International Journal of Music Science, Technology and Art
(IJMSTA) 1 (2): 71-80, Music Academy "Studio Musica", 2019 ISSN: 2612-2146 (Online)

A Smart Shuffle approach to Playlist Shuffling with User Defined Constraints

S. Ramroach[1, ✉], P. Hosein[2]

[1]Department of Electrical and Computer Engineering, The University of the West Indies, St. Augustine, Trinidad and Tobago.
[2]Department of Computing and Information Technology, The University of the West Indies, St. Augustine, Trinidad and Tobago.
✉ Corresponding author: sramroach@gmail.com

ARTICLE INFO

Received: June 23, 2019
Accepted: August 21, 2019
Published: September 1, 2019

Keywords:
Playlist shuffling
Music sequencing
Optimization
Data Analytics

ABSTRACT

We consider the problem of track sequencing for a given music playlist. We assume that a user chooses a set of desirable songs to form a playlist as would be done in applications such as iTunes or Google Play Music. However, instead of using the typical random shuffle feature, we introduce what we call a smart shuffle option in which the user specifies various constraints that must be satisfied when determining the playback sequence. These constraints are based on several attributes of the songs. If the user does not provide any constraints, all attributes are considered equal. The general computational problem is the Travelling Salesman Problem in Euclidean space. We consider the following approaches: three hierarchical clustering techniques, K-means clustering, a greedy swapping approach, and an approximation approach (Christofides' 3/2 -approximation). We then compare performances based on a defined performance metric. We also perform a subjective evaluation to ensure that the proposed model enhances the listening experience of a user.

1 Introduction

Music consumers have unlimited access to extensive music collections and pre-defined playlists. These playlists are usually generated and sorted by music producers or disc jockeys (DJs). The sorting process may place consecutive songs from the same genre or artist next to each other or may mix genres harshly. The variation and subjectivity of the outcome is a major drawback to the listening pleasure of a consumer. However, technological advances in the digitization of music have revolutionized the way in which users listen to music as they are now able to create personalized playlists. Services such as Spotify [1], iTunes [2], and Google Play Music [3], provide a platform for users to compile their playlists with a large selection of songs. These collections can be used for special events or as background music while studying. Previous researchers built tools to extract the meta-data from songs [4], [5], [6] and these were combined to form large datasets for public usage [7], [8]. Current research focuses on assisting the consumer with the generation of personalized lists [9], [10], [11]. The "mood" of these lists are often inputted by the consumer and the playlist is built via the meta-data of songs in the pool [12]. The problem addressed in this paper is the ordering of songs in a list that has already been created by the user, such that there is the smoothest gradient of "mood" or pleasantness from the first to the last song in the list. Current means include sorting randomly, alphabetically, on metrics such as number of hits, or using the data generated by asking the user to rate each song [13]. Naive ordering such as random or alphabetical has a low probability of producing a smooth change in pleasantness, and as playlists increase in size, it is not feasible for a user to rate every song. The presented solution removes the need for extensive user interaction by retrieving objective metrics about each song in the playlist and applying various techniques to achieve the smoothest or smartest ordering of songs.

2 Problem Definition

We consider the problem in which a user has chosen a set of songs *S* for a playlist and wishes to hear them all. Therefore we are not concerned with the problem of choosing songs for the playlist. Although the user likes all of the songs in the list, the playback order is also important. For example, the user may prefer to listen to a sequence of several songs from a single artist or genre before switching to another one. We, therefore, address the problem of finding a playback sequence in which all songs are played but where the sequence has the smoothest possible pleasantness gradient from beginning to end. No user input is required nor are labels or hit statistics used.

We assume that each song has the following meta-data: title, artist, and year released. If this meta-data is not available, it can be acquired online. We use Spotify's Web API to obtain attributes for each song. The attributes used are, acousticness, danceability, duration, energy, instrumentalness, key, liveness, loudness, mode, spechiness, tempo and valence (see [14] for descriptions of these). Our objective is to determine a playback sequence that minimizes the difference in attributes between consecutive songs. We wish to avoid sudden changes between consecutive songs. If we are listening to a high tempo song we wish to maintain a similar level of tempo in the next song but over time the tempo will gradually change. Therefore, we must determine a sequence of songs from the playlist that minimizes transitions between consecutive songs, across multiple attributes.

