Playlist Interpolation With Double Clustering Song Network (aka dj.NET: live version at `djnet.herokuapp.com/create`)

CS224W Project Final Report

SUNet IDs: jfhannel, atran3
Name: Jordan Hannel, Alex Duc Tran

Motivation

Our goal is a practical one. Creating a good playlist is a serious challenge and can consume a lot of time. Our project - dj.NET - can whip you up a playlist in no time, just supply the first song, the last song, and your desired playlist length. The generated playlist aims to maximize the "smoothness" of song transitions.

Problem Definition

Given a start song, end song and playlist duration, we must create a playlist (reasonably quickly) such that adjacent songs have a natural progression. Our search space for a sequence of n songs (out of m total available songs) is $O(m^n)$, and even this assumes that we are given the number of songs in the playlist. We have access to hundreds of thousands of songs, so searching over the entire space is infeasible for even small n. Our solution utilizes a static network (where nodes are songs) that we construct ahead of time that can be queried efficiently to yield a playlist between any two songs.

More formally, let N be a set of nodes (tracks) and let $d(n_1, n_2)$ represent the dissimilarity of two songs (we will also refer to this metric as the distance between two songs). Given a start track s, a destination track t, and the desired time interval between them dt, we will execute an algorithm over a network of tracks that aims to return a sequence of nodes N' that minimizes $\sum_{i=1}^{|N'|-1} d(N'_i, N'_{i+1})$ such that the total duration of N' is reasonably close to dt. In order to fulfill these kinds of queries efficiently and accurately, the network needs certain properties. Firstly, there must be a fairly short path between any pair of nodes (start and end nodes can be any pair, and no one wants a 1500 song playlist). Second, there must be a fairly short smooth path between any pair of nodes (smoothness is according to our distance function). Third, these paths must be efficiently computable (in a matter of seconds). This task is very challenging, and our solution hinges on three major design decisions. Firstly, and most importantly, is the Double Clustering model we use to construct the network that we query on. Second are the distance functions we use to parameterize the model - in order to reason about distances between songs, we characterize songs in a multi-dimensional space based on acoustic features and metadata from the EchoNest music database. The last design decision we made is what distance function to use for our decentralized search.

Prior Work

A (simplified) variation of this problem was attempted in this class last year [1]. The initial goal of their project was to create a playlist of minimum length which smoothly connects two songs or lists of songs. However the problem was quickly reduced to smoothly connecting two artists. The general format is a network where nodes are artists and an edge between two artists exists with probability as a function that increases with the similarity of the two artists. The algorithm for connecting artists is simply to start at the starting artist, move to the neighbor whose distance to the target artist is least, and repeat the process until the target is reached, a dead node is reached, or the path reaches a threshold length. This project from last year sets up the motivation for our work. It attempts to create a solution for the same problem area that we are solving and serves as a baseline model and algorithm. However, this project did not realize very much of its potential. Firstly, it immediately discarded its initial goal of connecting songs and ignored much of the available data on the song level. Secondly, its probabilistic model for edge generation was not very well justified, nor was the function which determined the probabilities. Lastly, there was no effort to theoretically assert the navigability of this probabilistic network and justify the validity of using decentralized search. We seek to establish finer granularity in our graph by using track nodes as opposed to artist nodes and look at the problem in a much more complex

way. This granularity will help us take advantage of the significant song level data in EchoNest and handle the variance in style for a single artist.

In terms of our song network, as opposed to the previous probabilistic model, we need a graphical model that maintains a low average shortest path between nodes while still being efficiently computable, i.e. low degree nodes to limit search space. However, when limiting the degree of nodes, we have to be careful not to affect the navigability of the graph, especially in terms of smoothness. Thus, our desired model must do a good job of balancing this need for efficiency and smooth navigability. While researching graph models that might have the properties we need, we found a brilliant paper [2] by Oskar Sandberg entitled "The Structure and Dynamics of Navigable Networks". In his generous 134 pages of wisdom, Sandberg discusses random rewiring, Milgram's Small World, percolation, and most importantly, double clustering. We provide a comprehensive discussion of double clustering in our Model and Algorithm section, but the intuition is that edges are created between nodes that are similar in at least one of two different geometric feature spaces. Sandberg concludes that graphs created by double clustering are navigable in both feature spaces (meaning that short paths can be found between pairs of nodes using a greedy search where distance is judged in either feature space). Furthermore, since edges are designed to connect nodes which are similar in one or more of the feature spaces, we achieve by default the property that adjacent nodes in paths will be similar in terms of those feature spaces. So all we need to do is determine what a human judges to be a smooth transition.

