Playlist Shuffling given User-Defined Constraints on Song Sequencing

Sterling Ramroach[1] and Patrick Hosein[2]

[1] Department of Electrical and Computer Engineering
[2] Department of Computing and Information Technology
The University of the West Indies, St. Augustine campus
sramroach@gmail.com, patrick.hosein@sta.uwi.edu

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 *smartshuffle* 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: hierarchical clustering (Ward's variance minimization), nearest neighbor, and an approximation approach (Christofides' $\frac{3}{2}$-approximation). We then compare performances based on a defined performance metric. We also perform subjective evaluation to ensure that the proposed model enhances the listening experience of a user.

Keywords: Playlist shuffling · Music sequencing · Optimization · Data Analytics

1 Introduction

Today music consumers have access to extensive music collections and predefined playlists. These playlists are usually generated and sorted by music producers or disc jockeys. 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 has revolutionized the way in which users listen to music as they are now able to create personalized playlists. Services such as Spotify [22], iTunes [2], and Google Play Music [11], 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.

In this paper we focus on the sequencing of the playback of songs in a given playlist such that the transitions are soothing to the user. The attributes used to

determine the optimal sequencing can be provided with different weights by the user based on their personal preferences. Existing research is inconclusive about which combination of attributes and their various weightings lead to a measure of the pleasantness of a song. It is also inconclusive about which combinations of attributes in subsequent songs lead to a more pleasant playback sequence. As a result, we use the mean euclidean distance of all attributes of all songs in the playlist as a measure of pleasantness. When user defined preferences are considered, this metric will be adjusted to suit the user's preferences.

2 Related Work and Contributions

Previous researchers built tools to extract the meta-data from songs [18, 1, 17] and these were combined to form large datasets for public usage [5]. Current research focuses on assisting the consumer with the generation of personalized lists [6, 15, 21]. The mood of these lists are often supplied by the consumer and the playlist is built via the meta-data of songs in the pool [13]. A common metric used to assess the quality of the playlist is user satisfaction. This is quantified by observing listening times, customer return rates, track downloads or surveys [6]. Playlists are generated based on the user's listening history, hit statistics, and song tags.

The problem addressed in this paper is the ordering of songs in a list that has already been created by the user, such that the *pleasantness* of the sequence is maximized. 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 [4]. Naive ordering such as random or alphabetical has a low probability of producing a pleasant sequence, 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 most pleasant or smartest ordering of songs. The Nearest Neighbor approach has been shown to be superior for playlist generation [14] and is thus included in our investigation. We find that the general computational problem is the Travelling Salesman Problem (TSP) in euclidean space and so we also consider the Christofides' $\frac{3}{2}$-approximation algorithm [8]. The last algorithm considered is Ward's variance minimization hierarchical algorithm as it is known to provide reasonable results with low run-time for problems in euclidean space [10].

3 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 provides the most pleasant experience from beginning to end. No user input is required nor are labels or hit statistics used. Pleasantness is a measure of the pairwise euclidean distance between consecutive songs. This definition can be updated via a weighted computation discussed later.

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 [9] for descriptions of these). Our objective is to determine a playback sequence that minimizes the difference in attributes 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 artist there are no good similarity metrics since the similarity between two artists tends to be subjective [21]. We therefore 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 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.

In the general sense, this is the Travelling Salesman Problem (TSP) in euclidean space (NP hard) since we are going from song to song (city to city) with the intent of minimizing the sum of the attribute differences (sum of the distances between cities). We need to find a path which includes all songs but where the sum distance metric is minimized. The order in which songs are *visited* is the playback sequence. Some of the best known algorithms for this problem are the $\frac{3}{2}$-approximation via Christofides' Algorithm [8], and polynomial time approximation schemes (PTAS) by Arora [3] and Mitchell [16]. However, due to the lack of evidence supporting the practicality of PTAS [20], we focus on Christofides' algorithm. Next, we provide various heuristics for this problem with significantly less computational complexities.

4 Heuristics

We investigated various approaches to the problem in both its specific application and the generalized TSP and evaluated performance in terms of average distance between consecutive songs as well as computation time. We need to consider

computation time since it is always possible to obtain the optimal solution using an exhaustive search but this will take a considerable amount of time. In the following sections we assume that there are N songs in the playlist, which results in $N-1$ song pairs and hence transitions.