All attributes, except artist and year, have been normalized to values between 0 and 1. In the case of the artist, there are no good similarity metrics since the similarity between two artists tends to be subjective [11]. Therefore, we use a simple distance metric for artists called artist difference. If we have two songs by the same artist then the artist difference is 0, whereas if the artists are different then the difference is 1. This will help cluster songs by a single artist together. In the case of year of release, the distance metric between two songs is simply the absolute value of the difference in the year released, divided by 10. A decade is therefore represented by a value of 1. Using these distance definitions we define the distance between two songs as the sum of the square of the difference of each attribute. Note that in order to find the optimal solution one would have to evaluate all $|S|!$ possible solutions. Next, we provide various heuristics for this problem with significantly less computational complexities.

3 Heuristics

We investigated various approaches to the problem and evaluated both performance in terms of average distance between consecutive songs as well as computation time. We need to observe computation time since it is always possible to achieve optimal performance by doing an exhaustive search. In the following sections we assume that there are N songs in the playlist, which results in N - 1 song pairs.

3.1 Hierarchical Clustering

In this section, a clustering approach to the problem is described. First, the songs are divided based on a year range. For example, all 70s songs are put in one cluster, all 80s are put in another, etc. Within each of these clusters, sub-clusters are formed based on the artist of the songs. Such clusters are only formed if the number of songs by an artist exceeds some threshold. All artists that do not satisfy this criterion will have all of their songs included in one cluster. Next, sub-clusters are formed from within the artist cluster based on another attribute (e.g. tempo). A threshold is again used and if a subcluster is smaller in size than this threshold then the cluster will no longer be split. Otherwise, the process continues with a new attribute. The order in which attributes are chosen to determine the best hierarchy is varied.

Regarding the Spotify-derived attributes, a threshold of 0.5 is used to split a cluster (i.e., songs with a metric value of 0.5 or less are placed in one cluster while the others are placed in another). If the present cluster has at least some given number of songs, this process is repeated. Finally, the sequence of songs is obtained by traversing the components of each cluster going from left to right (or right to left). Note that the members of each cluster are similar and thus exhibit low differences. Adjacent clusters are also similar since the prior splits kept certain attributes close to each other. Hence we expect the final sequence to have small variations from song to song. Figure 1 illustrates an example of a Hierarchical Cluster. In this particular case the sequence of the playlist would be:

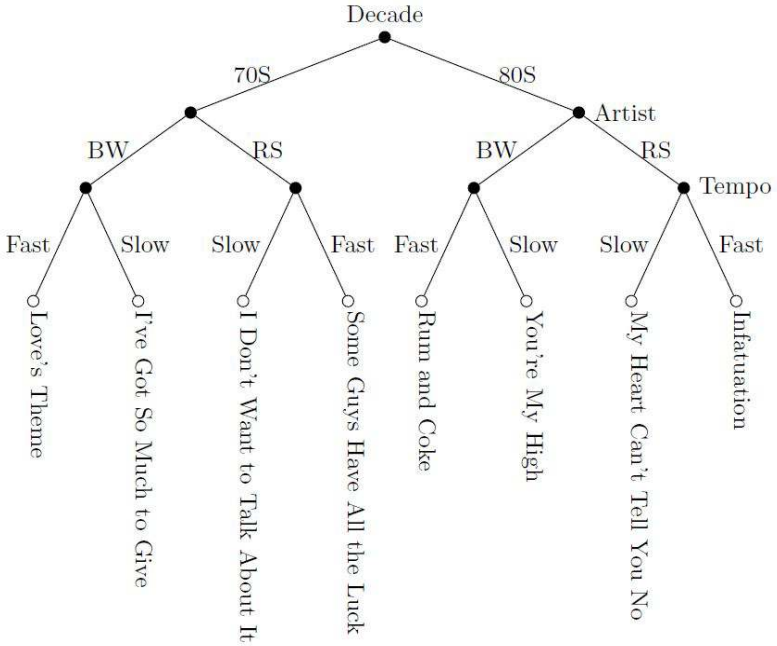

Figure 1. Example of hierarchical clustering.

Love's Theme
I've Got So Much to Give
I Don't Want to Talk About It
Some Guys Have All the Luck
Rum and Coke
You're My High
My Heart Can't Tell You No
Infatuation

Three types of hierarchical clustering are used: single linkage [15], complete linkage [16], and Ward's variance minimization method [17]. All methods use Euclidean distance as their distance function. Single linkage outputs its smartest shuffle using the following procedure. Initially, each of our *N* songs exists as its own cluster. Next, the two closest clusters are merged. This is done sequentially such that after *N* - 1 iterations, all *N* songs would exist within a single cluster. However, on step 2, a cluster would have 2 songs compared to all other clusters of 1 song. Now, we define the distance between two clusters *X* and *Y* as the smallest distance between any song $x \in X$ and any song $y \in Y$. Finally, the songs in the cluster can be read from left to right, or vice versa to output the smart shuffle playlist.

