node which has the minimum distance to the target node according to the distance metric used to generate the graph.

5 Results

5.1 Distance metric statistics

Before actually generating our graphs we had to compute the distance metrics from our datasets, in this section we describe the results of this computation.

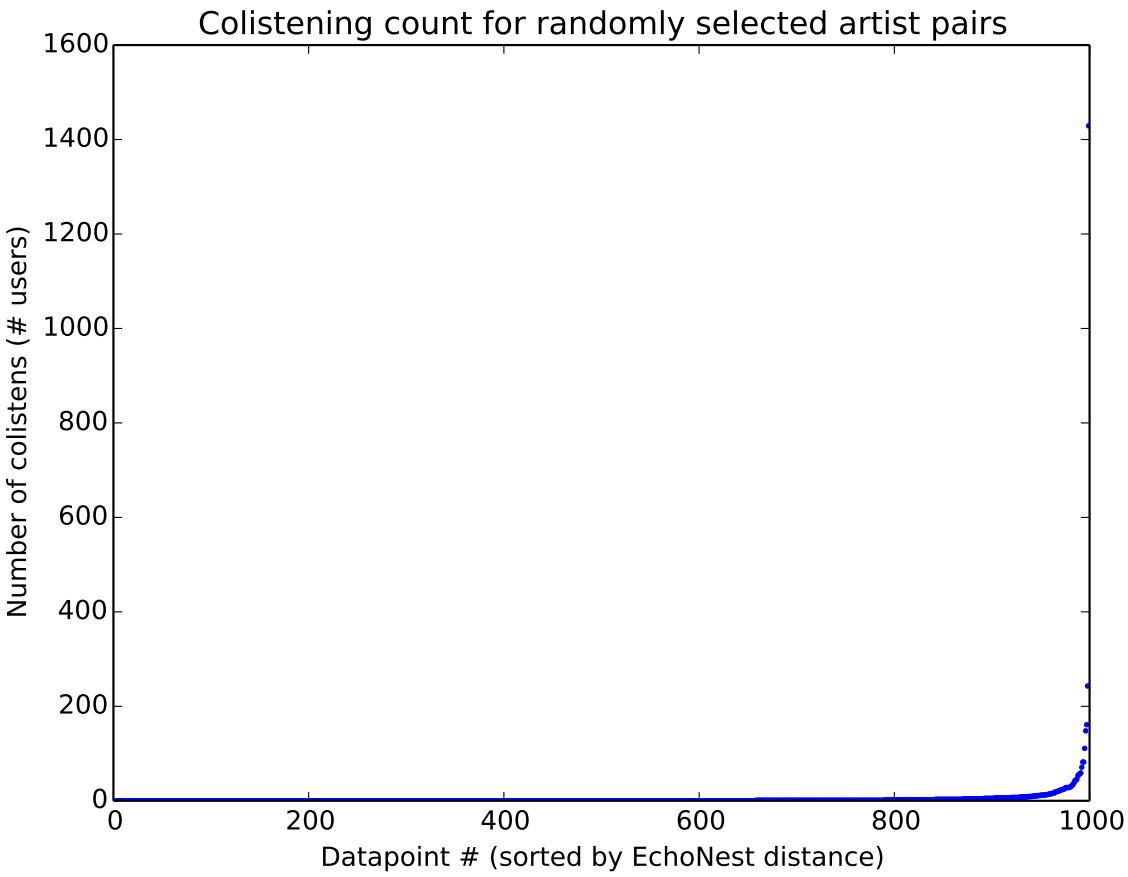

Figure 1: Colistening count for randomly selected artist pairs

Figure 1 shows the raw colistening count for 1000 randomly selected artist pairs. (I.e. "# of users that listened to both artists a_1 and a_2.") In other words, co-listens define a metric measuring if two artists are likely to be simultaneously preferred by a user.

It is interesting to see the wide range of values in this metric. As expected, many pairs of artists had 0 co-listens. However a small number of outliers had tens of thousands of colistens. For reference, the max number of co-listens was 16123 co-listens (The Beatles and Radiohead), while the 95th percentile was only 35 co-listens. This skew in the data makes the plot appear as though most data points are virtually 0 co-listens. This plot informed the decisions described in Model section 4.1.2.

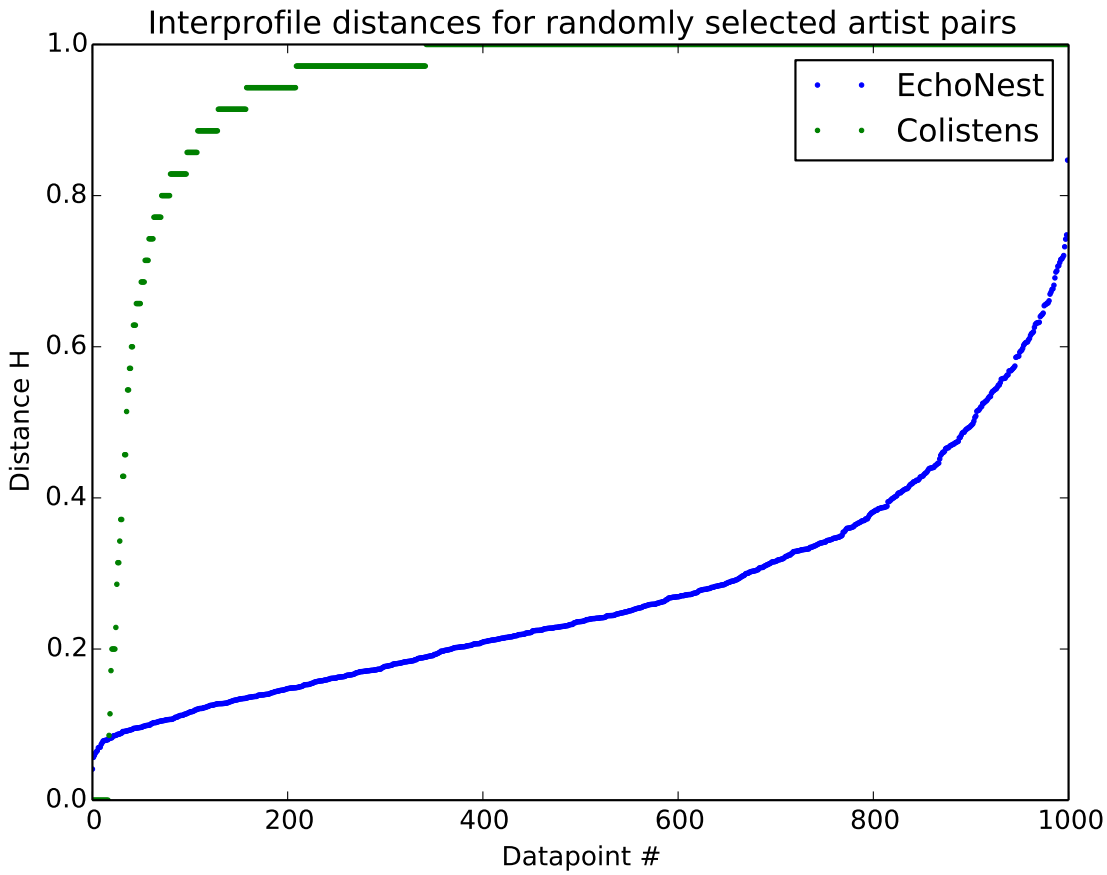

Figure 2: Interprofile distances for randomly selected artist pairs

In Figure 2 we plot the distances between randomly selected artist pairs. As described above, our distance metrics are normalized such that they value between 0 and 1, where 0 represents similar artists, while 1 represents dissimilar artists. Note that the data points for the two series are sorted independently.

