Towards Lean-Back Music Listeners: An Automated Method to Generate Highly Personalized Playlists

MARCO FURINI, Dep. of Communication and Economics, University of Modena and Reggio Emilia, Italy
MANUELA MONTANGERO, Dep. of Computer Science, University of Modena and Reggio Emilia, Italy
FRANCESCO POGGI, Dep. of Communication and Economics, University of Modena and Reggio Emilia, Italy

Playlists have become the soundtrack of our life but choosing the one to play is a long and boring task. To avoid this hassle, music streaming platforms and recent studies are proposing approaches to generate personalized playlists with the idea of transforming users into a lean-back music listeners. An ideal scenario where the music platform knows the right music to play. Through an analysis of the theoretical framework, we show that the achieved personalization is partial as it only considers what music to play, and does not consider that the musical taste, the propensity to listen to new music and the listening stream of users vary from hour to hour. In this paper, we propose a novel method to generate highly personalized playlists. Through an analysis of the user's listening history, the use of clustering algorithms and of dynamic programming, our method generates playlists by first understanding *what* songs the user likes, *when* she likes to play such songs, and *how* to sequence such songs to meet her musical taste. A comparison evaluation showed that 97% of the playlists generated by our method are better personalized with respect to the ones generated by the Spotify recommender system. To the best of our knowledge this is the first study that generates highly personalized playlists (i.e., *what, when* and *how*) able to transform a user into a lean-back music listener.

CCS Concepts: • **Applied computing** → **Sound and music computing**; • **Information systems**;

Additional Key Words and Phrases: Music playlist, listening history, clustering algorithms, dynamic programming

ACM Reference Format:
Marco Furini, Manuela Montangero, and Francesco Poggi. 2020. Towards Lean-Back Music Listeners: An Automated Method to Generate Highly Personalized Playlists. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 1 (May 2020), 21 pages. https://doi.org/XXXX

1 INTRODUCTION

Four billion playlists against sixty million songs: with no doubts, playlists are the backbone of Spotify, the music streaming service launched in Sweden in 2006[1]. But Spotify is no exception: any streaming service (e.g., Amazon Prime Music, Apple Music, YouTube Music) provides playlists of all kinds, from those that highlight the hits of the moment, to those with music suitable for office

[1]https://newsroom.spotify.com/company-info/

Authors' addresses: Marco Furini, marco.furini@unimore.it, Dep. of Communication and Economics, University of Modena and Reggio Emilia, Viale Allegri 9, Reggio Emilia, Italy, Italy, 42121; Manuela Montangero, Dep. of Computer Science, University of Modena and Reggio Emilia, Via Campi, Modena, Italy, manuela.montangero@unimore.it; Francesco Poggi, Dep. of Communication and Economics, University of Modena and Reggio Emilia, Viale Allegri 9, Reggio Emilia, Italy, Italy, 42121.

1551-6857/2020/5-ART $15.00
https://doi.org/XXXX

work, to those with songs suitable for people who do physical activity, to those who love pop or indie music, to those that aim to bring joy to people who feel sad [5, 29]. No wonder that 68% of the users listening time is spent listening to playlists.

Playlists provide a novel mode of music consumption [31], users love them and artists see playlists as a door to success. Indeed, getting a song into a million-followers playlist means popularity and money. For instance, a study by the European Commission [1] shows that the inclusion of a song within on Spotify's "Today's Top Hits" playlist, with around 26 million followers, increases streams by close to 20 million (in money terms, this means $116,000- $163,000 to an artist); similarly, artists have even found themselves in the Billboard Hot 100 by virtue of inclusion on a playlist. As stated in [12], playlists are today what radios were yesterday: a promotional tool for music.

What is a playlist? The terminology in the literature is not consistent and the term playlist is used to refer both to ordered sequences of songs and to unordered sets of songs [5]. Spotify introduced them in 2013 to assist users in the personal organization of their favorite music; playlists were a way to leverage the burden of having to browse huge music collection [8]. Indeed, [27] showed that when faced with an immense catalog, the music selection process might be long, tedious, anxious and unsatisfactory. Without playlists, users end up choosing the same well-known songs, undermining the potential of a system that gives them lots of listening possibilities, and threatening the business of these streaming service providers (i.e., if users always listen to the usual songs, what would be the add-on with respect to radios or to personal libraries?) [10, 14].

Who produces playlists? Thinking of playlists as simple sequences of songs manually produced by some professionals is deeply wrong. Indeed, although a manual approach was initially used, the automatic playlist generation has been the focus of many researches during the last decade [8]. It is of no surprise that music platforms were flooded with hundreds of thousands of playlists automatically produced by algorithms that exploited the immense music catalog to select songs based on *similarity* (i.e., coherence of the tracks' low-level audio features), *collaborative filtering* (i.e., community-provided ratings), *frequent pattern mining* (i.e., frequently presence on different playlists), *statistical models* (i.e., the previous played song), *hybrid* (i.e., combination of different techniques) [5]. In addition to songs selection, another key aspect of playlists generation is the songs order [31] as good playlists cannot be generated without considering the ordering of the songs [3]. For instance, listening to the Beatles' "Abby Road" on random order is not nearly as enjoyable as listening to it in the original order [17]. The importance of songs order is highlighted by Spotify that uses the "shuffle" feature as a discriminator between the premium and the free version of its mobile app: those who have the free version must use the "shuffle" mode, whereas those who pay for the premium account have the "original order" feature. However, it is to note that, although approaches that consider songs ordering have been shown to outperform approaches with no songs ordering in terms of log likelihood [26], the songs ordering process makes playlist generation a highly complex task [26]. Indeed, it is necessary to establish an ordering criterion and in literature the approaches have not moved in the same direction (e.g., best tracks of given artists [7], most popular tracks at the end of the sequence [5], first and the last tracks are the important ones [17], just use the user's preferred tracks [3]). However, recently, recommender systems generate playlists using a *similarity* approach, where the term similarity might refer to low-level audio features (e.g., loudness, acousticness) or to high-level metadata (e.g., lyrics, genre): starting from a song, the next song is chosen to have a smooth transition between the two songs; the process continues using the characteristics of song $(n-1)$ to choose song (n). The idea is that users perceive songs transition as one of the most important thing in a sequence of songs [25].

The new arising problem within this scenario lies in the number of playlists: when a user opens the music streaming platform she is faced with too many playlists and the choice becomes long, boring and often ends up with a click on the playlists already listened to [27]. To address

this problem, streaming platforms are moving towards playlists personalization with the goal of transforming the user into a lean-back music listener who doesn't have to search for music, but simply has to sit back and listen to what the algorithms have decided for her [9, 22]. In this regards, Spotify provides users with a limited number of personalized playlists (e.g., usually from one to ten) to help users in the playlist selection process (e.g., *Daily mix, Time machine, Discovery weekly* and *Release radar* are the user-tailored playlists that Spotify produces). This personalization is achieved with machine learning algorithms that crunch huge amount of user's data and metadata (e.g., listening history, skipped songs, browsing behavior). However, it is to note that the user still has an active role: even though the number of playlists is limited, the user has to choose which one is best suited to the activity she is performing. For instance, while working, is it better a Daily mix or a Discovery Weekly? After dinner is better to listen to the Release Radar or to the Time Machine playlist? Is the Saturday afternoon playlist okay on Monday afternoon too? Difficult to say because users' habits are not the same throughout the day and change from day to day [14]. Therefore, the availability of these playlists might reduce the time it takes to choose what to listen to, but it still requires users to have an active role as a manual check is required to see if the playlist has suitable songs.