Complete linkage is the conceptual opposite of single linkage clustering. Clusters are combined by the farthest neighbour. Similar to single linkage, each of the N songs initially exists as its own cluster. Songs are also sequentially added to its closest cluster until one cluster encompasses all songs. However, the distance between two clusters *X* and *Y* is the largest distance between any song $x \in X$ and any song $y \in Y$. Finally, the songs in the cluster can be read from left to right, or vice versa to output the smart shuffle playlist. Single linkage tends to produce long clusters, as opposed to the tight compact clusters of complete linkage.

Ward's variance minimization method begins with each song existing as its own cluster. Larger clusters are formed sequentially until there is only one cluster of all *N* songs. However, at each step, the two clusters whose union results in the minimum increase in total within-cluster variance after merging, are combined. For example, if a cluster *a* is comprised of clusters *b* and *c*, and an unused cluster *d* is to be assessed for its suitability in a merger, the variance $v(a, d)$ is calculated as follows:

$$sq = \frac{|d| + |b|}{T} v(d,b)^2 + \frac{|d| + |c|}{T} v(d,c)^2 - \frac{|d|}{T} v(c,b)^2$$

$$v(a,d) = \sqrt{sq}$$

where $T = |b| + |c| + |d|$, and |*| is the cardinality of its argument. Variance minimization is maintained by choosing to merge with the cluster, which results in the smallest increase in variance.

3.2 K-Means Clustering

The standard K-means clustering algorithm using Euclidean distance between songs is considered. Once clusters are determined, the song sequence is obtained as follows. Starting with the largest cluster, all songs are added from the cluster (in any order) into the playlist. This process is repeated for the closest cluster (based on centroid distances) to the previously processed cluster. The process stops when all clusters are iterated. The parameter that must be determined is K. Note that when K is small the clusters are large and songs in these clusters are added randomly. Therefore, it is expected that as K increases, so to will performance. However, for larger values of K, computational complexity, and thus execution time, is also expected to be higher. All values of K within the range $1 \leq K \leq N$, where N is the number of songs, are considered.

3.3 Greedy Swapping Algorithm

In this random-optimization approach, the playlist is randomly generated and the mean Euclidean distance between all songs in the list is computed. Two songs are chosen at random and if by swapping them both, the mean Euclidean distance is reduced, the songs are swapped, else they are kept in their present positions. This process is repeated for a specified number of times or until the performance improvement per iteration falls below some threshold. There were two numbers of iterations used in the experiments: 10,000 iterations (which has a runtime close to those of the other algorithms) and 100,000 iterations (which resulted in a mean Euclidean distance similar to the other algorithms).

3.4 Lower Bound Determination

The traditional approach is a random shuffle and hence the expected performance of a random shuffle can serve as a lower bound. Since the number of possible sequences is large we instead obtain an experimental average. There may be some peculiarities with specific random shuffles, so to avoid this issue, several random sequences are generated and the average of the performance over these sequences are computed. Note that a lower bound in performance corresponds to an upper bound on the performance metric which is the mean Euclidean distance.

3.5 Upper Bound on Optimal Solution

The upper bound on performance is calculated as follows. Consider all song pairs and sort them by their Euclidean distance. The average of the smallest $N - 1$ pairs will form a lower bound on any sequence of the songs and hence can be used as an upper bound on performance. Note that these pairs of songs will typically not be a feasible playlist since a single song may occur in more than two pairs in the chosen list.
A well-known upper bound for the TSP is the Held-Karp algorithm which has the optimization property that every sub-path of a path of minimum distance is itself of minimum distance. This would typically provide a superior upper bound but the run-time is impractical for a playlist of more than 10 songs.

3.6 Christofides' Approximation Algorithm

Approximation algorithms are used to provide good results in a reasonable time. Christofides' algorithm [19] finds approximate solutions on instances where the distances are symmetric and obey the triangle inequality. This technique begins by building a minimum spanning tree from the playlist, followed by a minimum-weight matching algorithm on the set of songs which have an odd degree. After adding both graphs, a Euler cycle is created from the combined graph with shortcuts to avoid visited songs. This algorithm ensures that the result is at most 3/2 times the optimal solution. Each song is represented as a node and distances or the weight of the edges from each song to every other song is calculated using the pairwise Euclidean distance. Figure 2 illustrates an example of a playlist of 10 songs being represented as a graph before and after applying Christofides' algorithm.