We see that the distribution of distances among artist pairs is quite different between the two metrics. Echonest has a smooth range of values between 0 and 1 while the co-listens distances are heavily skewed to be nearer to 1 and are more striated since colisten counts are constraited to integer values. However, we might still hope for some correlation between Echonest distance and colisten distance. Unfortunately, Figure 3, which plots the sorted Echonest distances alongside the corresponding colisten distance, demonstrates that these two metrics are not correlated at all.

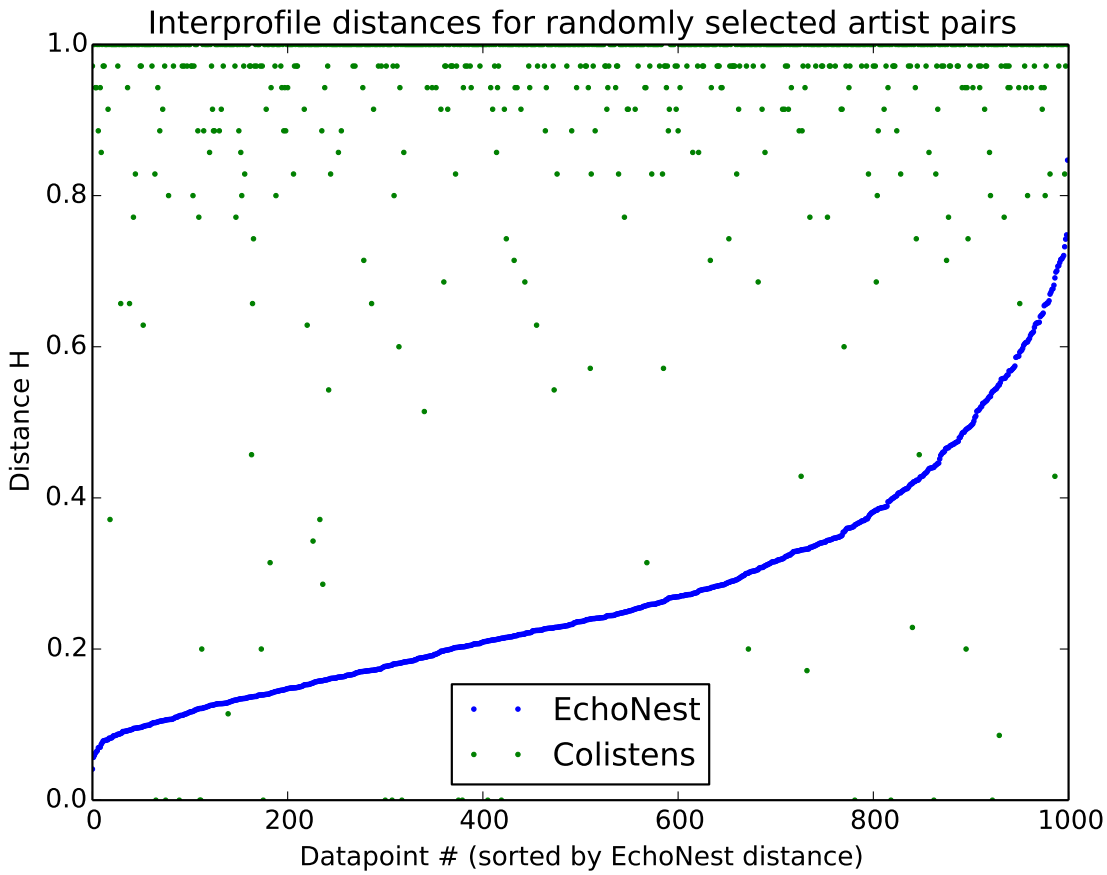

Figure 3: Interprofile distances for randomly selected artist pairs, sorted by Echonest distance

5.2 Basic graph statistics

As we have discussed above, we compute the probability of an edge by $p(e) = ce^{-\alpha\delta}$ where α is some parameter that multiplies the distance δ between two artists. Tuning α allows us to modify the structural properties of the network so that we can perform various analysis.

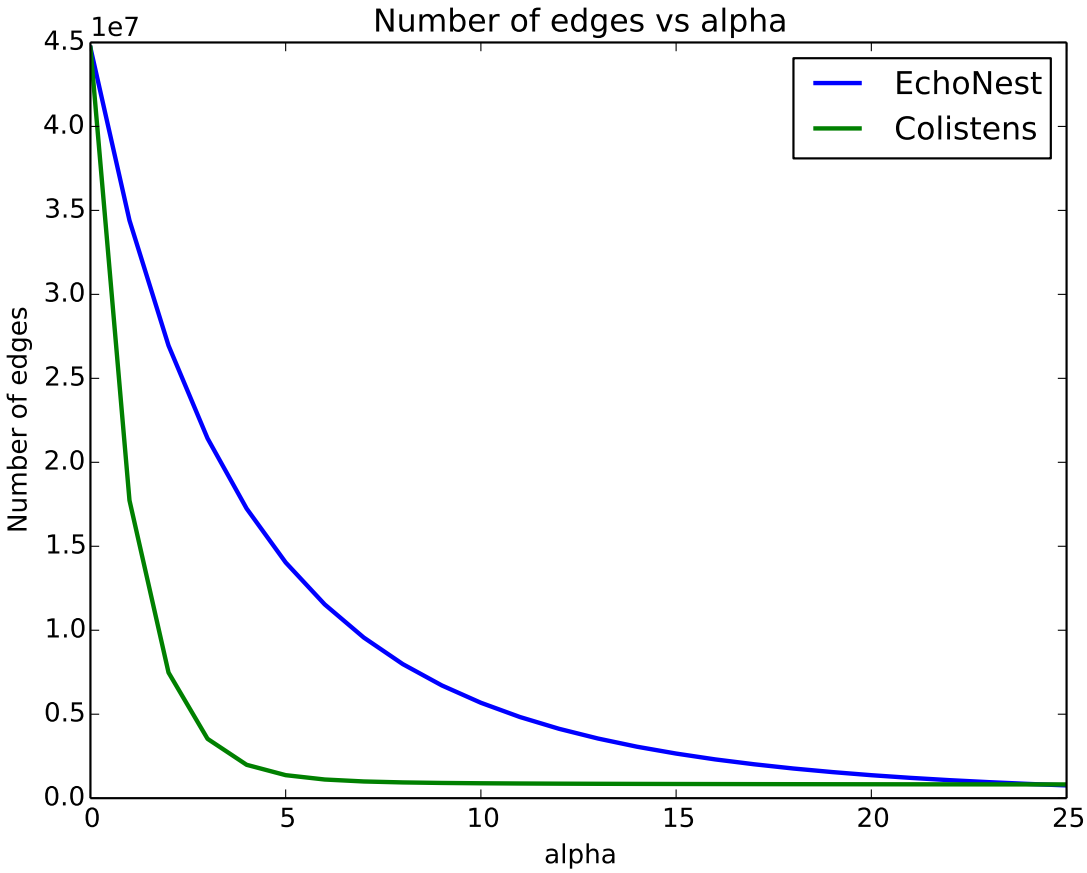

Figure 4: Number of edges vs. α

As expected, we see how increasing α reduces the number of edges in the overall network in figure 4. The number of edges decreases for the colistens graph more quickly than for the EchoNest graph because, although the distance functions in both cases range from 0.0 to 1.0, the expected value is lower for the Co-listens dataset. We limited our range of alphas between 0 and 25 after seeing the graph become increasingly sparse at higher alphas.