As far as music similarity metrics, there are many options as discussed by Berenzweig et al. [3]: symbolic representations, acoustic properties, subjective information, etc. In their work, they determine that acoustic properties are generally adequate in determining music similarity, yet the acoustic properties they use are very primitive in scope. That is, the properties they utilize fail to capture information about the temporal structure of a song and rely mainly on characteristics seen in small snippets of a song. However, we can do better in terms of acoustic properties. By taking advantage of more comprehensive acoustic properties, it is possible to make very reliable and accurate music similarity calculations. Fortunately, Echo Nest surfaces such acoustic properties as far as general song characteristics and overall temporal structure, as well as non-acoustic properties such as artist, release date, popularity, etc.

Dataset and Data Collection

When we originally searched for data, the most notable sources were MusicBrainz, Echo Nest, last.fm and the Music Genome Project. Our highest priority is to have enough information on individual tracks that we can create a music feature space that is accurate and precise enough to navigate between songs. MusicBrainz seems to have a lot of metadata on a lot of tracks but fell short on features associated with actual song content, and Last.fm is similar. Music Genome Project seems to have by far the best database, but it is proprietary (Pandora). Echo Nest provides nice acoustic attributes for song content as well as a wealth of metadata about tracks, artists, genres and the like, but the data is only available via API calls. As the Echo Nest data best accommodates our need, we decided to utilize it regardless of the API with rate limiting. Luckily, there are enough methods for batch data retrieval that it doesn't take too long to gather a moderate initial dataset. For our initial dataset, we gather up to 100 songs (depending if they had 100 songs listed) from each of the top 1,000 artists on EchoNest. This collection gives information on 77,528 songs total, which we have cached for later use.

The information we use for each track falls into two categories: attributes of the song itself, and song metadata. In the first category we have acousticness, danceability, energy, liveness, loudness, speechiness, tempo, valence, time signature, instrumentalness, key and mode (major/minor). In the second category we have song hotness, song currency, artist hotness, artist familiarity, artist, and song type (multiple sub genres that the song belongs to). Note that these attributes were generated by Echo Nest and are subjective, but as they are standard across all tracks we believe they provide a reasonable feature space.

Model and Algorithm

To reiterate, our goal is to design a graph such that we can find a path between any two nodes using only local graph knowledge that minimizes

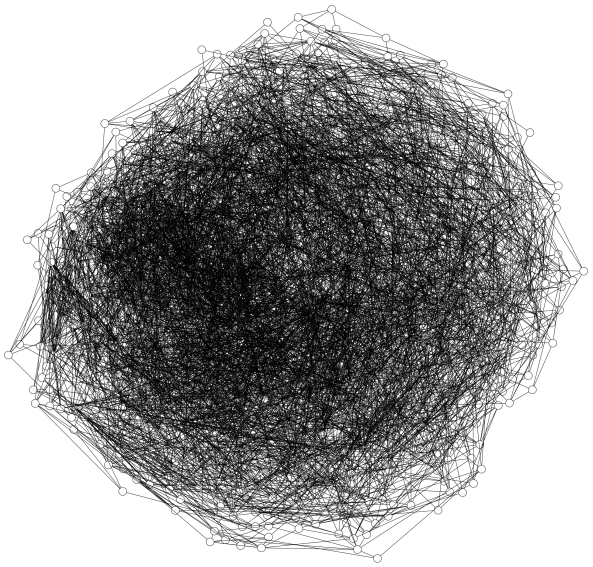
distance between adjacent nodes in the path (for our definition of distance - track dissimilarity). Recall that the defining characteristic of any Small World graph is that there is a reasonably short path between any pair of nodes that can be traversed without global knowledge of the graph. One example of a Small World graph we saw in class is a ring graph with long range random edges resulting from probabilistic rewiring. This graph has low diameter and is navigable, however there is no guarantee that the paths will have any merit with respect to smoothness, because long range links are created randomly (might transition from Beethoven to Miley Cyrus!). Another more structured example of a Small World network is the hierarchical network - it is navigable, has low diameter, and to some extent it has merit with respect to smoothness in that a path between two nodes will go through nodes which belong to the subtree of the two nodes' lowest common ancestor. If we design the hierarchy to reflect families of music, then this property will enforce some amount of smoothness on paths. However, the hierarchical model still falls short in that it only models distance as tree distance, and for node s, the presence of an edge to any node t depends only on this distance, and it is randomized - transitioning based on genre is not the granular interpretation that we are seeking. A complex domain such as music requires a more complex model of distance, and does not require randomization.