To transform a user into a lean back music listener, it is necessary to go beyond the proposition of a limited number of playlists and the theoretical contribution of the present paper is the proposal of a novel method to achieve this goal. The idea is to understand *what* a user wants to listen to *when* she opens the music platform and *how* to sequence the selected songs to meet her musical taste: our method understands if a user wants to listen to relaxing or exciting music, if she wants songs to concentrate or to have fun and then it generates the playlist accordingly. No time wasted to chose the playlist to listen to, as our method generates only one highly personalized playlist that meets her musical tastes and her listening habits. In particular, the produced playlist is **user-tailored** to meet the user's musical tastes and her propensity to listen to new music (e.g., not only music she likes, but also music she has never listened to before), **time-sensitive** to match her listening habits (e.g., at 10:00am she will find a different playlist than at 03:00pm, just as yesterday's 10:00am playlist is different from today's 10:00am playlist), **ordered** to meet how she listens to music (e.g., we avoid using general purpose rules as they may not be suitable for the user).

Technically speaking, our method operates in two steps: the first one selects a set of songs to be included in the playlist and the second one orders them. The selection is based on the listening time and on the low-level audio features of the songs listened to by the user in the last three months. This data is processed by two clustering algorithms (i.e., k-means and Furthest-Point-First) to understand the user's listening habits (i.e., inclination towards new music, musical tastes throughout the day) and then these listening habits are passed to the Spotify recommender system that returns a set of songs that best suit the user's listening habits. The second step understands *how* the user listens to music and then uses dynamic programming to create an optimal sequence of songs that is as close as possible to the user's listening habits. Indeed, to generate personalized playlists and to turn a user into a lean-back music listener, it is necessary to understand how the user listens to music: does she prefer a sequence of songs with smooth transitions between two consecutive songs? Does she prefer to start with high-energy songs? Does she prefer to have a sequence of songs that are very different from each other? Hard to say because users have different tastes and habits from each other and the literature gives us general insights that are interesting but may be useless for the considered user. The analysis of the listening history tells us *how* the user listens to music and the dynamic programming allows us to replicate this listening mode by ordering, in an optimal way, the songs selected in the first step.

To evaluate our proposal, we set up an experimental assessment performing 96 different experiments. For each one, we compared our generated playlist against the personalized playlist

generated by the Spotify recommender system. In 93 out of 96 experiments (i.e., 97%), the playlists generated by our method are better personalized.

To the best of our knowledge we found no studies capable of achieving such high personalization. Therefore, the theoretical contribution of the present paper is to move literature studies towards the transformation of users into lean back music listeners by proposing a method to produce highly personalized playlists. Moreover, the present paper also provides a practical contribution as music platforms might employ our method to improve the music experience of their customers.

The reminder of this paper is organized as follows. In Section 2 we present a brief description of recent studies focused on playlist generation. In Section 3 we present details of our proposed method, where its evaluation is described in Section 4. Results are discussed in Section 5 and conclusions are drawn in Section 6.

2 RELATED WORK

In literature, music playlists have been studied under different points of view: recommendation music system that builds song-by-song playlists trying to identify the next song compared to the recent past [20] or that exploits user's preferences to recommend novel songs that are potentially interesting [30], playlist analysis to understand the characteristics that might bring playlist to popularity [23], human behavior analysis to understand the influence playlists have on users [19]. Given the goal of the present study, in the following we narrow down the literature to studies that focus on the personalization of playlists and to methods that order songs within playlists.

2.1 Production of Personalized Playlists

Over the last decade, many different studies agree on the need to customize playlists and in the following we briefly review recent approaches that personalize playlists according to different data such as user's emotions, listening habits and preferred artists.

Sen et al. [28] proposed a method to select songs within a database according to the user's mood. The system uses song lyrics and clusters them into fifteen different classes of moods. Then, the method use facial expression analysis to detect the user's mood and to map it into one of the cluster classes and it selects three different songs to play. A similar approach is proposed by Chakole et al. [4], where facial expression analysis is used to get the individual's emotion and to group songs accordingly. Songs are processed using low-level audio features like spectral flux and spectral role and are mapped into four emotional classes. These approaches are interesting but require the user to be in front of a device equipped with a webcam, while it is known that music listening takes place mainly while the user performs other activities (e.g., socializes with others, does sports).

Furini et al. [14] proposed to use the user's listening history to select a set of songs from the Spotify database. The idea was to get the user's propensity to listen to new music and to use low-level audio features to understand the user's musical taste. Then, by interacting with the Spotify recommender system, the proposal generated different sets of songs similar to the ones played by the user in the recent past, and selected the one according to the user's propensity to listen to new music. No songs ordering was proposed to actually produce playlists. Cheng et al. [6] proposed a personalized music recommender system based on the user's listening habits. The idea was to use word embedding techniques in music play sequences to find and suggest similar songs. Matrix factorization and a k-nearest approach were used for features learning and discovery. The experimental evaluation was based on two different datasets and it showed that it is possible to recommend songs according to a music play sequence. We observe that this approach achieves a partial personalization as it is not time-sensitive, nor it orders songs according to the user's musical taste. Andric and Haus [3] proposed to analyze the user's listening habits to automatically produce personalized playlists. The method used the user's personal music collection and analyzed what the

user had listened to in the past. Then, it used a very simple model to predict the future behavior: if a song (or group of songs) has been listened to frequently recently, then it is likely that the song (or group of songs) will be played again. Similarly, if a song was played a long time ago, then it is likely that it will be played again in the near future. Being based on the set of personal songs, this approach does not involve new tracks and therefore it is not reasonable to think of using it in the scenario of streaming platforms. Pichl et al. [24] proposed a method to understand what users like by mining the personal listening habits from social media and by using socio-economic and cultural factors. The idea is to search the Spotify username on the Twitter platform and to get tweets with the `#nowplaying` hashtag. Then, these tweets are used to get the user's location and the location is used to get some cultural factors (from the World Happiness Report). By clustering low-level audio features (obtained from the Spotify API) and the cultural factors, the method identified classes of users. The authors did not propose any playlist generation method, but they suggested to embed their proposal within music recommender systems. A limitation of this approach is the need to have users willing to post tweets of what they play and willing to embed their geographical location inside these tweets.

Germain and Chakareski [16] proposed to produce playlists based on artist similarity. They considered the user's listening history and the user's Facebook likes to artists' pages. Then, they searched for similar artists using a third-party service (i.e., Echonest API) and they assigned a score to every artist based on the frequency of his/her appearance within the listening history. Finally, they constructed a playlist comprising randomly selected popular songs associated with the most frequently cited artists. We observe that this approach requires personal information available on Facebook to work; thus it might rise privacy concerns and does not work for users who do not have (or are not willing to share) a Facebook profile. Lin et al. [21] proposed a recommendation system based on the novelty of the singer and on their similarity to the user's favorite artists. To achieve this, they ask the user's favorite artists and uses a third-party system (i.e., Last.fm API) to get a list of similar artists. Finally, they remove the user's favorite artists from the list and recommend the remaining ones. This approach achieves a partial personalization as it is not time-sensitive, nor it orders songs according to the user's musical taste.

2.2 Songs Ordering

Establishing the right sequence of songs within a playlist is a difficult and very delicate task [31] and in the literature we found different approaches: there are researchers who argue that songs order is not important and therefore they think of playlists as simple sets of songs and there are others who argue the opposite and think that it is essential to establish a sorting criterion of the tracks (in this regard, Bonnin and Jannach [5] provide a very comprehensive overview on the subject). On a theoretical level it is difficult, if not impossible, to tell who is right or wrong; but on a practical level there are no doubts: Spotify, the most popular music platform in the world, forces ordinary mobile users to play playlists with random songs order (i.e., the well-known "shuffle" mode), whereas premium mobile users might play playlists with the *correct* order. Unfortunately, the criterion used to order the tracks is not known and for this reason it is interesting to briefly review literature studies that addressed the question.