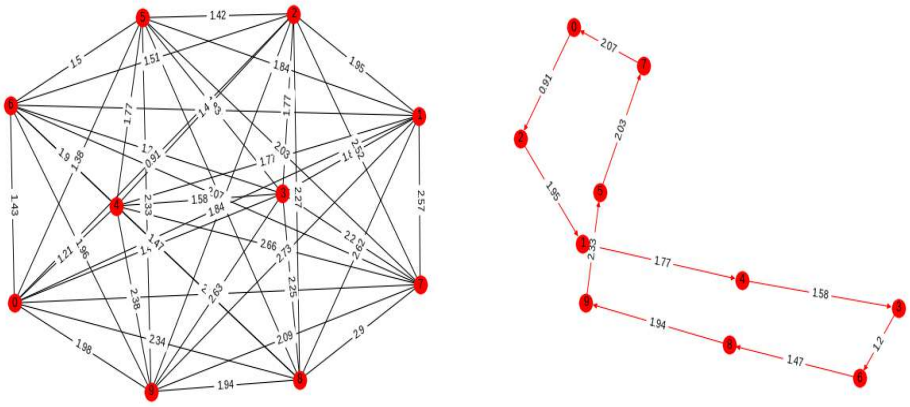

Figure 2. The initial graph and solution by Christofides' algorithm.

4 Numerical Results

4.1 Dataset and Features

The dataset was populated by creating a pool of the most popular songs from 1996 to 2017 and randomly selecting N songs. The resulting dataset contains N = 101 songs and information such as the title, artist, and year, are included. Spotify's API is also used to obtain the following attributes: acousticness, dance-ability, duration, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time signature, and valence.
All attributes, except year, artist, and title, were normalized to values between 0 and 1. The title of the songs played no role in any of the computation performed. Music is often categorized by decades and as such, the difference in a decade is reduced to a value of 1 (i.e., the year attribute is divided by 10). Finally, the pairwise Euclidean distance between the two songs was unchanged if both songs were created by the same artist. Otherwise, when calculating the pairwise Euclidean distance, a value of 1 is added. These assignments ensure that songs from the same decade and artist are given a higher chance of being grouped together in the final sequence. Due to the subjectivity of these decisions, provisions were made for user-defined preferences in a later section.

5 Discussion

The performance of all algorithms are compared via two metrics: the resulting mean Euclidean distance of the shuffled playlists and the time taken to create the sequence is also determined. The mean Euclidean distance $\overline{d}$ of a sequence of N songs is calculated by:

$$\overline{d} = \frac{1}{N-1}\sum_{i=1}^{i=N-1} d(i, i+1)$$

where $d(q, r)$ is the pairwise Euclidean distance between songs q and r. The pairwise Euclidean distance is calculated using the formula below:

$$d(q,r) = \sqrt{dot(q,q) - 2 \times dot(q,r) + dot(r,r)}$$

where $dot(a, b)$ is defined as:

$$dot(a,b)[i,j,k,m] = sum(a[i,j,:] \times b[k,:,m])$$

There are other methods for computing distance but this formulation has two advantages. The most relevant advantage is if one argument varies, then $dot(q, q)$ and $dot(r, r)$ can be pre-computed.

TABLE 1. PERFORMANCE AND RUN TIME COMPARISONS.

Algorithm	Mean Distance $\overline{d}$	Run time (s)
Christofides' Algorithm	0.98	5.54
Single Linkage	1.31	2.37
Complete Linkage	1.24	2.40
Ward's Method	1.24	2.36
K-Means (K = 16)	1.32	2.18
K-Means (K = N)	1.14	2.08
Greedy Swap (10,000)	1.25	2.22
Greedy Swap (100,000)	1.22	3.30
Lower Bound	1.75	2.05
Upper Bound	0.66	2.08

The other advantage is its computational efficiency when handling sparse data. The equations above were included in scikit-learn libraries [18], which were used in these experiments.

Table 1 presents performance and run time results for the various heuristics as well as the bounds. Christofides' algorithm attained the best playback sequence with a $\overline{d}$ of 0.98. The one drawback to this approach is the length of time taken to produce the output. As the size of the playlist increases, so too will the time taken to produce the most pleasant playback sequence.

Single linkage is the worst performing hierarchical cluster, indicating that the formation of longer clusters is not optimal for the given task. Complete linkage ensures clusters are tighter and this method tied Ward's variance minimization method in terms of $\overline{d}$. Ward's method is the fastest of the hierarchical techniques, indicating that the calculation of variance between clusters is faster than both finding the smallest distance between specific songs in each cluster (Single linkage) and finding the lower bound of largest distances between songs of all clusters (Complete linkage).