Double Clustering

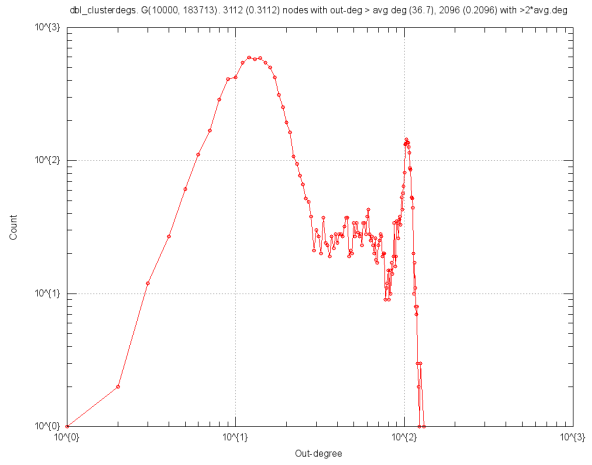
The graph model we are looking for must have low diameter, it must be navigable, and it should allow for a complex model of node distance. Our chosen solution is the Double Clustering model, presented by Oskar Sandberg [4]. The model was originally meant to model the social network Small World phenomenon - a graph of people was created where edges correspond to either geographical proximity of two people OR "common interest" of two people; their non-geographic similarity. The intuition is that there are two models for node distance, and edges exist between nodes that are similar with respect to either model. To be precise, a vertex x will have an edge to another vertex y if there is no third vertex which is closer to x than y in both spaces. So, in Sandberg's example, Jordan will have an edge to Alex if and only if there is no other person who both lives closer to Jordan than Alex AND has more similar interests than Alex. Note that this definition is asymmetric (i.e. the graph G is directed). Formally, with distance functions d_1 and d_2 respectively. The graph $G = (V, E)$ is constructed as follows:

$V = 1, 2, ..., n$
$(i, j)\ \epsilon\ E$ if $\forall\ k\ \epsilon\ V,\ \ k\ \neq\ i,\ j$:
$\quad d_1(x_i, x_k) < d_1(x_i, x_j) \rightarrow d_2(y_i, y_k) \geq d_2(y_i, y_j)$

Sandberg created the Double Clustering model to test a hypothesis that if people make connections based on two separate sets of criteria, the Small World phenomenon will arise. And, in fact, this hypothesis was validated: Sandberg showed both in practice and in theory, the Double Clustering graph is navigable (path lengths O(logn) using decentralized search) with respect to either distance function! Our purpose is a corollary of Sandberg's in a way: we hope to achieve Small World phenomenon (as well as smooth paths) by using two distance functions. Sandberg concerned himself than a more general case than we require and his proofs are rather involved, but navigability generally follows from the idea that halving the distance between two nodes can be done in a constant number of steps, resulting in O(logn) paths. The following is a visualization of our 1,000 node Double Clustering graph, and the degree distribution for the final 10,000 node version:



1,000 node Double Clustering song network



10,000 node Double Clustering song network degree distribution

Note that while the definition of a Double Clustering graph uses directed edges, we interpret edges as undirected for our application because song similarity is symmetric (Sandberg notes that his results hold for the undirected interpretation, it was just preferable to prove them in the directed setting). Our graph has very distinctive properties which we have not seen from examples in class. Aside from the breadth of degrees represented, the most notable characteristic of the degree distribution are the two distinct humps. We hypothesize that in each of our feature spaces there is a "middle ground" where most of the songs reside, and songs closer to this median will be closer to more songs, and will have more edges as a result. Songs that are close to the median in both feature spaces will have more edges still. We believe that the first hump represents songs in the median of only one feature space, while the second hump represents songs near the median of both feature spaces. It is clear from the visualization that the network is very well connected - in fact, our 1000 node version has diameter of 7 with average degree of 13.4, and our 10000 node version has diameter of 8 with average degree 36.7.