4.1 Christofides' Approximation Algorithm

Approximation algorithms are used to provide good results in reasonable time. Christofides' algorithm [8] 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, an Euler cycle is created from the combined graph with shortcuts to avoid visited songs. This algorithm ensures that the result is at most $\frac{3}{2}$ 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 1 illustrates an example of a playlist of 10 songs being represented as a graph before and after applying Christofides' algorithm.

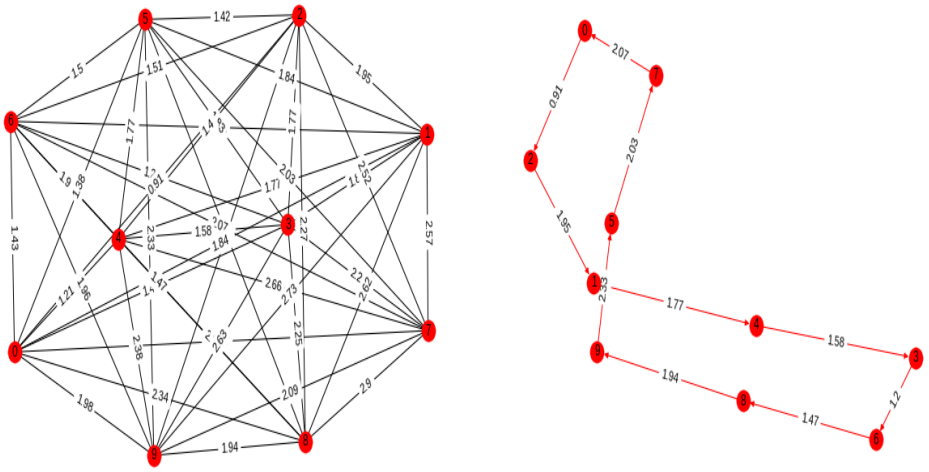

Fig. 1. The initial graph and solution by Christofides' algorithm.

4.2 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 sub-cluster is smaller in size than this threshold then the cluster will no longer be split. Otherwise the process continues with a new attribute. Figure 2 illustrates an example of a Hierarchical Cluster of a 6 song playlist comprised of Barry White (BW) and Rod Stewart (RS).

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.

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

The type of hierarchical clustering used is Ward's variance minimization method [7]. Euclidean distance is used as the distance function. 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:

$$v(a,d) = \sqrt{\frac{|d|+|b|}{T}v(d,b)^2 + \frac{|d|+|c|}{T}v(d,c)^2 - \frac{|d|}{T}v(c,b)^2} \tag{1}$$

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. Ward's method is the closest to the Nearest Neighbor in terms of properties and efficiency.

4.3 Nearest Neighbor Greedy Algorithm

The Nearest Neighbor Greedy algorithm uses the euclidean distance between songs. The playback sequence is determined as follows: start at a random song,

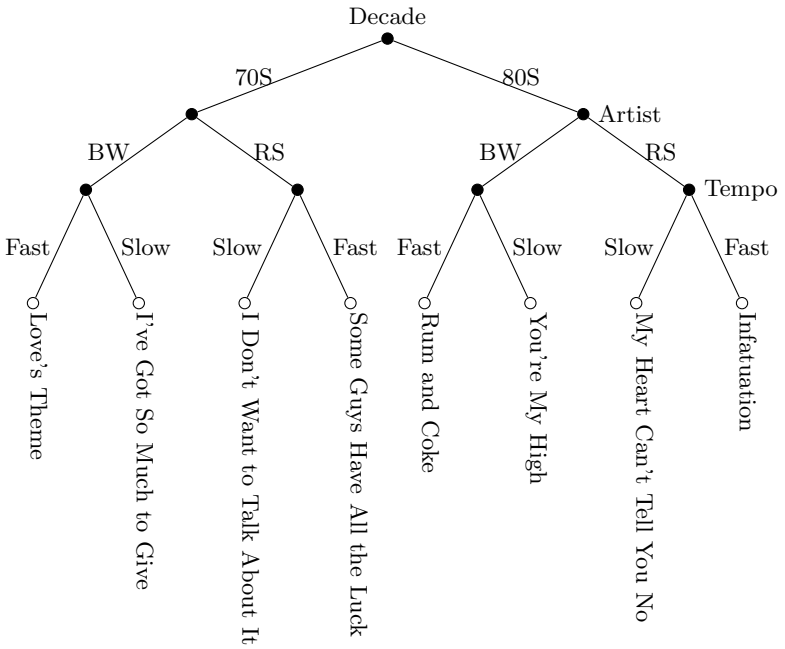

Fig. 2. Example of hierarchical clustering

find the nearest song that hasn't yet been added to the sequence, repeat until all songs are in the sequence.