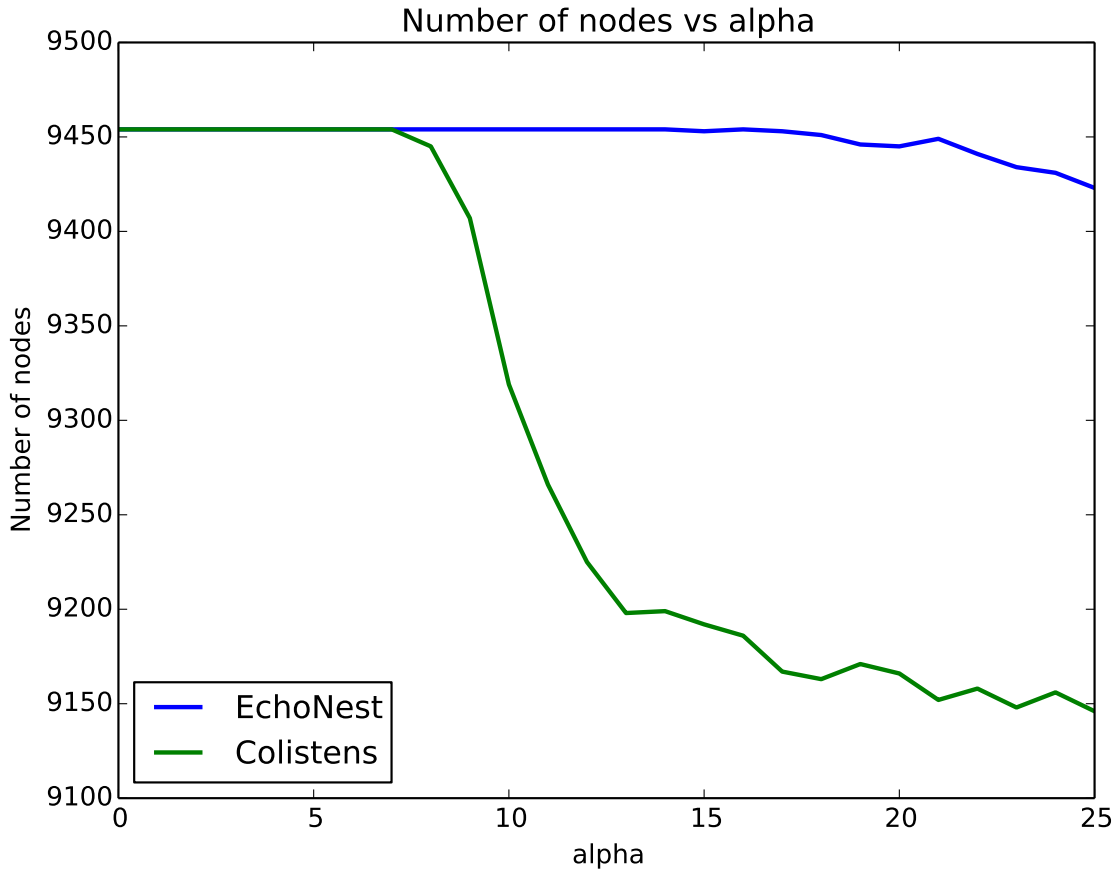

Figure 5: Number of node vs. α

Although varying α does not explicitly change the number of nodes in our graphs, it is possible that by removing edges a node becomes completely isolated and inaccessible. Since we represent our graphs as edge lists, these nodes cease to exist in our graphs.

Thus, in figure 5, we show how modifying α changes the number of nodes in the graph. It is interesting to note how the EchoNest nodes remain relatively constant whereas the Last.FM number of nodes drops sharply especially at $\alpha \geq 7$. Although we would expect the number of nodes to decrease monotonically with the number of edges, there is a random component to graph generation.

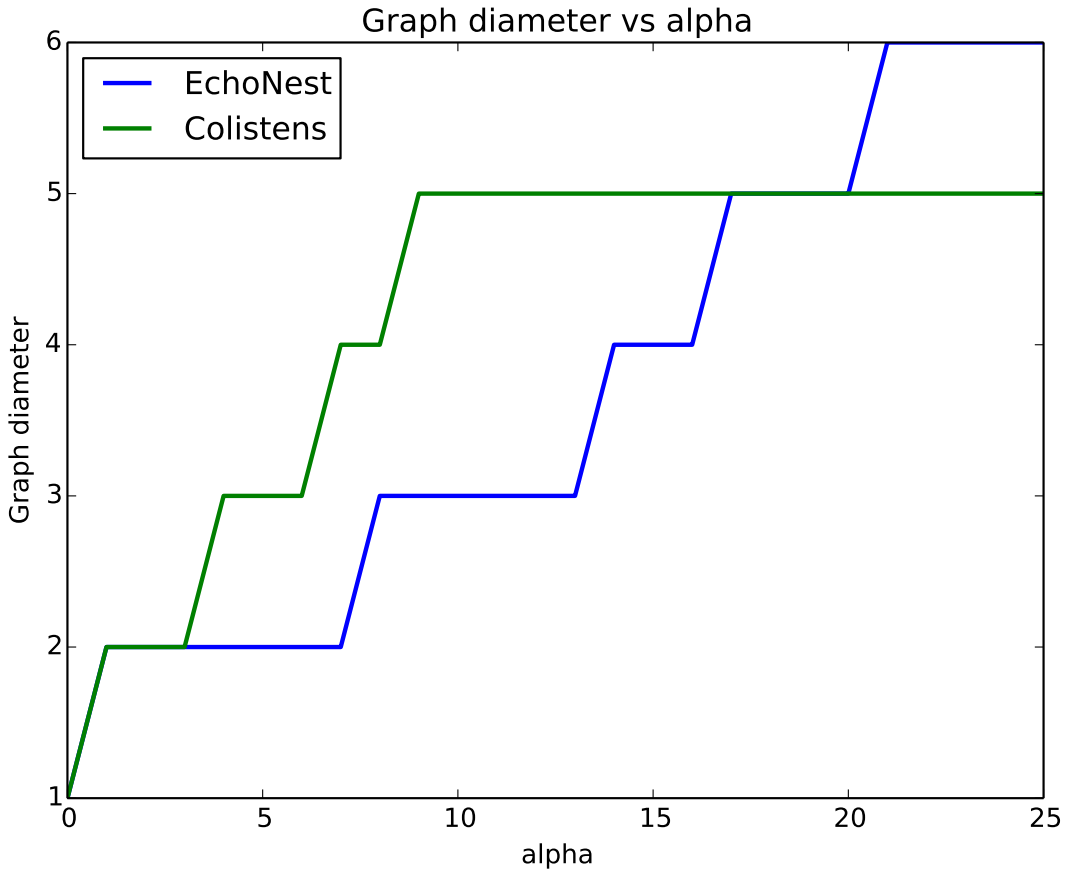

Figure 6: Graph diameter vs. α

In figure 6, we show how the diameter of the graph increases as we increase α. This is because as we increase the α we reduce the probability of an edge between two nodes (especially between distant nodes) and therefore reduce long range edges in the network. Still however, up to alpha 25 the diameter is quite small, never exceeding the so called six degrees of separation.

Finally, Figures 11 and 12 in Appendix A demonstrate that our model does not suffer from creating isolated components of any significant size: the entire graph is basically a single SCC, perfect for fully searchable graphs.

5.3 Searching the graph

The following sections describe the results of running decentralized search on our graphs.

5.3.1 Successes and failures

Recalling that we initially set out to create graphs which we could then traverse to get from a given artist to a target artist, the most basic measure of success is whether or not our search algorithm successfully finds the target artist. (Recall that our decentralized search only has local knowledge of the node it is on and knowledge of the distance between it's neighbors and the target node).

We say that a search is unsuccessful (i.e 'fails') if any of the following criteria are met:

- Start node not in graph.
- Destination node not in graph.
- We reach a dead end node where there are no outgoing edges to unvisited nodes.
- We have visited over 200 nodes without reaching our destination. (To understand why this limit makes sense, consider that, even at the highest value of α, visiting 200 nodes would mean traveling over 30x the graph diameter.)