Song Characterization and Distance Functions

The hypothesis that validates this model in our case is that people don't consider a transition smooth based simply on songs sounding similar, but also based on whether the songs are from similar eras, by similar (or the same) artists, of the same genre, etc. This hypothesis seems to align with human preferences: you like certain sounds in general, but don't want a playlist to sound entirely homogenous; you like certain artists in general, but you don't want to be restricted to those artists. We have one distance function for the content of the songs, and another for their similarity in terms of metadata. For Song Distance, we take all the real-valued acoustic features (shown below) of each song, normalize them to the range [0,1], then take a weighted sum (weights shown below) over the absolute values of their differences. Finally, we incorporate a small bonus if the songs are in the same key/mode. For Metadata Distance, we take all the real-valued non-acoustic features (shown below) of each song, normalize them to the range [0,1], then take a weighted sum (weights shown below) over the absolute values of their differences. Finally we incorporate a bonus if the songs have the same artist or if they share genres.

Acoustic Features: acousticness: 2, danceability: 2, energy: 2, liveness: 2, loudness: 0.5, speechiness: 1.5, tempo: 1.5, valence: 1.0, instrumentalness: 1.5, time signature: 0.5

Non-Acoustic Features: artist hotness: 2, song hotness: 2, song currency: 1, artist familiarity: 1

Decentralized Search

Now that we have a Double Clustering graph, we need a method by which to get playlists from it in a matter of seconds, or less. From node s, we extend the path to the neighbor of s that minimizes some function of s, the neighbor, and the target node (final song). We have the freedom to choose this function such that our playlists are of desirable length and such that the transitions are smooth. Our first step is to evaluate the paths we find using either of our two Double Clustering distance functions (acoustic and metadata). We found that the graph is navigable (path lengths are mostly on the order of 5 to 50) using either distance function, or the sum of both distance functions, and even with a slight variation on that - our final solution chooses the neighbor that minimizes the sum of both distance functions minus the popularity of the neighbor song (this is only to make the people happy - popularity is weighted half as much as distance). Popularity is in the range [0,1] and is defined as the average of artist hotness, song hotness and artist familiarity. Amazingly the graph is navigable even using this function - a result that was not proven by Sandberg. This function will serve as our measure of smoothness and will henceforth be known as the Objective Function, and we will discuss further

how we arrived at this function in Results and Analysis.

Post Processing

Since the user can request playlists of variable length, there is no guarantee that the path we find will result in a playlist of appropriate duration. Most of the time, our playlist will be too long, but sometimes it falls short as well. When it is too long, we employ a greedy algorithm which evaluates the distance (according to Objective Function) between the 1st and 3rd songs, 2nd and 4th, etc, then eliminates the song whose neighbors are closest together. This process repeats until the playlist is of appropriate length. When the playlist is too short, we take each adjacent pair (a, b), find the mutual neighbor (c) that minimizes $f(a, c) + f(c, b)$ where f is the Objective Function. We then insert to the playlist one of these mutual neighbors that has the lowest $f(a, c) + f(c, b)$, and repeat this process until the playlist is of appropriate length. In a production version, to maximize performance we would ask users to supply multiple anchor songs throughout the playlist such that the given songs are separated by a moderate amount (so that we will not have to post process very much).