4.4 Lower Bound on Optimal Solution

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. Note that a lower bound in performance corresponds to an upper bound on the performance metric which is the mean euclidean distance.

4.5 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 [12] 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.

5 Numerical Results

5.1 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, danceability, 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 two songs were 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.

5.2 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 $\bar{d}$ of a sequence of N songs is calculated by:

$$\bar{d} = \frac{1}{N-1} \sum_{i=1}^{i=N-1} d(i, i+1) \tag{2}$$

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

$$d(q, r) = \sqrt{dot(q, q) - 2 \times dot(q, r) + dot(r, r)} \tag{3}$$

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

$$dot(a, b)[i, j, k, m] = sum(a[i, j, :] * b[k, :, m]) \tag{4}$$

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. The other advantage is its computational

Table 1. Performance and Run Time Comparisons

Algorithm	**Mean Distance $\bar{d}$**	**Run time(s)**
Christofides' Algorithm	0.98	5.54
Ward's Method	1.24	2.36
Nearest Neighbor	1.14	2.08
Lower Bound on Performance	1.75	2.05
Upper Bound on Performance	0.66	2.08

efficiency when handling sparse data. Equations 3 and 4 were included in scikit-learn libraries [19] which were used in these experiments.

Table 1 presents performance and run time results for the various heuritics as well as the bounds. Christofides' algorithm attained the best playback sequence with a $\bar{d}$ 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. The Nearest Neighbor algorithm achieved the second best $\bar{d}$ of 1.14, and was computed in the second fastest 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. The results show that this greedy approach is faster than other viable approaches. As expected, the random shuffle (Lower Performance Bound) performed the worst, and the Upper Performance Bound resulted in an unattainable $\bar{d}$. Figure 3 illustrates the pairwise euclidean distances between all songs in sequences produced by a random shuffle and the best shuffle (Christofides' algorithm). The difference between songs are much higher in the random shuffle.

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 are 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 U_1 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 mood of a playlist. Another user U_2 may agree with U_1's judgment of which attributes seem irrelevant, but can be more interested in a shuffle where the "energy" of

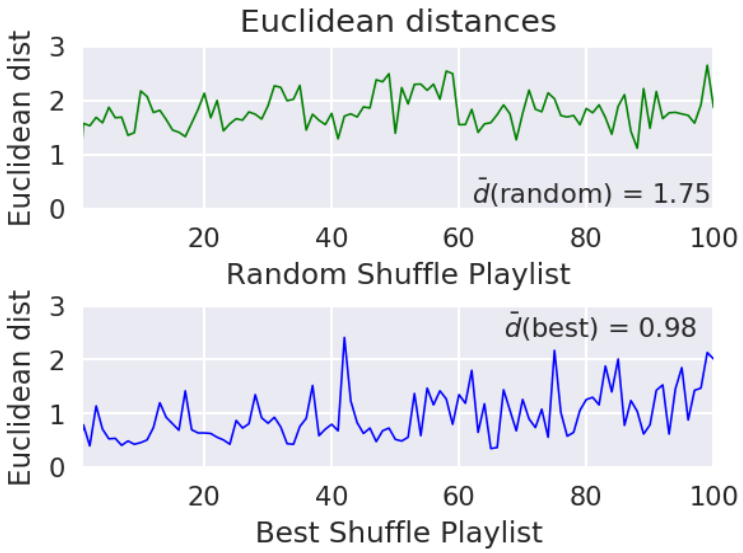

Fig. 3. Pairwise euclidean distances of random vs best sequences

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 U_1's preferences. It is expected that the $\bar{d}$ 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 $\bar{d}$ than all other techniques but with the worst execution time. The simplicity of U_1's preferences result in essentially the deletion of three attributes from the dataset.