As shown in in Figure 7, out of 100 randomly selected searches (i.e. start and destination nodes are randomly selected), decentralized search on the EchoNest graph 'never' fails, whereas search on the colistens graph nearly always fails.

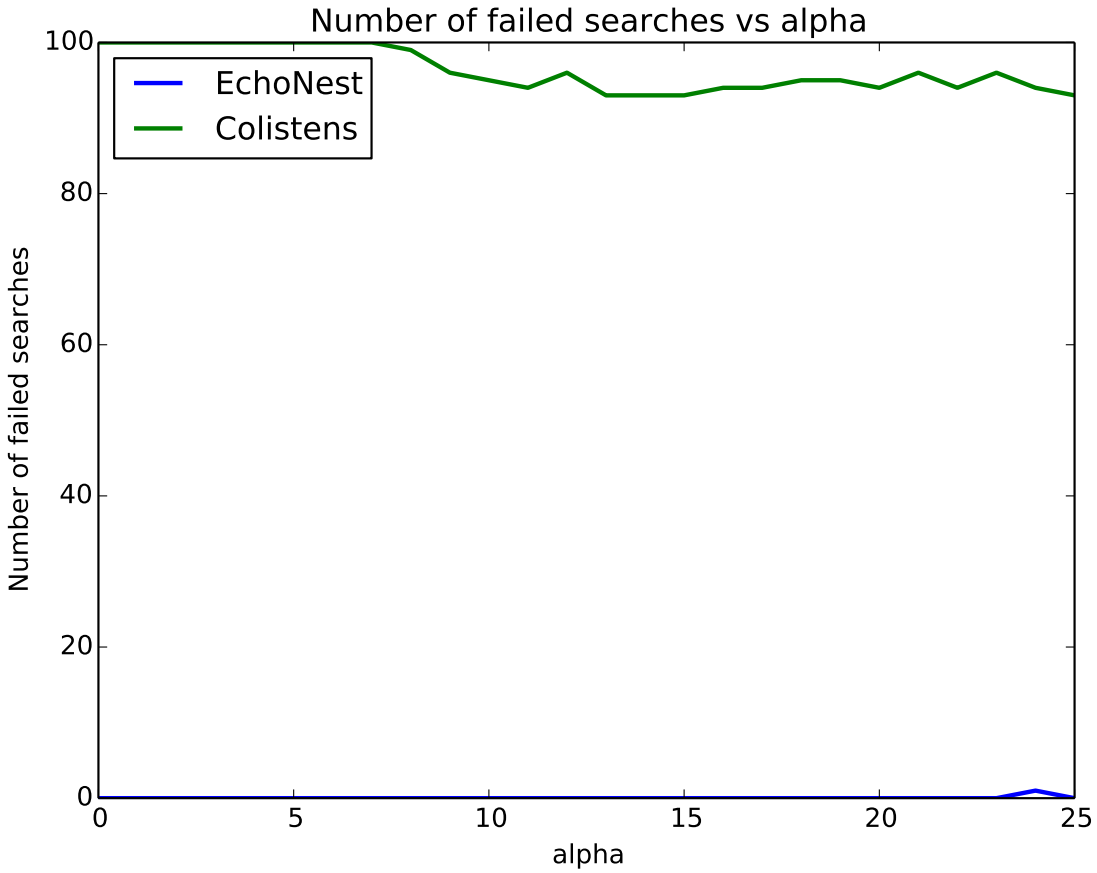

Figure 7: Number of failed searches vs. α. (100 searches performed in total)

To attempt to understand why the colistening graph has so many failures, we plot in Figure 8 the average search path length. (To see the distribution of path lengths for a given α, please refer to Figure 14 in Appendix A.)

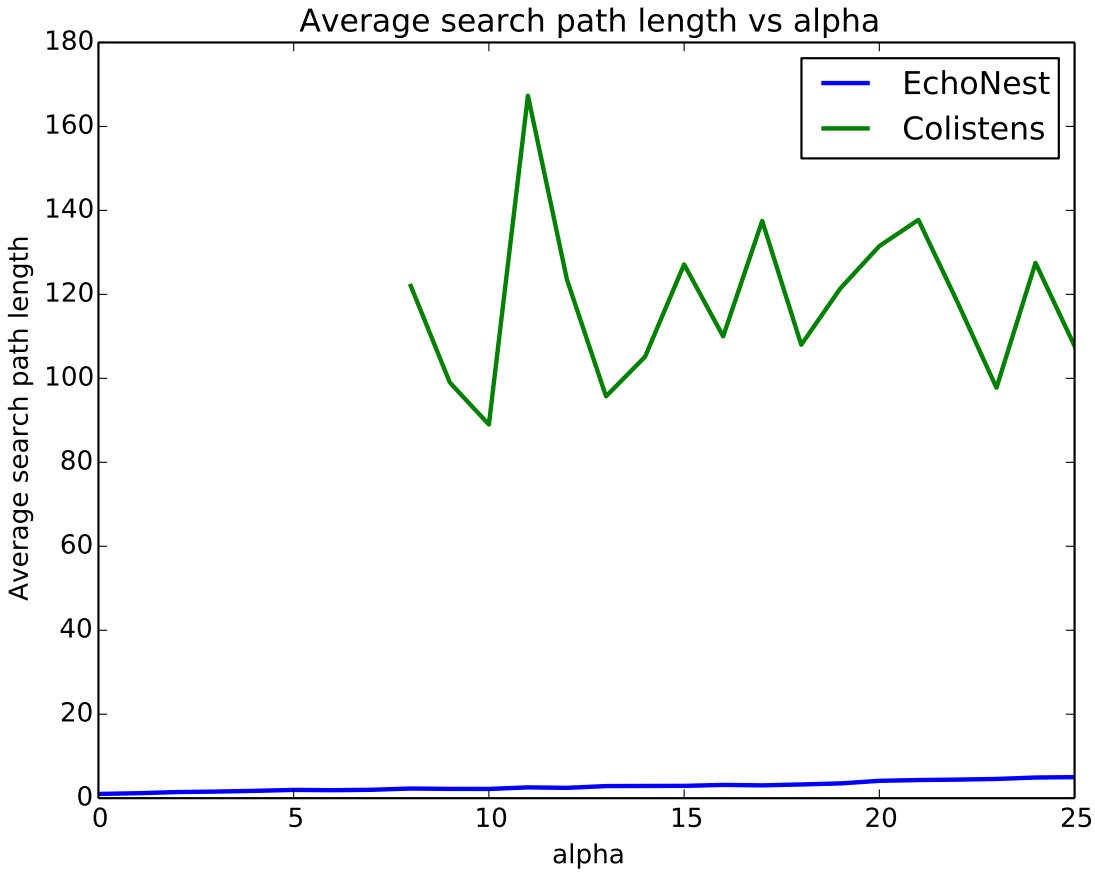

Figure 8: Average path length vs. α.

First we note that the Echonest path length increases linearly with α: this is expected and convenient should we wish to create search paths (i.e. playlists) of a certain target length. Next, we note that the colistens searches, when not failing, are of much higher path lengths. Furthermore, most searches are 'timing out', artificially lowering this value. We also note that the average length is extremely volatile. This leads us to believe that successful searches in the colistens graph are essentially random, 'lucky stumbles' on the target node.

Thus we conclude that we succeeded in creating a searchable graph from the Echonest dataset, but not from the colistens dataset. One possible explanation of this is that colistened tracks are not actually similar, but perhaps instead are both popular and fit people's musical taste profiles.