Results and Analysis

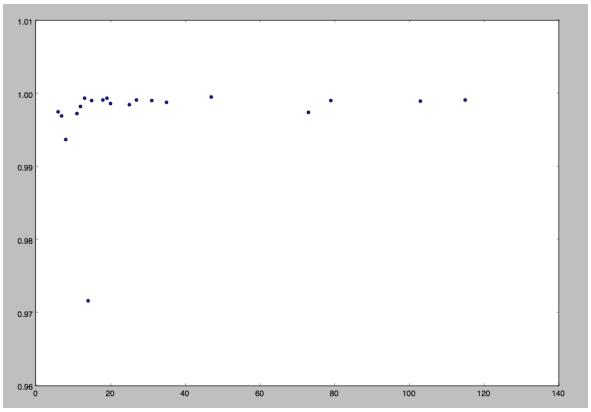
Evaluation Methodology

Evaluating what is a good playlist is an inherently subjective question, however feedback is necessary for iterative development so it was not practical to conduct a survey every time we wanted to evaluate the performance of our program. Our solution is to measure performance and optimize based on our Objective Function. This allows us to rapidly iterate, and also allows us to independently optimize for subjective criteria by changing our Objective Function based on survey results at another time. Evaluating our playlists based on Objective Function value itself is not ideal, because some songs are simply further from all other songs in total so the best that can be done is not as good. Our chosen evaluation metric is to determine the Transition Percentile for each transition in the playlist, then take the average of those. The transition percentile between song a and song b is the percentage of all songs c in the network for which $f(a, c) < f(a, b)$ where f is the Objective Function. A transition percentile of 1.0 means song b is the best song to follow song a, while a value of 0.0 means song b is the worst song to follow song a.

Selection of Objective Function and Final Results

At this point our graph has been created and will not change (1000 nodes, 6773 edges), and we have decided the format of our decentralized search, so all that is left is to choose an Objective Function with which to search and also to evaluate our playlists. Naturally, we started with our two distance functions (separately), simply to confirm that our graph is indeed navigable. The two relevant features of a returned playlist are the length and the average Transition Percentile - over a sample of 20 random playlists, lengths were in the range 10 to 50 with a couple anomalies in the hundreds when we used Acoustic difference, and in the range 5 to 10 when we used Metadata difference. TPs were typically in the range 0.94-0.98 with only marginal outliers in both cases. Clearly, the graph is navigable as the lengths are quite reasonable, and the TP's are quite high as well. In fact, the lengths are short enough that we should consider sacrificing some navigability for better TP's. We tried using the sum of both distances as our Objective Function, and we calculated the distance from neighbor to target as usual, but we also added the distance from current node to neighbor. In traditional decentralized search, we only care about the distance to the target, but since our objective is smoothness we also care about taking small steps to create a smooth trajectory. The results were good - typical path length was around 5 to 20 (and there were some concerning anomalies of very long paths) and TP increased to consistently over 0.98 and often above 0.99. We found that we could maintain our TP increases and eliminate the anomalous long paths by weighting the distance from current node to neighbor less. We also found that including popularity in our function did not adversely affect objective results and produced more likable overall playlists. Our final Objective Function is the sum of both distances from neighbor to target plus 0.5 times the sum of both distances from current node to neighbor minus the popularity of the neighbor. For the above tests, we used our 1000 node network, but after we were sure that we found our final solution we computed a 10000 node network (this took several

hours) and reran our test. The results are quite impressive: below is a scatter plot with length on the x axis and average Transition Percentile on the y. Each dot represents a generated playlist between two randomly selected songs.

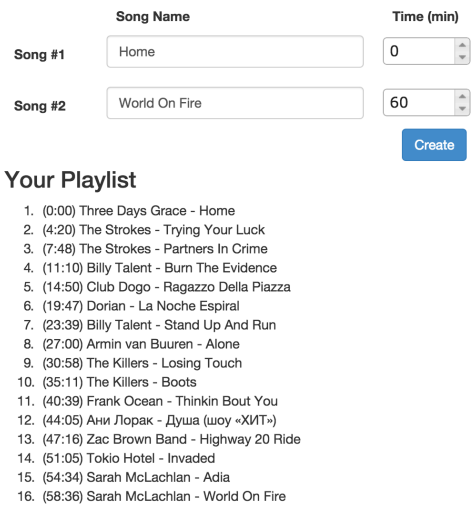


Playlist length vs. average Transition Percentile

While our methods are well supported theoretically and seemed to be working well throughout the process, we are in awe of these results. Playlists between random songs that may not be similar in the slightest, in a 10000 song network have lengths that are reasonably distributed from 10 to 100 or so songs, and whats exceedingly impressive - their average Transition Percentiles are extremely close to perfect. In fact, 18 of the 20 playlists had average TP above 0.997 - this means that for a typical song in each playlist, out of 10000 songs, only 30 songs would have made for a better transition. With such remarkable performance relative to our own distance metrics, the only way to optimize in the future is to tune our distance metrics to fit subjective tastes.