A well-known and largely employed approach is the use of songs similarity to order tracks within a playlist, where similarity might refer to low-level audio features or to high-level data such as lyrics artists, genre, ratings [20]. For instance, Sarroff and Casey [25] investigated which audio features are perceived as most important for song transitions and their results showed the importance of fade duration and of the mean timbre of track endings and beginnings. Flexer et al. [11] used audio similarity to create a smooth transition between the start and the end of two consecutive songs. There is no comparison with other approaches and the evaluation is quite limited as it is done

through simple genre labels match. Furthermore, the approach proposed a general rule to order songs, but the user is not considered. Ikeda et al. [18] proposed a playlist recommendation method to recommend a music sequence that has smooth transitions of the acoustic features. The idea is to describe the dataset songs through two acoustic parameters and to map these songs on a Cartesian plane. Through the comparison of different functions, the study proposed a mathematical method to order the songs. De Almeida et al. [2] proposed two methods to randomly select an ordered list of songs to be played in situations where it is necessary to play songs with different styles (e.g., a wedding party or parties in general) starting from songs that the user certainly likes. The idea is to ask user two (or more) anchor songs and the number of songs in the playlist and then the proposed algorithms select songs to generate heterogeneous playlists with smooth transitions between consecutive songs. Since users have an active role, this approach does not transform users into lean-back music listeners.

Few studies used a different approach as they focus on some aspects of the songs order. For instance, Cunningham et al. [7] suggested that songs order is not important if the playlist contains only the best tracks of the user's preferred artists. A similar idea was suggested by Andric and Haus [3]. Hansen and Golbeck [17] asked 52 volunteers and results showed that users consider the first and the last tracks of a playlist as very important.

In summary, as suggested in [5], the current literature gives us important, but not sufficient, insights about what users find important. In addition, it is important to remember that each user is different and therefore generic insights are interesting from a theoretical point of view but, from a practical point of view, they do not necessarily work. Indeed, to generate a personalized playlist, the songs order must also be personalized. The method proposed in this paper moves in this direction: the songs and their order is personalized with respect to the time period the user wants to listen to music. To the best of our knowledge, in literature there are no studies able to achieve such a high level of personalization in music playlist generation.

3 METHOD

The generation of highly personalized playlists means understanding *what, when* and *how* users listens to music. So far, personalized playlists have been generated considering only one aspect (what), whereas when and how have not been taken into consideration by playlist generation algorithms. This partial personalization might generate playlists that the user likes, but they might not be appropriated for the time the user wants to listen to music (i.e., users generally listen to different type of music during the day, for example quiet and relaxing when they have to perform tasks that require concentration or more energetic and cheerful music during the pre-dinner or during sports activities) and it is not certain that the songs order encounters the user's tastes (i.e., some prefer to alternate tracks with different rhythms, others would like to have tracks with similar rhythms). Our approach is different. It produces highly personalized playlists and, to the best of our knowledge, there is no other approach in the literature (and in the market) that produces playlists with this level of personalization. It generates a single playlist to play when the user accesses to the music platform and the playlist is generated to meet the user's listening habits (i.e., it is time-sensitive and it is ordered according to how the user listens to music).

Figure 1 shows the general architecture of our proposed method. It is composed of two main steps: the first one is in charge of producing a set of songs that represents *what* the user likes *when* she wants to listen to music. This is accomplished by processing, transforming, filtering and clustering the user's listening history (i.e., the list of songs recently played by the user) to generate a set of songs similar to those that have been listened to by the user (i.e., similar music and similar percentage of new/known music the user is used to listen to) at a specific time of day. It is to note that this step is based on an external music recommender system (i.e., the one of Spotify in our

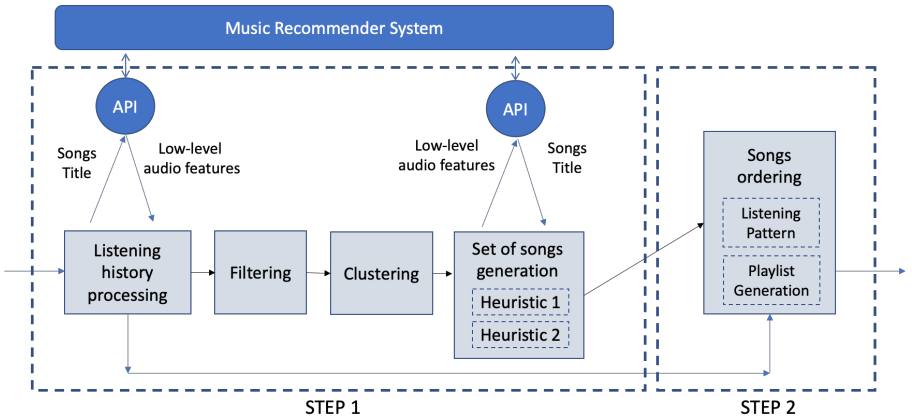

Fig. 1. Architecture of the proposed method.

case) to transform a song into a sequence of low-level audio features and to retrieve songs similar to a sequence of low-level audio features. The second step generates the playlist by ordering the selected set of songs in order to mimic *how* the user listens to music with respect to her listening habits. In the following, we provide details of these two steps.

3.1 Listening history processing

This module is in charge of understanding the user's listening habits. To this aim, it relies on the user's listening history, which is a time-coded sequence of the songs played by the users in the recent past. Every song is described through different high-level metadata (e.g., artist name, song title, music genre) and is transformed into a sequence of low-level audio features by using an external music recommender system (the present study exploits the Spotify API). Mathematically speaking, suppose that the listening history is composed of h tracks. Every song s is transformed into a vector of features $x_s \in \mathbb{R}^f$, where f is the song-feature dimension (e.g., Spotify uses the 12 audio features described in Table 1 to characterize every song). Therefore, the listening history can be seen as a matrix H with h rows and f columns.

The listening history is used to understand the amount of new music that the user listened to, but since the user's listening habits might vary during the day, we use a subset of the listening history. Indeed, if we suppose the user connects to the music platform on day d and time t, we are interested to know the music she listens to starting at time t. To this aim, let P be the time period that starts at time t and ends at time $t+v$: $P = [t, t+v]$. Without loss of generality, here we assume that the listening history in P is not empty and we will discuss about the proper length of the time window (i.e., the value of v) in the experimental phase.

For each day i in the listening history, we consider the set of songs $H_i \subseteq H$ played in period P and we compute the percentage of new tracks played by the users (i.e., tracks never played before day i), making a distinction between unknown artists (i.e., artists never played before day i) and artists known to the user (i.e., artists already played before day i). In details, we compute the following:

$$\text{NTNA} = 100 \cdot \frac{\sum_{i \in Days} \text{dNTNA}_i}{h},$$

$$\text{NTKA} = 100 \cdot \frac{\sum_{i \in Days} \text{dNTKA}_i}{h},$$

Table 1. The 12 Low-level audio features used by Spotify to describe songs.

Key	Description
Acousticness	Song acoustincness [0.0 – 1.0]
Danceability	Song danceability [0.0 – 1.0]
Energy	Perception of intensity and activity [0.0 – 1.0]
Instrumentalness	Vocal within the song [0.0 – 1.0]
key	The key the track is in
Liveness	Is the track performed live? [0.0 – 1.0]
Loudness	Track loudness in decibels
Mode	Track modality (major or minor) [0 or 1]
Speechiness	Presence of spoken words [0.0 – 1.0]
Tempo	Track tempo in beats per minute
Time_signature	Track time signature
Valence	Musical positiveness [0.0 – 1.0]

where, *Days* is the set of days in the listening history, dNTNA_i is the number of new tracks by new artists played during day i in the period P and $\mathsf{dNTKA}i$ is the number new tracks by known artists played during day i and period P.

The pair $Ph = (\mathsf{NTNA}, \mathsf{NTKA})$ represents the user's listening habits in period P, for what concerns newly played music and will be used to personalize the playlists.

3.2 Filtering

The listening history cannot be used entirely because during the day, the user's listening habits might vary (e.g., what is played in the morning might be different to what is played in the evening). For this reason the listening history is filtered as follows.