The K-Means algorithm was tested with a range of clusters $1 \leq K \leq N$. K-Means with $K = 16$ outperformed all other values of K, except for $K = N$. However, $K = 16$ had the worst $\overline{d}$ compared to all other algorithms. An arbitrary number of clusters does not produce an objectively smooth sequence. Whereas, the second closest result to the upper bound was achieved by $K = N$.

The $\overline{d}$ of 1.14 significantly outperformed all algorithms other than Christofides' approximation. The performance of the algorithm using this value of K indicates that a pleasant sequence is achieved when each song is treated as its own cluster and the sequence is built from choosing the song closest to the current song as the subsequent song in the list.

One peculiarity noticed is the time taken for this method to produce a sequence. $K = N$ was expected to have the longest execution time out of all other values of K, but instead, it achieved the best time. The generation of a sequence using a value of N for the number of clusters would entail comparing the distance of the first song to every other song (N - 1) in the list in order to choose the song with the smallest d. Each subsequent song would then repeat this process for all remaining songs in the list until the sequence is completed. Clearly, the results show that this tedious process is faster than the computation of the centroid of each new cluster for values of K less than N. The processing of a sequence for these values of K requires the formation of K clusters. The clusters are formed by joining the two existing clusters with the smallest Euclidean distance. This Euclidean distance is the distance between the centroids of each cluster. Therefore, at each step, the centroid needs to be computed and it is this computation, which accounts for the delayed output of a sequence.

It is evident that as the number of greedy swaps increases, the $\overline{d}$ will improve. This technique will result in the closest $\overline{d}$ to the upper bound if given a high enough number of swaps. However, it is impractical to use a sufficiently large number as its efficiency would be akin to that of brute force. Two values for the number of swaps were used in the experiments. A value of 10,000 was chosen due to a similar running time to the other algorithms, but the mean distance computed was worse than most of them. The second value of 100,000 was chosen to match the $\overline{d}$ of the competing algorithms. The main drawback of this number of swaps is the length of time taken to generate a sequence. This technique expectedly achieved the third best $\overline{d}$, only beaten by Christofides' approximation and K-Means with $K = N$.

Figure 3 illustrates the pairwise Euclidean distances between all songs in sequences produced by a random shuffle and the best shuffle. The difference between songs is much higher in the random shuffle. Although both plots depict spikes throughout their respective sequences, it can be observed that the spikes in the plot of the best shuffle are significantly lower than that of the random shuffle.

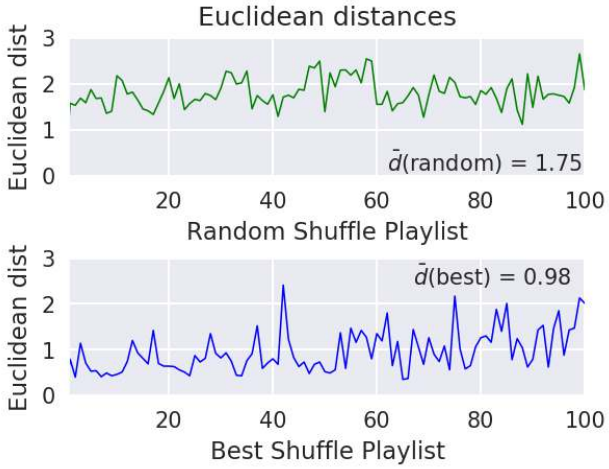

Figure 3. Pairwise Euclidean distances of random versus best sequences.

6 User Defined Preferences

In the above sections, it was assumed that the user did not provide any preferences (and so default values were used). In this section, we consider the case where users can adjust the weights for the various features. A song *s* has attributes acousticness, danceability, ..., time signature, and valence. Each of these attributes is multiplied by their respective weights: $w_{\text{acousticness}}$, …$w_{\text{timesignature}}$, and w_{valence}, where $0 \leq w \leq 1$.

We consider two examples. The user *U1* may prefer to place a weight of 0 to liveness, duration, and loudness (i.e., $w_{\text{liveness}} = w_{\text{duration}} = w_{\text{loudness}} = 0$), because these features do not seem relevant to the progression of the pleasantness of a playlist. Another user *U2* may agree with *U1*'s judgment of which attributes seem irrelevant, but can be more interested in a shuffle where the "energy" of the songs are given preference over the other attributes and thus use a weight of 0.9 for this attribute (i.e., $w_{\text{energy}} = 0.9$). In both cases, the weights of all other attributes are by default, set to 0.5.