Sample Playlists/Demo

We have created a live demo of our program for your enjoyment and evaluation, and for the sake of completeness of our report, we've included demo screenshots including sample playlists:



Difficulties and Future Work

Firstly, the computation required to create a double clustering graph presents a challenge. One of the most significant expenses is computing the distance between every distinct pair of nodes for each of the two distance functions. This is $O(n^2)$, but has a high constant factor. The other significant component is deciding each edge - we must iterate over every node pair (in both directions) and for each of those pairs iterate over every other node to see if it is closer by both distance measures. This is $O(n^3)$, but with a very low constant factor due to caching. Even with as much caching and optimizations as possible, creating a double clustering graph on 10,000 songs takes on the order of an hour on a single machine. Therefore, creating our model for a production system with 1 million songs (same as Pandora) would take around 250,000 minutes (on my MacBook Pro). However, the process is highly parallelizable and only needs to be done once, so it would be plausible to compute such a large graph using many cores over a day or so. Lastly, we have a couple of practical needs to meet since this is after all a very practical project in terms of scope and goal. The implementation for the demo is a simple web server that runs queries on our graph and puts the returned songs into a playlist using the Spotify API. Further, we hope to implement a method of dynamically updating our graph for each query made. As discussed in the Sandberg's thesis and the Clauset and Moore paper [4], rewiring is a quick and simple way to account for previous queries. Ideally, we will utilize some rewiring strategy that dynamically optimizes our graph by introducing smoothness into the more

distant areas of the graph as revealed by queries. In a production version we would also need to maintain the graph as Echo Nest changes (distance may change with time, new songs will be added, etc). Additionally, since our model works so well relative to its own distance metrics, it might produce incredible results if we customized distance metrics to each individual. This could be done by showing a user a song, and asking which of a pair of other songs the user would rather hear next - we would then apply a machine learning algorithm to arrive at a distance function.

Reflection

We found this project very engaging and satisfying because it revolved around a problem that is familiar and meaningful to us and it presented challenges that are certainly complex but not so complex as to prevent progress of any kind. However one aspect of our experience that I would like to highlight to the course staff is that our efforts involved studying network properties in order to create a network that achieves a certain goal, while the common perspective in class is to either find networks that describe real life phenomenon, or analyze networks derived from real life phenomenon. Our efforts were more on the engineering side as opposed to science, and of course both are valid, but we believe it might please students if the CS224W course included some material about how to create networks that serve a given purpose.

References

[1] Embiricos, Alexander, Salman Quazi, and Prasanth Veerina. Hip-Hop to Deep House: Navigating the Music Graph Using Decentralized Search. SNAP - Stanford.edu. Stanford.edu, n.d. Web. $< http://snap.stanford.edu/class/cs224w-2013/projects2013/cs224w-037-final.pdf >$.

[2] Sandberg, Oskar. The structure and dynamics of navigable networks. Division of Mathematical Statistics, Department of Mathematical Sciences, Chalmers University of Technology and Gteborg University, 2007

[3] Berenzweig, Adam, Beth Logan, Daniel P.W. Ellis, and Brian Whitman. A Large-Scale Evaluation of Acoustic and Subjective Music-Similarity Measures. MIT Press Journals. MIT, n.d. Web. $< http://www.mitpressjournals.org/doi/pdf/10.1162/014$

[4] Clauset, Aaron, and Christopher Moore. How Do Networks Become Navigable? SNAP - Stanford.edu. Stanford, n.d. Web. $< http://snap.stanford.edu/class/cs224w-readings/clauset03navigable.pdf >$.