5.3.2 Evaluating the paths: Are they smooth?

Now that we can successfully complete searches, we also recall that we desired 'smooth' transitions between artists. This is a difficult metric to measure, so we use several approaches:

At a most basic level, we apply the distance metric for a given graph to each search path. For example, on the EchoNest graph, we can compute the average Echonest distance between each pair of nodes on the search path. Figure 9 shows this value averaged over 100 random searches:

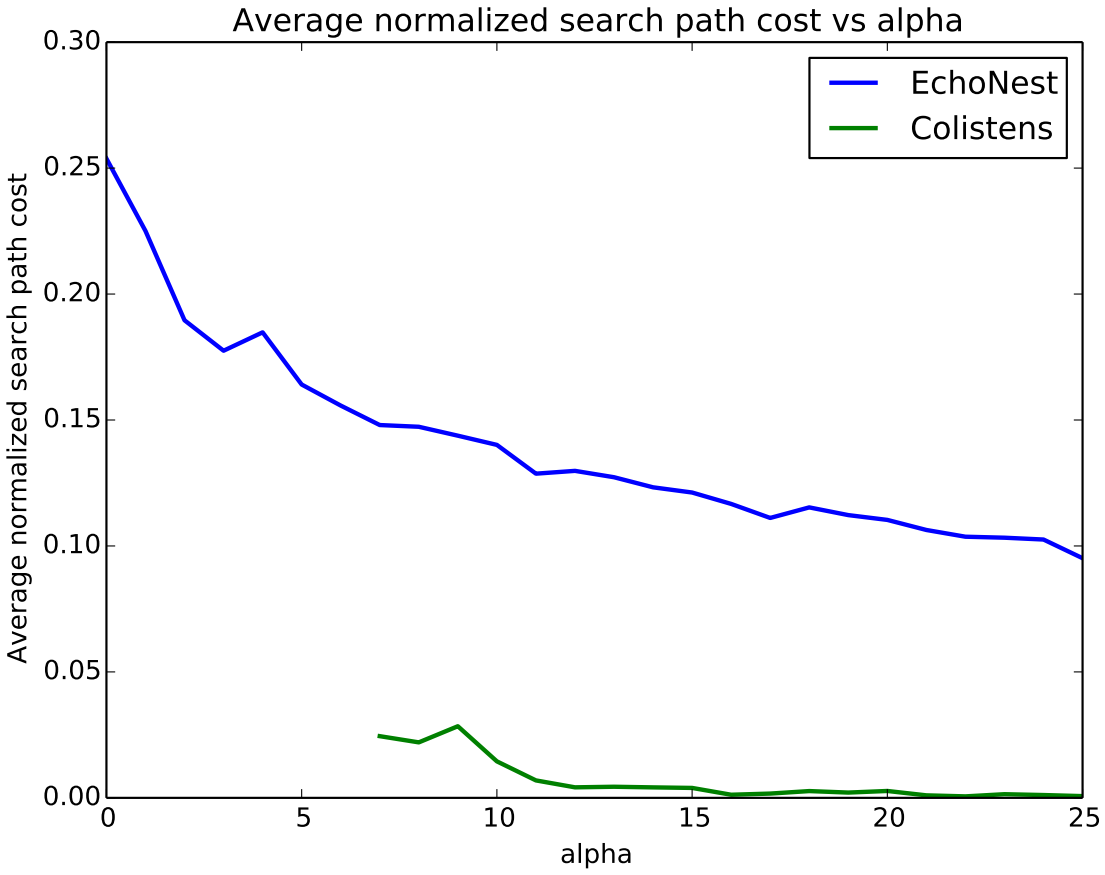

Figure 9: Average Normalized search path cost vs. alpha

From Figure 9, and recalling that the distance cost ranges from 0 to 1, we can observe that both graphs offer 'smooth' transitions. We observe that lowering α forces smoother transitions, as it is less likely for edges to exist between distant nodes. Coupled with the increase in search path length, we see that at higher α, our model takes more steps, but smaller (i.e. smoother) steps, to reach its destination.

We also note that steps taken by the colistens search are very smooth according to the colistens distance metric. This makes sense given the distribution of colistens distance values: most nodes are considered to be far and are thus less likely to be connected by an edge, while a very small number of nodes are extremely close and are thus connected.

However, due to the way that we constructed the graphs, the above is not surprising. We thus chose to attempt to evaluate our path transitions using another dataset collected from the Last.fm

`Artist.getSimilar` API call. As described in section 3.1.5, we transform that dataset into a metric which yields a distance score between 0 and 1. Since we return a distance of 1 for any transition to an artist not present in the 15 similar artists returned by the API, we expect most transitions to yield a distance of 1, meaning that any time the value is lower than 1 is a significant success.

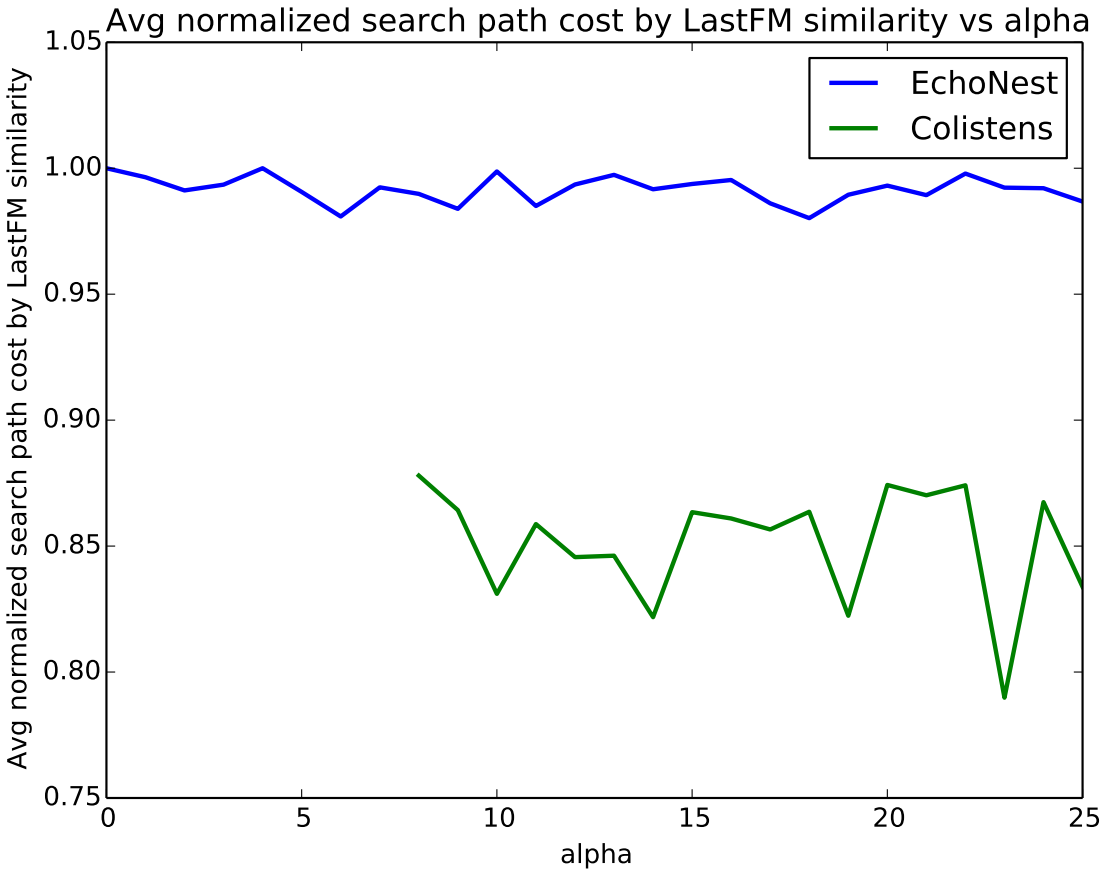

Figure 10: Average normalized search path cost by LastFM similarity vs. α

Figure 10 shows the average cost of a search path transition, averaged over 100 searches. Unfortunately, but as expected, the EchoNest searches usually 'miss' Last.fm's similar artists, yielding average values very near 1.