Some complexity is introduced when the sequences are generated for U_2. U_2 values a smooth progression of energy as opposed to all other attributes. The following relationships for $\bar{d}_{energy}$ and $\bar{d}$ are expected:

$$\bar{d}_{energy}(weighted) \leq \bar{d}_{energy}(best) \leq \bar{d}_{energy}(random) \tag{5}$$

and

$$\bar{d}(best) \leq \bar{d}(weighted) \leq \bar{d}(random) \tag{6}$$

Figure 4 illustrates the mean euclidean distance of the energy attribute $\bar{d}_{energy}$ 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

Table 2. Comparison for User-Defined Attributes

Algorithm	**Mean Distance $\bar{d}$**	**Run time(s)**
Christofides' Algorithm	0.87	3.76
Ward's Method	1.17	2.26
Nearest Neighbor	1.04	2.01
Lower Bound on Performance	1.67	1.97
Upper Bound on Performance	0.59	2.04

attributes and uses Christofides' algorithm to generate the final sequence. The $\bar{d}_{energy}$ of the weighted shuffle playlist is 0.15. The best shuffle creates a sequence with $\bar{d}_{energy}$ equal to 0.18. The random shuffle results in a $\bar{d}_{energy}$ of 0.24. These results support the expected relationship in equation 5. This illustrates that one can use weights to amplify 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.

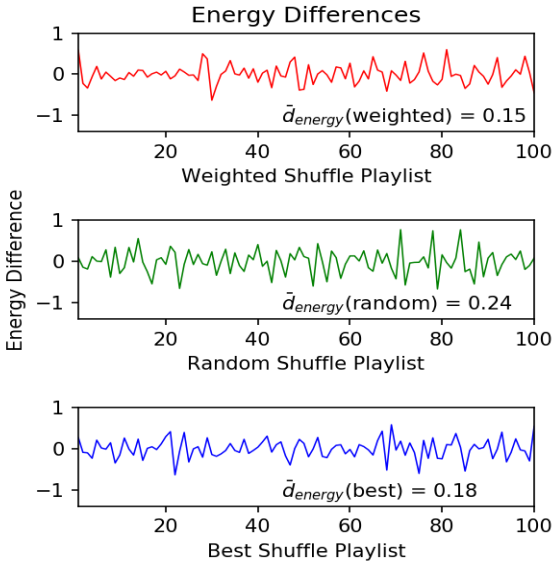

Fig. 4. Pairwise euclidean distances of the energy attribute.

Although U_2 wants a sequence in which energy is prioritized, the mean euclidean distances must still be considered. The $\bar{d}$ of the three sequences are as follows: the $\bar{d}$ of the best sequence is 1.14. The $\bar{d}$ of the sequence after the weights

were applied is 1.41, and finally, the $\bar{d}$ of the random sequence is 1.75. These results support the expected relationship described in equation 6. 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. The results show that objectively, the output sequence delivers what the user expects. Next, we investigate whether these results also hold subjectively.

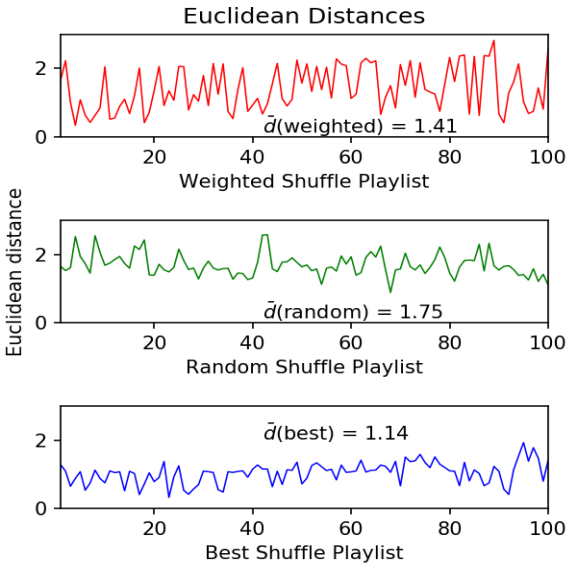

Fig. 5. Pairwise euclidean distances of all sequences

7 Subjective Testing

In this section we perform subjective testing to determine the effectiveness of the approach outlined in previous sections. We populated 3 playlists with 10 randomly selected songs. We then generated a random sequence and a sequence generated by the Christofides algorithm. These two sequences were provided to a subject (in random order) and the subject was asked to rank the pleasantness of the playback 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 three pools of 10 song pairs of random/best shuffles and presented to multiple subjects. Each subject rated 6 playlists. The results are provided in Table 3.

Table 3. Subjective ratings of various playlists.

Playlist	**Shuffle**	Computed $\bar{d}$	**Mean Score (1 - 10)**
A	Random	2.1	2.7
A	Best	1.8	7.5
B	Random	2.2	3.7
B	Best	1.8	7.7
C	Random	2.0	2.6
C	Best	1.7	7.4

Table 3 summarizes the results obtained from the subjective testing. All subjects gave the random shuffle a significantly worse score than the best shuffle. Objectively, the difference in $\bar{d}$ for random versus best shuffles for all sequences are within the range $0.2 \leq \Delta\bar{d} \leq 0.4$. There was a decrease in $\bar{d}$ 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 3.00. Whereas the average score of the best shuffles is 7.53. 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 subjectively produces a better gradient of mood 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.