Suppose a user connects to the music platform on day d and time t, we only consider the $h_d \leq h$ songs played by the user during the period P. Therefore, we denote with $H_d \in \mathbb{R}^{h_d \times f}$ the matrix containing only those rows of H that represent the user's listening history during period P.

3.3 Clustering

To understand *what* the user listens to, the listening history is analyzed with clustering techniques that allow understanding whether the played songs have common audio characteristics or not. It is worth reminding that each song is characterized by a set of audio features (12 in our case). In particular, the analysis of the listening history is done through the use of two different clustering algorithms (*k-means* and *Furthers-Point-First*) to prevent the specificities of the algorithm from affecting the results.

K-means is probably the most known and used clustering algorithms in the literature. It works in a very simple way: initially, k points are randomly chosen as initial cluster centroids. Each point of the dataset to cluster is assigned to the group of the closest centroid. Then, the centroids of each cluster are recalculated and the assignment of the dataset points is done again. These two steps are repeated until the number of desired clusters is reached.

Furthers-Point-First (FPF) is less known but it has proven to be very effective in media contexts [13, 15]. It is based on a simple iterative clustering strategy: initially, all the dataset points form a single cluster and a center is randomly chosen among them. Then, a new cluster is added in two steps: (1) the center of the new cluster is the point that is farthest form its center; (2) all the points that are closer to the new center than to their previous one are moved to the new cluster. These two steps are repeated until the number of desired clusters is reached. Once the clustering with *FPF* is done, we compute centroides for all its clusters, to be used in the next step. Indeed, centroids might not be points of the cluster but have an average distance from all the cluster points, while centers might be points that lie on the outbound of the cluster. In our opinion, this makes centroid better representatives of their clusters.

Both clustering algorithms require a distance measure and a number of clusters to produce (i.e., k). As for the former, we use the Euclidean distance to measure the distance between any two rows in H_d, whereas for the latter, we use the Davies-Bouldin index to estimate the most appropriate number of clusters. This index uses cluster-based information to measure the cluster quality, combining intra-cluster cohesion (how much clusters' elements are near each other) and separation between clusters (how far the elements of different clusters are). The smallest the index, the better the clustering. We run both clustering algorithms for increasing number of clusters k, for $k \in [2..h^{1/2}]$, and we retain the solution that minimizes the Davies-Bouldin index.

In conclusion, the clustering module produces: clustering K = k-means(H_d) generated with k-means and n_k clusters, and clustering F = FPF(H_d) generated with FPF and n_F clusters. As the two algorithms computes the clustering using different techniques it is likely that $n_k \neq n_F$.

3.4 Set of Songs Generation

This module takes as input clustering K and clustering F (computed starting from the same rows in H_d) and outputs a set of songs with NTNA and NTKA closest to the ones of the users. To this aim, it exploits the music recommender system that requires a pair $(trackID, f)$ (f is a feature vector, not necessarily related to the track ID) and returns a set of m similar songs. We define $m = \ell/(n * 4)$, where ℓ is the number of songs the playlist has to have, $n \in \{n_k, n_F\}$ is the number of clusters, and 4 is the number of request to the recommender system for each cluster. For instance, suppose that the playlist should have $\ell = 48$ songs, and suppose that the number of clusters is $n_k = 4$, then, for each point, we ask the music recommender system to return $m = 3$ songs.

To produce the input of the recommender system, we design two different heuristics (i.e., *linear* and *sphere*). Our method operates as follows: given a clustering $C = \{C_1, C_2, \ldots, C_t\}$, with $C \in \{K, F\}$, both heuristics work cluster by cluster, for each cluster C_i, with $i = 1, 2, \ldots, t$, and produce a set of seeds S_C.

- **Linear Heuristic.** Given C_i, we order the points in decreasing order of distance from the centroid, by using the Euclidean distance. Then, we consider the following four points: (1) c_{i_1} that is the centroid, (2) c_{i_2} that is the point in position $\lfloor |C_i|/3 \rfloor$ of the ordering of the points, (3) c_{i_3} that is the one in position $2\lfloor |C_i|/3 \rfloor$ of the ordering of the points, and (4) c_{i_4} that is the farthest from the centroid. These four points are used as input of the music recommender system. To clarify, Figure 2 shows the idea graphically. The centroid might be considered as a synthesis of the user's musical taste for the set of songs in the cluster, whereas the second, the third and fourth points gradually move away from such representative of the user's tastes. The underlying idea is to find music that does not perfectly match the user's musical taste, thus providing her with a set of songs that are either familiar or slightly different from the usually played ones.

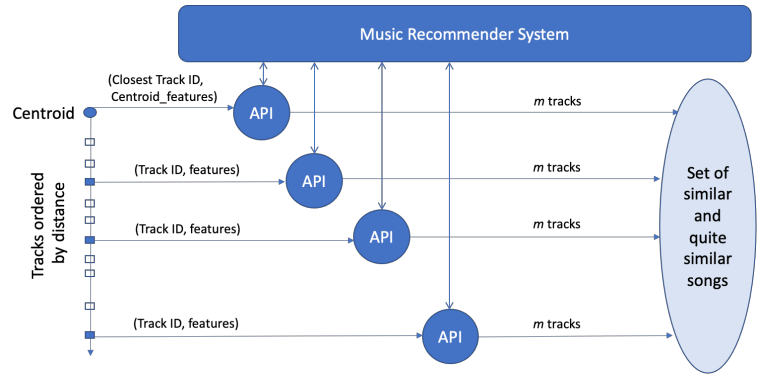

Fig. 2. Linear Heuristic: the centroid and three cluster tracks are used as input of the music recommender system to get a set of similar and quite similar songs to match and quite match the user's musical taste.

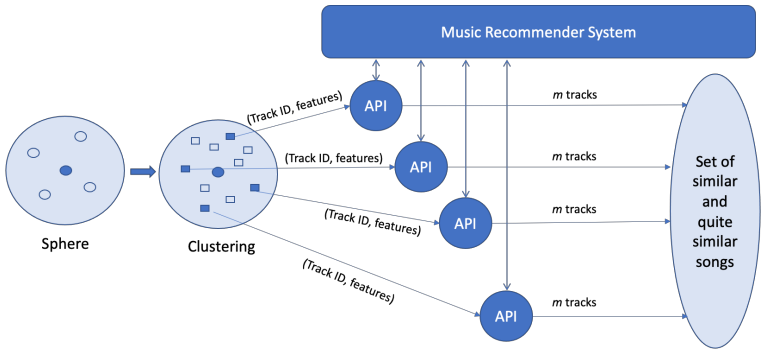

Fig. 3. Sphere Heuristic: four points in the hyper-sphere are randomly selected and used to select the closest corresponding tracks within the clustering. Then, the tracks are used as input of the music recommender system to produce a set of songs to match the user's musical taste.

- **Sphere Heuristic.** Given C_i, we consider a f-dimensional hyper-sphere where the center is the cluster centroid and the radius is the maximum distance of the cluster points from the centroid. Then, we select a random point rp in the hyper-sphere and we use it to compute a new point p whose coordinates satisfy the constrains of the acoustic features. To clarify, we round to the closest integer the coordinates that have to be integer values, and we set to the closer interval extreme those coordinate values that fall outside the feature interval. For instance, since the *acousticness* value has to lie in the range $[0, 1]$, if the corresponding coordinate of rp lies outside such interval, we set it to the closest acceptable value (i.e., a value of 1.2 becomes 1). Furthermore, since p might not be a cluster point, within the cluster we search for the point that is closest to p and we call it cp. The pair (c, cp) is then used as input of the music recommender system. We repeat this procedure as long as we have four different pairs. Note that if the cluster size is smaller than four, the selection process ends when the number of pairs is equal to the cluster cardinality.

The four different sets of songs (linear with k-means, linear with FPF, spherical with K-means and spherical with FPF) are then filtered to match the user's attitude to listen to familiar or new

music (i.e., the pair $Ph = (\mathsf{NTNA}, \mathsf{NTKA})$). Therefore, the set of songs with pair $(\mathsf{NTNA}, \mathsf{NTKA})$ closest to the user's attitude Ph is selected and it is the output of this module (and of Step 1).