On the other hand, the colistening transitions seem to be more correlated with the Last.fm similar artists. As described above, we do not believe that the colistening-based distance metric is an accurate measure of similarity. However, it does not seem unreasonable to conclude that, given the correlation shown, and given that the colistening dataset is also from Last.fm, the Last.fm similar artists database includes some form of colistening computation.

5.3.3 Qualitative results

In the end what we are trying to produce is the best transition to a target artist for our listeners. We decided to take two popular artists that are musically very different from each other and run the decentralized search against it.

State	Artist
Source Artist	Radiohead
Transition	Seth Lakeman
Transition	Tenniscoats
Target Artist	The Album Leaf

State	Artist
Source Artist	Coldplay
Transition	Afasi and Filthy
Transition	Danger
Transition	Ya Boy
Transition	Ill Bill
Target Artist	50 Cent

Even though we started with a limited dataset you will notice for the first example a transition from Radiohead to Seth Lakeman is almost a perfect transition for the target artist Album Leaf. Seth Lakeman provides the tonality and musical ambiance that one would expect from artist Album Leaf and yet has the vocal filling we would expect from Radiohead. The transitions are not always as smooth however. For our second sample, we intentionally picked two very disparate artists, Coldplay and 50 cent - and it's immediately becomes apparent when you notice Coldplay, a British alternative rock group, transitioning to Afasi and Filthy, a Swedish hip hop group, which is a more abrupt transition than one would expect.

6 Conclusions

In this project we set out to explore the properties of music graphs and to try and navigate them using decrentralized search. We found that using two different distance metrics to generate our graphs produced two very different outcomes. Generating graphs based on our colisten distance produced graphs which were essentially unusable for our purposes. On the other hand generating graphs based on Echonest musical distance produced highly searchable graphs that we were able to use to actually produce some interesting playlists. As part of our project we also prototyped a web application that generates and plays playlists (on the popular Spotify music service) given a starting and ending artist using the technique described in this paper. Screen captures are included in the appendix.

References

(2013), "last.fm." URL http://Last.fm.

(2013), "Musicbrainz database." URL http://musicbrainz.org/doc/MusicBrainz_Database.

Bu, Jiajun, Shulong Tan, Chun Chen, Can Wang, Hao Wu, Lijun Zhang, and Xiaofei He (2010), "Music recommendation by unified hypergraph: combining social media information and music content." In *Proceedings of the international conference on Multimedia*, 391–400, ACM.

Celma, O. (2010), *Music Recommendation and Discovery in the Long Tail*. Springer.

Watts, Duncan J, Peter Sheridan Dodds, and Mark EJ Newman (2002), "Identity and search in social networks." *science*, 296, 1302–1305.

7 Appendix A - Supporting Diagrams

Figures 11 and 12 show the sizes of the largest SCC and WCC of our graphs (which are the same since our graphs are undirected). Noting the y-axis scale, we observe that the graphs are well connected with few isolated components even at larger values of α. This is great for graph searchability.