We use the best algorithm obtained above (Christofides' algorithm) to illustrate how adjustments to the weights of the various features influence the computed sequence. Note that if an application is developed for this algorithm then these adjustments can be made via simple sliders. Table 2 shows the results of all algorithms on *U1*'s preferences. It is expected that the $\overline{d}$ for all algorithms would be less than that reported in Table 1 due to the reduction in the distance since the contribution of those attributes are now zero. Christofides' algorithm maintained the best performance, with a significantly smaller $\overline{d}$ than all other techniques but with the worst execution time. The simplicity of *U1*'s preferences results in essentially the deletion of three attributes from the dataset.

TABLE 2. COMPARISON OF USER-DEFINED ATTRIBUTES.

Algorithm	Mean Distance $\overline{d}$	Run time (s)
Christofides' Algorithm	0.87	3.76
Single Linkage	1.27	2.23
Complete Linkage	1.19	2.23
Ward's Method	1.17	2.26
K-Means (K = 15)	1.27	2.18
K-Means (K = N)	1.04	2.01
Greedy Swap (10,000)	1.19	2.13
Greedy Swap (100,000)	1.14	3.30
Lower Bound	1.67	1.97
Upper Bound	0.59	2.04

Some complexity is introduced when the sequences are generated for *U2*. *U2* values a smooth progression of energy as opposed to all other attributes. The following relationships for $\overline{d}_{\text{energy}}$ and $\overline{d}$ are expected:

$$\overline{d}_{\text{energy}}(\text{weighted}) \leq \overline{d}_{\text{energy}}(\text{best}) \leq \overline{d}_{\text{energy}}(\text{random})$$

$$\overline{d}(\text{best}) \leq \overline{d}(\text{random}) \leq \overline{d}(\text{weighted})$$

Figure 4 illustrates the mean Euclidean distance of the energy attribute $\overline{d}_{\text{energy}}$ across three sequences: weighted ($w_{\text{energy}} = 0.9$), random, and best (with $w_{\text{energy}} = 0.5$). The weighted shuffle playlist assigns the specified weights to all attributes and uses Christofides' algorithm to generate the final sequence. The $\overline{d}_{\text{energy}}$ of the weighted shuffle playlist is 0.15. The best shuffle creates a sequence with $\overline{d}_{\text{energy}}$ equal to 0.18. The random shuffle results in a $\overline{d}_{\text{energy}}$ of 0.24. These results support the expected relationship. This illustrates that one can use weights to amplify the importance of certain attributes over others. The plot fluctuates closer to 0 in the weighted and best shuffles, as opposed to wild fluctuations in the random shuffle.

Although *U2* wants a sequence in which energy is prioritized, the mean Euclidean distances must still be considered. The $\overline{d}$ of the three sequences are as follows: the $\overline{d}$ of the best sequence is 1.14. The $\overline{d}$ of the sequence after the weights were applied is 1.41, and finally, the $\overline{d}$ of the random sequence is 1.75. These results support the expected relationship described earlier. These results show that although preference is given to the energy attribute in a weighted shuffle, the mean Euclidean distance of all attributes is still lower than that of the random shuffle. This is illustrated in Figure 5. The progression of the energy attribute is smoothest in the weighted shuffle, but the overall progression of all attributes is smoothest in the best shuffle. A user can manipulate a wide array of variables as input preferences.

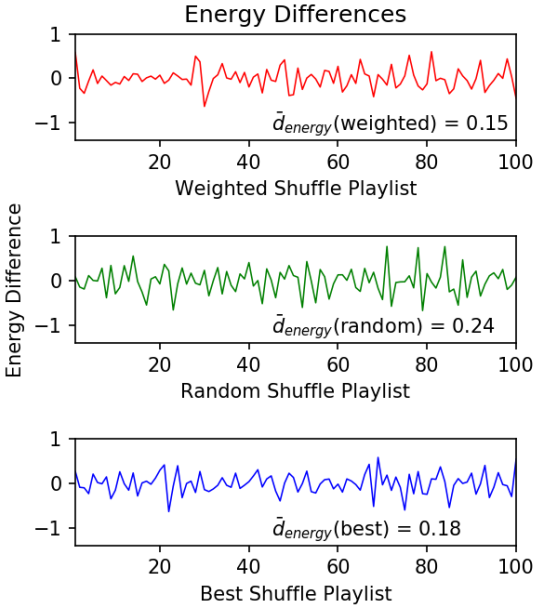

Figure 4. Pairwise Euclidean distances of the energy attribute.

The results show that objectively, the output sequence delivers what the user expects. Next, we investigate whether these results also hold subjectively.