3.5 Songs Ordering

The first step produced a set of songs that represents *what* the user likes *when* she wants to listen to music. The second step analyzes the user's listening history to understand *how* she listens to music and to produce a playlist that meets the user's listening habits starting from the set of songs.

3.5.1 Listening Pattern. Think of Anne and Louis, two friends who use music as the soundtrack of their daily running activity. Anne prefers to have quiet music at the start of the run and she likes to slowly increase the number of beats per minute while she runs; Louis instead prefers to start the activity with high energetic music and then he likes to decrease it while he continues to run. Each user has its own way of listening to music. That is why it is necessary to understand *how* a user listens to music: without knowing how, the playlist will never be fully personalized. As earlier mentioned, we found no studies able to fully personalize playlist.

To understand *how* a user listens to music, we analyze her listening history during the period P. Since each song is described through a feature vector (i.e., a point in f-space where f is the number of features of the vector), the listening history is a path that moves from one point to another point. This path is the user's *listening history pattern* and is formally defined as follows:

Definition 3.1. A *listening history pattern* $hP =< h_1, h_2, \ldots, h_k >$ of length k is an ordered sequence of k points $h_i \in \mathbb{R}^f$, where f is the song-feature dimension and where the user played song h_i just before song h_{i+1}, for $i = 1, \ldots, k-1$.

Similarly, the sought playlist pattern is defined as follows:

Definition 3.2. A *playlist pattern* $< p_1, p_2, \ldots, p_m >$ of length m is an ordered sequence of m points $p_i \in \mathbb{R}^f$, where f is the song-feature dimension and where song p_i is played in the playlist just before song p_{i+1}, for $i = 1, \ldots, m-1$.

The goal of the songs ordering module is to generate a playlist whose pattern is as more similar as possible to the user's listening pattern starting from the songs retrieved in the previous step.

3.5.2 Playlist Generation. Let C be the set of songs produced at step 1 and let hP be user's listening history pattern composed of k vertices. To generate the ordered playlist, this module defines the playing order of the selected songs in a playlist pattern P^*, so that its shape is as similar as possible to the shape of the listening history pattern hP. This allows the playlist to be highly personalized: indeed, in addition to *what* and *when*, the ordered playlist allows to mimic *how* a user listens to music.

To order songs, first, we define a distance measure between pairs of playlist patterns that captures the idea of similar shapes, and then we formalize our problem as an optimization problem that we can optimally solve with a dynamic programming algorithm (i.e., determine the order of the songs in the playlist so that its shape is the most similar to the listening history than any other ordering).

Definition 3.3. Let $d(\cdot, \cdot)$ be the euclidean distance. Given two playlist patterns of length t, $P_1 =< p_{1,1}, p_{1,2}, \ldots, p_{1,t} >$ and $P_2 =< p_{2,1}, p_{2,2}, \ldots, p_{2,t} >$, we define the following distances between P_1 and P_2:

- *Pattern Vertex Distance (VD)*: the sum of the pairwise distances of vertices in the same positions in the two patterns:

$$VD(P_1, P_2) = \sum_{i=1}^{t} d(p_{1,i}, p_{2,i});$$

- *Pattern Segment Distance (SD)*: the sum, in absolute values, of the differences of the lengths of corresponding segments in the two patterns:

$$SD(P_1, P_2) = \sum_{i=1}^{t-1} |d(p_{1,i}, p_{1,i+1}) - d(p_{2,i}, p_{2,i+1})|;$$

- *Playlist Pattern Distance (PD)*: the sum of the Pattern Vertex Distance and the Pattern Segment Distance:

$$PD(P_1, P_2) = VD(P_1, P_2) + SD(P_1, P_2).$$

This playlist pattern distance is used to compute a *similarity score* between the user's listening history and any other candidate playlist. Note that, by definition, the playlist pattern distance is always a non negative value. Furthermore, the larger the playlist pattern distance, the more the patterns differ in their shapes (meaning both songs and order). Therefore, given a listening history pattern hP and two candidate playlist patterns P_1 and P_2, P_1 is more similar to hP than P_2 if

$$PD(hP, P_1) < PD(hP, P_2).$$

To clarify, we now briefly describe the geometric intuition behind the definition of the playlist pattern distance. Observe that: (i) any playlist pattern can be plotted in a Cartesian coordinate system; (ii) each pattern vertex represents a song; (iii) the vertex ordering in the pattern reflects the songs order in the playlist. By definition, the playlist pattern distance PD is the sum of the pattern vertex distance (VD) and of the pattern segment distance (SD):

- the playlist pattern vertex distance (VD): the closer the two vertices, the more similar are the coordinate points (i.e., songs have similar audio features). Since, VD is the sum of the distances of vertices in the same position in the two playlists, VD indicates how similar songs in the same positions in the two playlist are, and it can be considered as a measure of *song-by-song local similarity* between two playlists.
- the playlist pattern segment distance SD: a segment connects two consecutive songs in a playlist, the longer the segment, the less similar the consecutive songs are, because they differ in the values of their features. Each SD term contributes with the absolute value of the difference of the lengths of two segments, one from the history pattern and the other from the candidate pattern, whose endpoints are in the same positions in the two patterns. The two patterns are similar if such corresponding segments have similar lengths. Indeed, the SD is small if a large number of corresponding segments in the two playlists has the same length. Conversely, the SD gets larger if many corresponding segments have different lengths. We observe that this measure is much finer than just comparing the total length of the two playlists, computed as the sum of the segments lengths. Indeed, two playlists might have the same total length but might be composed of successive segments of very different lengths. Therefore, SD can be seen as an estimation of the *song-after-song local similarity* between two playlists.

By considering both VD and SD, the PD function provides a *global measure* of similarity between two playlists combining at the same time local song-by-song and local song-after-song dimensions of similarity. Indeed, if two corresponding segments have similar lengths and close endpoints, they also have similar directions. Whenever this happens for many segments, then the two patterns have similar shapes.

An example is depicted in Figure 4. For the sake of presentation, we limit ourselves to playlist pattern vertices with two dimensions. The figure shows a history playlist pattern $hP =< h_1, \ldots, h_4 >$ (blue solid lines) of length four, and two candidate playlist patterns $P_1 =< p_{1,1}, \ldots, p_{1,4} >$ (red solid lines) and $P_2 =< p_{2,1}, \ldots, p_{2,4} >$ (green solid lines) of length four as well. The euclidean distance

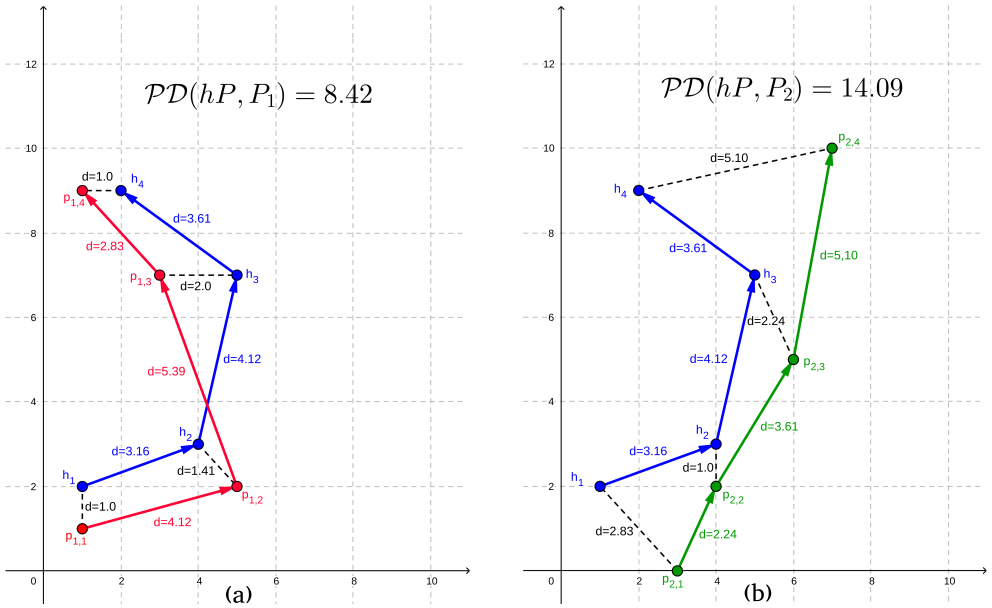

Fig. 4. Comparison between two candidate playlist patterns P_1 (a) and P_2 (b) generated for the listening pattern hP. Given hP, the value of the PD function for P_1 is smaller than for P_2. In this example, for the sake of clarity, we consider pattern vertices with two dimensions.