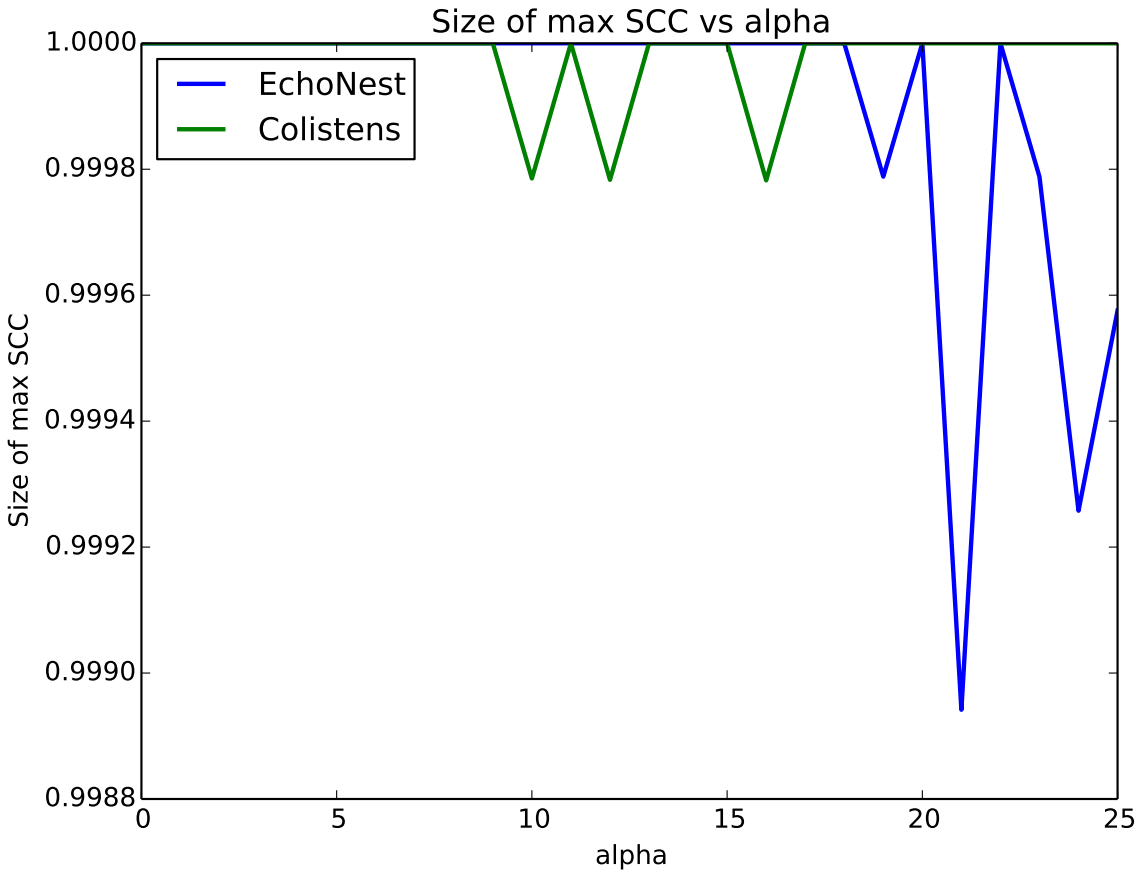

Figure 11: Size of max Strongly Connected Component (SCC) vs. α

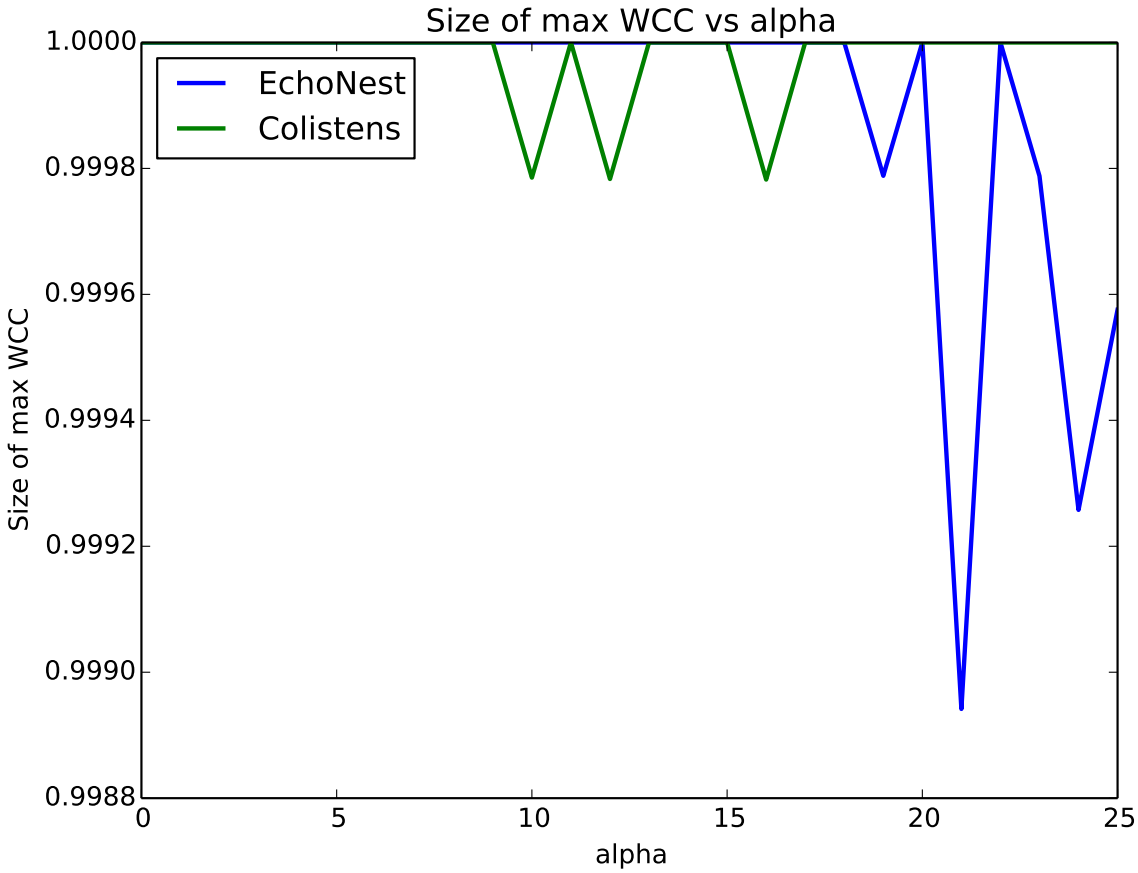

Figure 12: Size of max Weakly Connected Component (WCC) vs. α

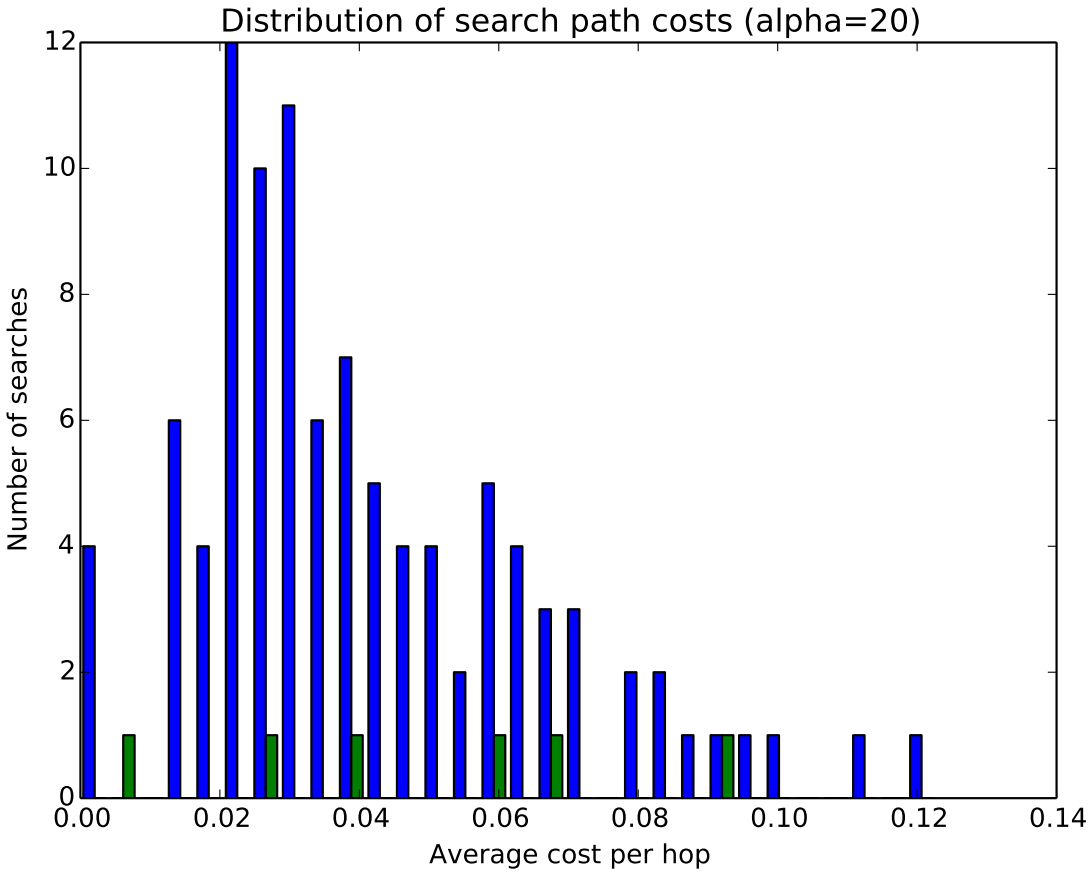

Figure 13: Distribution of search path costs at $\alpha = 20$

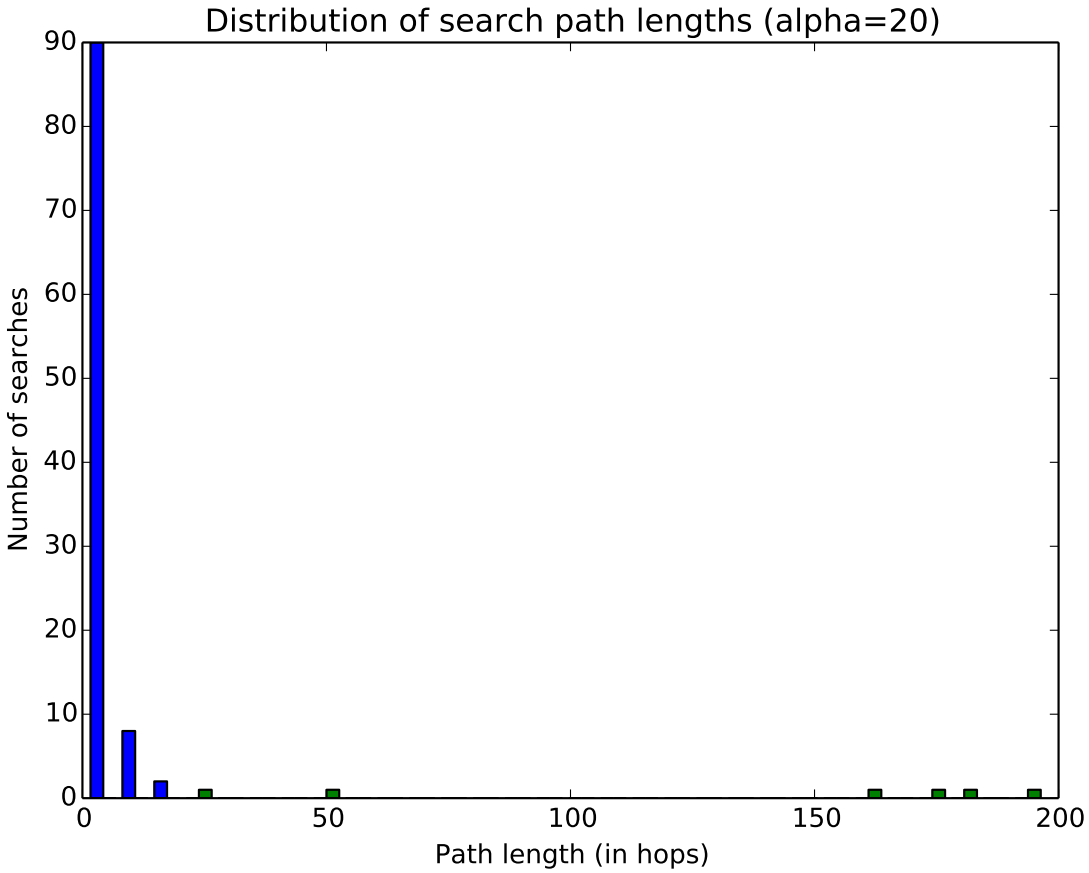

Figure 14: Distribution of search path lengths at $\alpha = 20$