7 Subjective testing

In this section, we perform subjective testing to determine the effectiveness of the approach outlined in previous sections. We randomly selected 10 songs, produced two sequences from this pool of 10 songs: a random sequence and a sequence generated by Christofides' algorithm. These two sequences were provided to subjects (in random order) and subjects were asked to rank the smoothness of the pleasantness gradient from the start to the end of the sequence, on a scale from 1 to 10. A score of 1 indicated poor ordering of songs and 10 indicated a near-perfect order. This was repeated for five pools of 10 song pairs of random/best shuffles for the subject, for multiple subjects. Each subject rated 10 playlists.

Table 3 summarizes the results obtained from subjective testing. All subjects gave the random shuffle a significantly worse score than the best shuffle. Objectively, the difference in $\overline{d}$ for random versus best shuffles for all sequences are within the range $0.2 \leq \Delta\overline{d} \leq 0.4$. There was a decrease in $\overline{d}$ of approximately 15% after applying the best shuffle to the sequence. This difference had a significant effect on the song progression reported by the subjects. The average score received by the random shuffles is 2.88. Whereas, the average score of the best shuffles is 7.32. In all cases, the rating of the best shuffle outshines that of the random shuffle. Even though a large portion of the population would need to be tested before any subjective test can claim to be conclusive, these preliminary results show that the best shuffle of a playlist

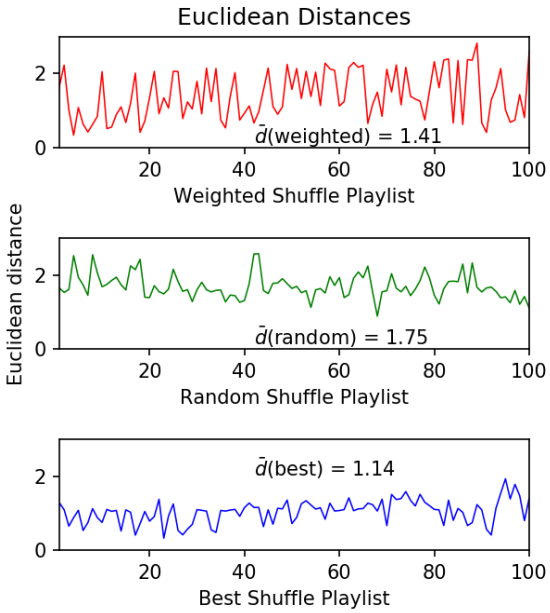

Figure 5. Pairwise Euclidean distances of all attributes.

subjectively produces a better gradient of pleasantness from the start to the end of the sequence than a random shuffle. These subjective results also support the objective statistics reported earlier in this paper.

TABLE 3. COMPARISON OF USER-DEFINED ATTRIBUTES.

Playlist	Shuffle	$\overline{d}$	Mean Score
A	Random	2.1	2.7
A	Best	1.8	7.5
B	Random	1.9	3.7
B	Best	1.6	6.0
C	Random	2.2	3.7
C	Best	1.8	7.7
D	Random	1.9	1.7
D	Best	1.7	8.0
E	Random	2.0	2.6
E	Best	1.7	7.4

The best shuffle of playlist C had $\overline{d}$ = 1.8, as opposed to the 1.9 of the random shuffle for playlist D, the subjective scores of both playlists differ by 6.0. This indicates that a random shuffle with a $\overline{d}$ close to the best shuffle of another list is still vastly inferior to the best shuffle. $\overline{d}$ as a standalone indicator of the quality of a sequence may only be relevant for objective analysis. This can be further supported by observing the $\overline{d}$ of the best shuffles for playlists A and B. Although the best shuffle for A resulted in $\overline{d}$ = 1.8 compared to the 1.6 of B, playlist A was scored 7.5 as opposed to the 6.0 of B. We expected the subjective scores to be inversely proportional to $\overline{d}$, but the results proved otherwise. The best shuffle for D had a $\overline{d}$ of 1.7 with a score of 8.0. This shows that the subjective scores behaved in a random fashion and although there isn't a linear relationship with d, it is clear that there is a sharp distinction between a random shuffle and the best shuffle.