of endpoints of each segment is shown close to the segment itself. For instance, $d(h_1, h_2) = 3.16$ and $d(p_{1,2}, p_{1,3}) = 5.39$. Dashed lines connect vertices in the patterns that are in the same position and the euclidean distance between connected vertices is shown close to the dashed line. For instance, the Euclidean distance between the first vertex of hP and the first one in P_1 (resp. P_2) is $d(h_1, p_{1,1}) = 1$ (resp. $d(h_1, p_{2,1}) = 2.83$).

To compare the candidate playlist patterns against the history pattern we compute their PD with respect to hP. Considering hP and P_1 (red and blue patterns), we have:

$$VD(hP, P_1) = 1 + 1.41 + 2 + 1 = 5.41,$$

$$SD(hP, P_1) = |3.16 - 4.12| + |4.12 - 5.39| + |3.61 - 2.83| = 3.01,$$

and thus

$$PD(hP, P_1) = VD(hP, P_1) + SD(hP, P_1) = 8.42.$$

Considering hP and P_2 (green and blue patterns) we have:

$$VD(hP, P_2) = 2.83 + 1 + 2.24 + 5.1 = 11.17,$$

$$SD(hP, P_2) = |3.16 - 2.24| + |4.12 - 3.61| + |3.61 - 2.83| = 2.92,$$

and thus

$$PD(hP, P_2) = VD(hP, P_2) + SD(hP, P_2) = 14.09.$$

We conclude that P_1 is more similar to hP than P_2 as $PD(hP, P_1)$ is smaller than $PD(hP, P_2)$.

We are now ready to introduce the optimization problem.

Definition 3.4. The Playlist Pattern Distance Minimization Problem is defined in the following way:

INPUT: listening history pattern $hP =< h_1, h_2, \ldots, h_k >$ of length k and a song set of candidates $C = \{c_1, c_2, \ldots, c_m\} \subseteq \mathbb{R}^f$, with k ≤ m.

OUTPUT: a playlist pattern $P^* =< p_1^*, p_2^*, \ldots, p_k^* >$ that minimizes the playlist pattern distance PD with the listening history pattern hP, where $p_i \in C$, for $i = 1, \ldots, k$.

In essence, we look for the playlist pattern P^* of length k, with vertices in C, and such that

$$P^* = \arg\min_{pP} PD(hP, pP).$$

3.6 Dynamic Programming Algorithm for playlist generation

In this section we provide a dynamic programming algorithm to find an optimal solution to the pattern distance minimization problem in polynomial time. Observe that a brute force approach would require to enumerate all possible combinations, leading to exponential computational time.

Definition 3.5. Given a playlist (resp. history) pattern $pP =< p_1, \ldots, p_k >$ (resp. $hP =< h_1, \ldots, h_k >$) and an integer $i \in [1..k]$, $pP_i =< p_1, \ldots, p_i >$ (resp. $hP_i =< h_1, \ldots, h_i >$) is a *sub-pattern* of pP (resp. of hP) of length i.

We start by observing that the problem has optimal substructure: let $P^* =< p_1^*, \ldots, p_{k-1}^*, p_k^* >$ be an optimal playlist for $hP =< h_1, \ldots, h_k >$, then the sub-pattern $P_{k-1}^* =< p_1^*, \ldots, p_{k-1}^* >$ must be the best pattern for $hP_{k-1} =< h_1, \ldots, h_{k-1} >$, among all those ending with p_{k-1}^*. Therefore, we define the following sub-problem:

Definition 3.6. Given $i \in [1..k]$ and $j \in [1..m]$, the *Playlist Pattern Distance Minimization Sub-problem for i and j* is to find a playlist pattern $P_{i,j}$ with i vertices chosen in C, such that:

1) the last vertex is c_j;
2) it minimizes the playlist distance PD with the listening history sub-pattern of length i $hP_i =< h_1, h_2, \ldots, h_i >$.

We define $M[i, j] = PD(hP_i, P_{i,j})$ as the value of such optimal solution.

Observe that we have $k \times m$ sub-problems, and that the value OPT of the optimal solution of our original playlist pattern distance minimization problem is given by

$$OPT = \min_{1 \le j \le m} \{M[k, j]\}, \tag{1}$$

i.e., among optimal playlist patterns ending in each one of the m candidates in C, we select the one that minimizes the playlist pattern distance with the history pattern.

In order to compute the $M[k, j]$s, we use dynamic programming: we compute values $M[i, j]$, for all $j \in [1..m]$, for increasing values of i in the range $[1..k]$ in the following way:

- $M[1, j] = d(h_1, c_j)$, for $j = 1, \ldots, m$;
- For $i \in [2..k]$ and $j \in [1..m]$, $M[i, j]$ is computed by means of values $M[i-1, t]$s, with $t = 1, \ldots, m$, in the following way:

$$M[i, j] \;=\; d(h_i, c_j) + \min_{1 \le t \le m} \{M[i-1, t] + |d(h_{i-1}, h_i) - d(c_t, c_j)|\}. \tag{2}$$

Lemma 3.7. *Given the history sub-pattern hP_i and the candidate set $C = \{c_1, \ldots, c_m\}$, $M[i, j]$ is the value of the optimal solution to the Playlist Pattern Distance Minimization sub-problem for i and j, for all $i \in [1..k]$ and $j \in [1..m]$.*

Proof. The proof is by induction on i. For $i = 1$ we have $hP_1 =< h_1 >$ and, for each $j \in [1..m]$, $P_{1,j} =< c_j >$ and $M[1, j] = d(h_1, c_j)$. By definition we have

$$VD(hP_1, P_{1,j}) = d(h_1, c_j) \quad \text{and} \quad SD(hP_1, P_{1,j}) = 0,$$

thus $PD(hP_1, P_{1,j}) = VD(hP_1, P_{1,j}) + SD(hP_1, P_{1,j}) = M[1, j]$.

Assume now $i > 1$ and $1 \leq j \leq m$. Fix any $t \in [1..m]$. By inductive hypotheses, $M[i-1, t]$ is the value of the optimal solution $P_{i-1,t}$ for the sub-problem for $i-1$ and t, i.e., we have $M[i, j] = PD(hP_{i-1}, P_{i-1,t})$.