The best shuffle of playlist A had $\bar{d} = 1.8$, as opposed to the 2.0 of the random shuffle for playlist C, the subjective scores of both playlists differ by 4.9. This indicates that a random shuffle with an $\bar{d}$ close to the best shuffle of another list is still vastly inferior to the best shuffle. $\bar{d}$ as a standalone indicator of the quality of a sequence may only be relevant for objective analysis. Although the best shuffle for B resulted in $\bar{d} = 1.8$ compared to the 1.7 of C, the average score given to playlist B by the subjects is 7.7 as opposed to the 7.2 of C. We expected the subjective scores to be inversely proportional to $\bar{d}$, but the results proved otherwise. Although a linear relationship between $\bar{d}$ and mean score does not exist, 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 100 song playlist.

This approximation algorithm was followed by the Nearest Neighbour algorithm, 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 Nearest Neighbour algorithm is sufficient for our purposes because of its much better runtime.

The best technique (Christofides algorithm) was used to shuffle three 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 7.53 and 3.00, 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 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. Allik, A., Fazekas, G., Sandler, M.B.: An ontology for audio features. In: ISMIR. pp. 73–79 (2016)
2. Apple: itunes (2018), https://www.apple.com/lae/itunes/
3. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM (JACM) **45**(5), 753–782 (1998)
4. Askey, W.J., Svendsen, H.: Method and system for sorting media items in a playlist on a media device (Feb 19 2009), uS Patent App. 11/757,219
5. Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The million song dataset. In: Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011) (2011)
6. Bonnin, G., Jannach, D.: Automated generation of music playlists: Survey and experiments. ACM Comput. Surv. **47**(2), 26:1–26:35 (Nov 2014). https://doi.org/10.1145/2652481, http://doi.acm.org/10.1145/2652481
7. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods **3**(1), 1–27 (1974)
8. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group (1976)
9. Diaz, F.: Spotify: Music access at scale. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1349–1349. SIGIR '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3077136.3096471, http://doi.acm.org/10.1145/3077136.3096471
10. Gallardo, G., Wells III, W., Deriche, R., Wassermann, D.: Groupwise structural parcellation of the whole cortex: A logistic random effects model based approach. Neuroimage **170**, 307–320 (2018)

11. Google: Google play music (2018), https://play.google.com/music/listen
12. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics **10**(1), 196–210 (1962)
13. Lehtiniemi, A., Ojala, J.: Evaluating moodpic-a concept for collaborative mood music playlist creation. In: Information Visualisation (IV), 2013 17th International Conference. pp. 86–95. IEEE (2013)
14. Ludewig, M., Kamehkhosh, I., Landia, N., Jannach, D.: Effective nearest-neighbor music recommendations. In: Proceedings of the ACM Recommender Systems Challenge 2018. p. 3. ACM (2018)
15. McFee, B., Raffel, C., Liang, D., Ellis, D.P., McVicar, M., Battenberg, E., Nieto, O.: librosa: Audio and music signal analysis in python. In: Proceedings of the 14th python in science conference. pp. 18–25 (2015)
16. Mitchell, J.S.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. SIAM Journal on computing **28**(4), 1298–1309 (1999)
17. Mitrović, D., Zeppelzauer, M., Breiteneder, C.: Features for content-based audio retrieval. In: Advances in computers, vol. 78, pp. 71–150. Elsevier (2010)
18. Moffat, D., Ronan, D., Reiss, J.D.: An evaluation of audio feature extraction toolboxes. In: Proc. 18th International Conference on Digital Audio Effects (DAFx-15). DAFx-15 (November 2015)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Rodeker, B., Cifuentes, M.V., Favre, L.: An empirical analysis of approximation algorithms for euclidean tsp. In: CSC. pp. 190–196 (2009)
21. Shao, B., Li, T., Ogihara, M.: Quantify music artist similarity based on style and mood. In: Proceedings of the 10th ACM Workshop on Web Information and Data Management. pp. 119–124. ACM (2008). https://doi.org/10.1145/1458502.1458522
22. Spotify: Spotify (2018), https://www.spotify.com/us/