8 Conclusions

Christofides' algorithm significantly outperformed other techniques in generating the smartest shuffle for songs from a playlist such that pleasantness is optimized via a reduction in pairwise Euclidean distance but at the cost of long execution times. It achieved a mean Euclidean distance of 0.98 for the playlist. This approximation algorithm was followed by K-means with $K = N$, and Ward's variance minimization hierarchical clustering. Since playlists can be on the order of 1000 songs or more, we will investigate if the solution provided by the K-means clustering algorithm is sufficient for our purposes because of its much better runtime. Complete linkage and Ward's variance minimization hierarchical clustering methods performed better than single linkage, with mean Euclidean distances of 1.24, 1.24, and 1.31, respectively. This indicated that tighter clusters produced a better sequence than longer clusters. Greedily swapping songs which improve the mean Euclidean distance could have had the

best performance. However, for this technique to be effective, a sufficiently large number of swaps would need to be considered and this tends to match the efficiency of a brute force approach. Using 10,000 and 100,000 swaps, this technique scored a mean Euclidean distance of 1.25 and 1.22, respectively. The time taken to achieve comparable results to the other techniques was more than 50% longer.

The best technique was used to shuffle five playlists of 10 songs each, for subjective testing. These tests indicate that a human can distinctly differentiate the best shuffle from a random shuffle. The average score of these shuffles were 7.32 and 2.88, respectively. The subjective results show that the mean Euclidean distance can be a viable measure of pleasantness.

The next step for this work is to build a mobile application to provide this functionality. The application would allow the user to import their playlists from various sources and it would provide the optimal or smartest shuffle of the songs for the smoothest gradient. Another important feature would be to allow the user to enter constraints, which would affect the smart shuffle. We have shown that with user constraints, our smart shuffle creates objectively better sequences than a random shuffle. Additional subjective testing would also be performed to determine which attributes are important to a user and so should be included in the application.

References

[1] Spotify. (2018) Spotify. [Online]. Available: https://www.spotify.com/us/

[2] Apple. (2018) itunes. [Online]. Available: https://www.apple.com/lae/itunes/

[3] Google. (2018) Google play music. [Online]. Available: https://play.google.com/music/listen

[4] D. Moffat, D. Ronan, J.D. Reiss, (2015). *An evaluation of audio feature extraction toolboxes*. In Proc. 18th International Conference on Digital Audio Effects (DAFx-15). DAFx-15, November 2015.

[5] A. Allik, G. Fazekas, M. B. Sandler, (2016). *An ontology for audio features*. In ISMIR, 2016, pp. 73–79.

[6] D. Mitrovi´c, M. Zeppelzauer, C. Breiteneder, (2010). *Features for content based audio retrieval*. In Advances in computers. Elsevier, 2010, vol. 78, pp. 71–150.

[7] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, P. Lamere, (2011). *The million song dataset*. In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011), 2011.

[8] K. Benzi, M. Defferrard, P. Vandergheynst, X. Bresson, (2016). *FMA: A dataset for music analysis*. CoRR, vol. abs/1612.01840, 2016. [Online]. Available: http://arxiv.org/abs/1612.01840

[9] G. Bonnin, D. Jannach, (2014). *Automated generation of music playlists: Survey and experiments*. ACM Comput. Surv., vol. 47, no. 2, pp. 26:1–26:35, Nov. 2014. [Online]. Available: http://doi.acm.org/10.1145/2652481

[10] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, O. Nieto, (2015). *Librosa: Audio and music signal analysis in python*, 2015.

[11] B. Shao, T. Li, M. Ogihara, (2008). *Quantify music artist similarity based on style and mood*. In Proceedings of the 10th ACM Workshop on Web Information and Data Management, ser. WIDM '08. New York, NY, USA: ACM, 2008, pp. 119–124. [Online]. Available: http://doi.acm.org/10.1145/1458502.1458522

[12] A. Lehtiniemi, J. Ojala, (2013). *Evaluating moodpic-a concept for collaborative mood music playlist creation*. In Information Visualisation (IV), 2013 17th International Conference. IEEE, 2013, pp. 86–95.

[13] W. J. Askey, H. Svendsen, (2009). *Method and system for sorting media items in a playlist on a media device*. Feb. 19 2009, US Patent App. 11/757,219.

[14] F. Diaz, (2017). *Spotify: Music access at scale*. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '17. New York, NY, USA: ACM, 2017, pp. 1349–1349. [Online]. Available: http://doi.acm.org/10.1145/3077136.3096471

[15] J. C. Gower, G. J. Ross, (1969). *Minimum spanning trees and single linkage cluster analysis*. Applied statistics, pp. 54–64, 1969.

[16] C. J. Krebs et al., (1989). *Ecological methodology*. Harper & Row New York, Tech. Rep., 1989.

[17] T. Cali´nski, J. Harabasz, (1974). *A dendrite method for cluster analysis*. Communications in Statistics-theory and Methods, vol. 3, no. 1, pp. 1–27, 1974.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, (2011). *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[19] N. Christofides, (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group (1976).