Let us now prolong the history pattern adding the i^{th} vertex, and let $P_{i,t,j}$ be the pattern obtained by adding vertex $c_j \in C$ to the end of $P_{i-1,t}$ (i.e., the last but one vertex of $P_{i,t,j}$ is c_k and the last one is c_j). By using (2) for the given t, we have:

$$\begin{aligned}
& d(h_i, c_j) + M[i-1, t] + |d(h_{i-1}, h_i) - d(c_t, c_j)| \\
= \; & d(h_i, c_j) + PD(hP_{i-1}, P_{i-1,t}) + |d(h_{i-1}, h_i) - d(c_t, c_j)| \\
= \; & d(h_i, c_j) + VD(hP_{i-1}, P_{i-1,t}) + SV(hP_{i-1}, P_{i-1,t}) + |d(h_{i-1}, h_i) - d(c_t, c_j)| \\
= \; & VD(hP_i, P_{i,t,j}) + SV(hP_i, P_{i,t,j}) \\
= \; & PD(hP_i, P_{i,t,j})
\end{aligned}$$

Varying t in the interval $[1..m]$, we have that

$$M[i, j] = \min_{1 \leq t \leq m} PD(hP_i, P_{i,t,j}) = PD(hP_i, P_{i,j}).$$

□

To determine the actual solution $P^* =< p_1^*, \ldots, p_k^* >$ that has value OPT, we use a $k \times m$ matrix V: in $V[i, j]$ we store the index t that was used to compute $M[i, j]$, for each pair of indices i, j. Then, starting from the last vertex of the history playlists pattern, going backwards, we recover all vertices of the optimal solution. In details:

- Vertex p_k^* (the last one) is candidate $c_r \in C$ such that $M[k, r] = \min_{1 \leq j \leq m}\{M[k, j]\}$.
- For each $i = k-1, \ldots, 1$, let c_t be the candidate selected for position $i+1$ in the solution, then $r = V[i+1, t]$ is the index of the candidate for position i, i.e, we have that $p_i^* = c_r$.

Computational cost. **Time.** There are $O(m \cdot k)$ sub-problem and the cost to compute the optimal solution to each sub-problem is $O(m + f)$ (remember that f is the song-feature dimension). Thus, the cost to compute all values $M[i, j]$ is $O(m^2 \cdot k)$, reasonably assuming that $f \in O(m)$. Computing OPT by means of equation (1) and determining the actual solution adds a $O(m + k)$ additive factor. In conclusion, the time computational cost to find the optimal solution is dominated by $O(m^2 \cdot k)$. **Space.** We need to store two $k \times m$ matrices (M and V), thus we have that the space computational cost is $O(k \cdot m)$.

4 EXPERIMENTAL ASSESSMENT AND RESULTS

To evaluate our proposed method, we proceed as follows:

(1) given a period P, we generate a playlist for this period;
(2) we compare our generated playlist against the ones produced by other approaches;
(3) we use the PD function as a metric to measure the similarity degree between the user's listening history pattern and the generated playlists.

Before presenting the experimental details, it is worth recalling that the time period is defined as $P = [t, t + v]$, where t is the time the user wants to play music and v is set to one hour in this experimental phase. Indeed, we empirically noted that longer periods might include very different listening habits (e.g., during the evening a user can switch from very energetic music during pre-dinner to very relaxing music after-dinner) and shorter period might be misleading. In essence, if the user wants to play music at 12:00, the user's listening history has just the songs played from 12:00 to 13:00.

Our method is compared against the following ones:

- **REC-1.** Each song of the user's listening history is used as input to the Spotify recommender system to get a similar song. Notice that since this playlist is generated song-by-song, no song-set is produced;
- **REC-2.** To provide more contextual data to the recommender system, three songs of the user's listening history (the current, the previous and the next) are used as input of the Spotify recommender system to get a similar song. Again, since the playlist is generated song-by-song, no song-set is produced;
- **HYB-1.** A hybrid method that uses the Spotify recommendation system to generate a pool of candidate songs (i.e. the song-set), and uses our dynamic programming algorithm to select the songs to include in the user-tailored playlist, and to sort them. The song-set is created as follows. Since the maximum number of songs that can be passed to the recommender is five, we split the listening history in blocks composed of five consecutive songs. Then, for each block, the recommender is asked to return a number of songs so as to have a song-set of the same size as that of the dynamic programming method.

We analyze our method over different periods. In particular, since the period is one hour long, we consider 96 different experiments and, for each one, we generate the playlist and we compare it against those generated by the approaches described above. For each day of the listening history, we compute the listening history pattern of period P and we only consider the longest one. The underlying idea is that this pattern, being the longest, it is the user's preferred one. As for the days of the listening history, we limit the analysis to the previous 90 days. Indeed, empirically we noted that three month can be considered a reasonable period to observe in order to understand the user's listening habits: shorter periods might be influenced by specific events like a newly released CD or a concert, whereas longer period might no longer represent the user's musical preferences as the music taste changes over time [3]. It is also worth noting that the listening history returned by the Spotify API includes only tracks that has been played for at least 30 seconds. Therefore, we can assume that these songs were appreciated by the user (otherwise she would have skipped the song after few seconds). Note that different strategies could be adopted (e.g. different time windows, different filtering), but we believe that the aforementioned settings are reasonable for the purpose of this experiment.

We consider a user's listening history composed of 2853 songs and, for each time window, we selected the longest list of songs played consecutively, so as to have 24 listening history patterns (i.e. one for each hour of the day) that correspond to the user's habits in the considered periods. On average, this pattern is 31.4 song long.

To have a more comprehensive evaluation of our method, this experimental evaluation does not select a single set of songs but uses the four sets of songs produced during step 1 (i.e., being based on the user's musical taste, the selection might affect the experimental results). This causes our method to generate 4 different playlists, called DYN-1, DYN-2, DYN-3, and DYN-4, for each one of the 24 periods (e.g., 00:00-00:59, 01:00-01:59, etc.). To evaluate the quality of the 96 generated playlists, we compute the *PD* value of each playlist and we compare the obtained results against the

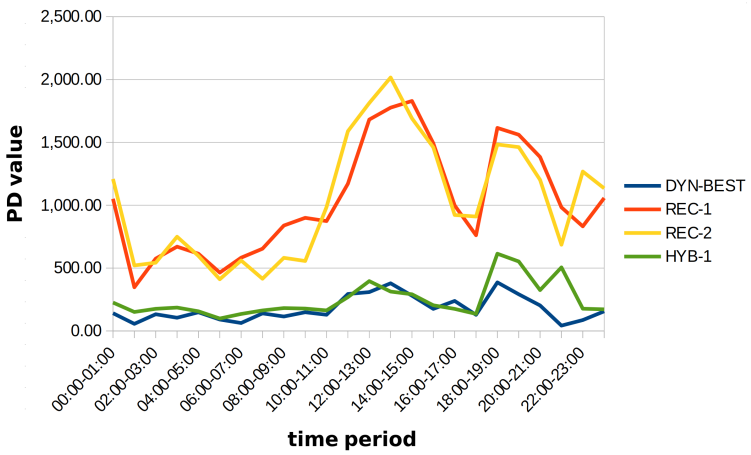

Fig. 5. A line chart with the results of our experiment. For the dynamic programming method we report only the best values for each input (DYN-BEST). Our approach (DYN-BEST) outperforms the ones based on the Spotify recommender (REC-1 and REC-2) and also has comparable but still lower PD values than the hybrid method (HYB-1) except than three cases.

ones obtained by computing the *PD* value for playlists generated with approaches REC-1, REC-2 and HYB-1[2]. It is worth noting that each of these methods produces different playlists for each run of the experiments. This is due to the fact that they are based on the Spotify recommender, which returns a different song (or a different set of songs) for each invocation. Consequently, different runs of the experiments produces different PD values with these three approaches. For this reason we performed ten runs of the experiments, and observed that the results obtained were very similar. Due to space limitations in this paper we report the results of the best run of the experiments for the three compared methods, i.e. the one with lower average PD values for each method.

Figure 5 summarizes the obtained results by showing the *PD* value (the lower the value, the better) for each period and for each generated playlist (for clarity, we have reported only the best playlist generated by our method). It is possible to note that the two playlists generated by the Spotify recommender system are very far from the user's listening history pattern (i.e., *how* the user listen to music), whereas the hybrid approach obtains comparable results, although higher than those obtained by the method proposed in this paper.

To get a complete understanding on the experiments, Table 2 reports the average *PD* values for the generated playlists (cells with blue background highlight the best *PD* values). In general, it can be noted that 93 times out of 96 (96.9 %), our method generates playlists that performs better than the ones generated by the Spotify recommender system and, in just three cases, it produces results slightly worse than the hybrid approach (where the idea of using dynamic programming is employed). These results show that playlists based on *what* user plays are outperformed by our proposed method that produces highly personalized playlists based on *what*, *when* and *how* a user listens to music.