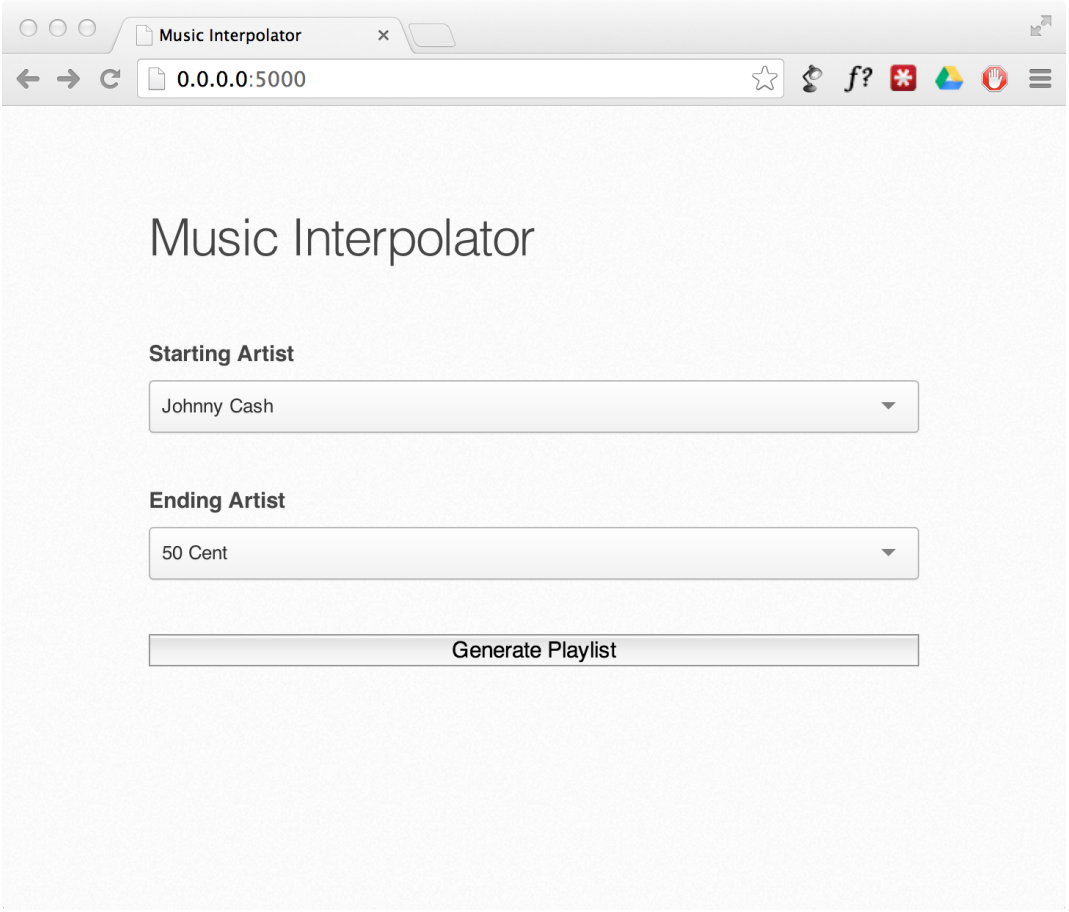

Figure 15: A screen shot of our prototype web application. This page allows the user to select a starting and ending artst for their playlist.

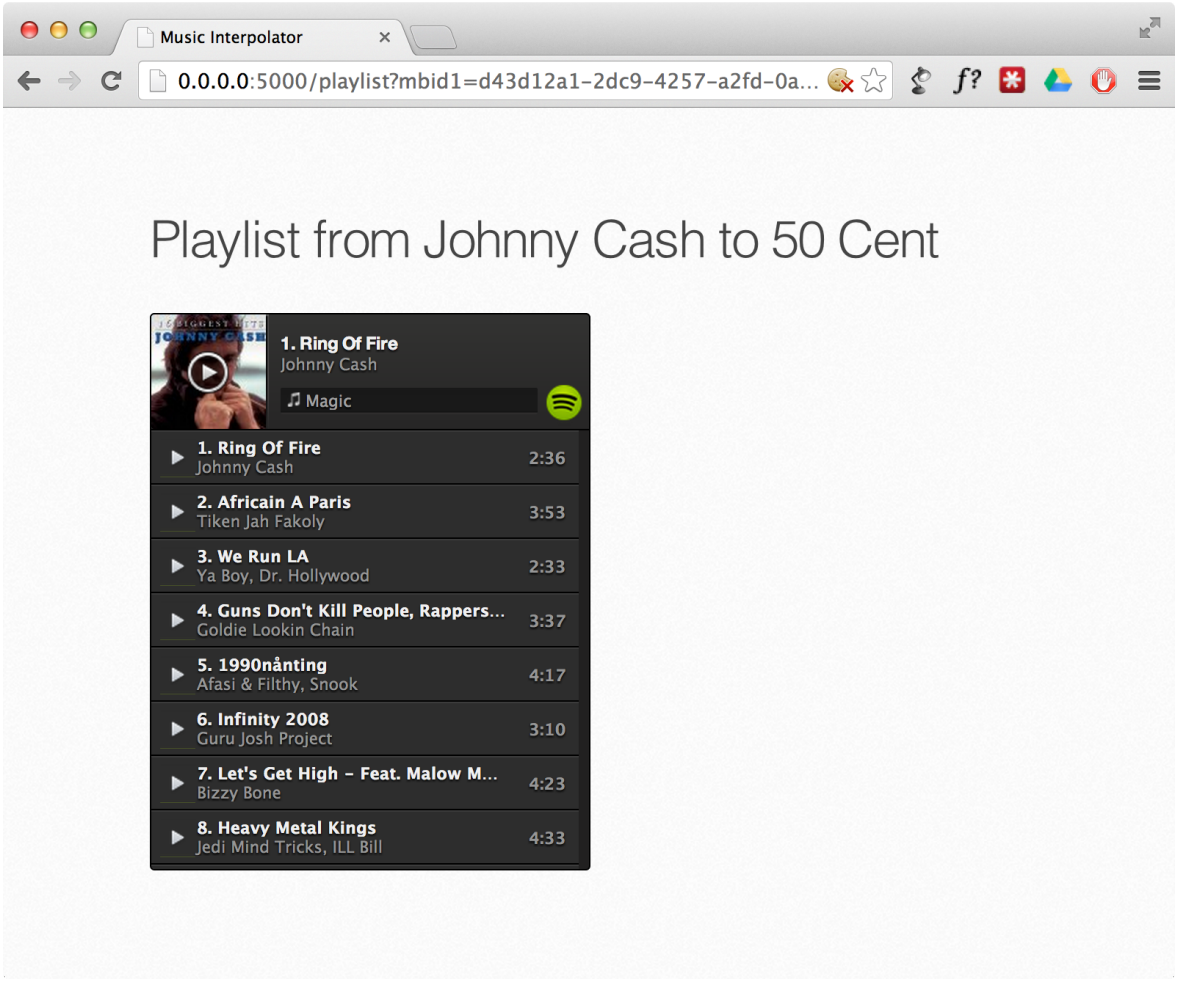

Figure 16: A screen shot of our prototype web application. This page shows the embedded Spotify music player with generated playlist.