[2]All the material needed to replicate the experiments presented in this section (e.g. the input playlist, the code developed to create the user's playlists with our approach and the three compared methods, etc.) is available at https://github.com/fpoggi/APPAGO

Table 2. *PD* values computed for each generated playlists. Best absolute *PD* values (the lower, the better) are highlighted with a blue background.

Input		Our Proposed Method				Spotify Recommender		Hybrid
Time	**History Length**	**DYN-1**	**DYN-2**	**DYN-3**	**DYN-4**	**REC-1**	**REC-2**	**HYB-1**
00:00-01:00	32	319.70	142.63	177.88	141.19	1051.36	1209.08	225.61
01:00-02:00	10	66.07	56.18	59.81	64.34	347.42	521.02	150.71
02:00-03:00	26	157.11	132.62	197.77	151.06	575.03	542.78	176.15
03:00-04:00	31	104.78	201.61	178.34	224.78	670.28	749.58	185.66
04:00-05:00	31	148.18	152.03	197.28	159.92	614.24	597.09	156.16
05:00-06:00	22	106.52	95.50	104.15	90.68	464.02	410.95	98.76
06:00-07:00	19	107.05	63.20	96.30	100.81	582.62	560.61	135.05
07:00-08:00	21	139.70	160.66	157.21	209.95	653.72	415.12	162.87
08:00-09:00	28	182.30	114.79	134.58	182.51	838.33	581.77	181.72
09:00-10:00	26	443.48	148.70	157.14	169.38	899.78	556.50	177.84
10:00-11:00	26	274.43	236.85	128.66	276.11	873.95	985.68	163.32
11:00-12:00	38	615.02	444.10	295.04	294.03	1170.89	1588.82	268.68
12:00-13:00	49	580.92	353.58	502.57	308.82	1681.43	1812.97	396.37
13:00-14:00	62	378.92	380.80	471.83	569.72	1776.51	2016.15	313.00
14:00-15:00	54	430.92	508.72	332.54	280.37	1829.61	1691.06	291.78
15:00-16:00	38	275.19	175.16	318.05	191.90	1492.97	1460.67	203.75
16:00-17:00	35	285.84	240.67	248.26	238.73	998.63	922.36	175.45
17:00-18:00	25	196.28	185.63	146.35	128.38	761.40	910.08	135.33
18:00-19:00	36	550.80	626.96	386.20	428.72	1615.23	1483.60	614.33
19:00-20:00	32	471.59	291.77	500.74	458.08	1561.00	1462.33	552.55
20:00-21:00	31	243.91	202.74	481.40	283.39	1383.61	1201.94	324.78
21:00-22:00	17	398.72	42.44	424.23	115.56	982.99	685.98	505.36
22:00-23:00	32	86.26	173.09	173.86	163.74	832.02	1268.00	176.81
23:00-00:00	32	224.17	155.97	185.43	236.22	1057.66	1134.24	171.97

5 DISCUSSION

When a music catalog is too large, the user struggles to choose what to listen to and ends up always choosing the same music [27]. This applies to both individual tracks and playlists. To overcome this problem, streaming platforms are trying to personalize playlists and generate them according to *what* the user listens to. In the literature, several studies move towards playlist personalization, but personalization is always referred to *what* the user listens to, but never considers *when* or *how* the user listens to music. The result in a partial personalization that does not fully adapt to the user because it does not understand how the listening habits vary during the day, nor does it understand what the user's preferred listening path is.

In this theoretical framework, our method introduces an important contribution: it shows how to generate highly personalized playlists starting from the user's listening history. In particular, it shows that:

- The listening history contains important information not only on *what* the user listens to, but also on *when* and *how* she listens to music. It also allows understanding the user's propensity to listen to new music.
- The transformation of songs into low-level audio features and the use of clustering algorithms allows understanding the user's musical tastes.
- Dynamic programming allows sorting a sequence of songs to optimally adapt the playlist to *how* the user listens to music.

In addition to the theoretical contribution, our method also has practical implications as it used easily implementable techniques; therefore, it might be used by music streaming platforms to provide customers with highly personalized playlists.

There are some limitations that are worth pointing out however. First of all, the listening history provided by Spotify contains only the songs played by the user for at least thirty seconds and does not contain information about songs skipped by the user. A second limitation still concerns the listening history that does not provide information on the likes the user gives to songs and/or artists and does not provide details on the artists followed by the user on the Spotify platform. Having this information (which is available at the platform side, but is not made accessible to external developers) would allow understanding what a user likes and does not like; hence, it would allow improving our method when generating the set of songs (step 1). If this data were made available, future studies might be related to the use of this data in our method.

6 CONCLUSIONS

Music streaming platforms and recent studies are proposing approaches to personalize playlists and the contribution of this paper falls into this field. The idea is to avoid users the boring and tedious task of music/playlist selection. First, we analyzed the theoretical framework and we observed that personalization is usually achieved in a partial way as it is based on *what* the user listens to and not on *when* she listens to music or *how* she listens to music. Our method follows the personalization path and uses the simple user's listening history to achieve a high level of personalization. It uses metadata to understand the propensity to listen to new music, it transforms songs into low-level audio features, it uses clustering algorithms and it interacts with the Spotify's music recommender system to generate sets of songs similar to the ones liked by the user; it uses dynamic programming to actually produce playlists (i.e., it orders the set of music in an optimal way with respect to *how* the user listens to music). The experimental results showed that the proposed method generates playlists that are more personalized with respect to the ones produced by other methods. To the best of our knowledge, this is the first method that allows generating highly personalized playlists with respect to *what, when* and *how* a user listens to music.

REFERENCES

[1] Luis Aguiar and Joel Waldfogel. 2018. Platforms, Promotion, and Product Discovery: Evidence from Spotify Playlists. JRC Digital Economy Working Paper 2018-04. JRC Technical Reports, JRC112023 (May 2018).

[2] Marcos Alves de Almeida, Carolina Coimbra Vieira, Pedro Olmo Stancioli Vaz De Melo, and Renato Martins Assunção. 2019. Random Playlists Smoothly Commuting Between Styles. ACM Trans. Multimedia Comput. Commun. Appl. 15, 4, Article 104 (Dec. 2019), 20 pages. https://doi.org/10.1145/3361742

[3] Andreja Andric and Goffredo Haus. 2006. Automatic Playlist Generation Based on Tracking User's Listening Habits. Multimedia Tools Appl. 29, 2 (June 2006), 127–151. https://doi.org/10.1007/s11042-006-0003-9

[4] Vinayak Bali, Shubham Haval, Snehal Patil, and R. Priyambiga. 2019. Emotion Based Music Player. Journal of Software Engineering & Software Testing 4, 1 (April 2019), 18– 30. https://doi.org/10.1109/ICEAST.2019.880255

[5] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. ACM Comput. Surv. 47, 2, Article 26 (Nov. 2014), 35 pages. https://doi.org/10.1145/2652481

[6] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. 3654–3660. https://doi.org/10.24963/ijcai.2017/511

[7] Sally Jo Cunningham, David Bainbridge, and Annette Falconer. 2006. More of an Art than a Science: Supporting the Creation of Playlists and Mixes.. In Proceedings of 7th International Conference on Music Information Retrieval (2007-01-05). 240–245.

[8] Ricardo Dias, Daniel Gonçalves, and Manuel J. Fonseca. 2017. From manual to assisted playlist creation: a survey. Multimedia Tools and Applications 76, 12 (01 Jun 2017), 14375–14403. https://doi.org/10.1007/s11042-016-3836-x

[9] Maria Eriksson. 2020. The editorial playlist as container technology: on Spotify and the logistical role of digital music packages. Journal of Cultural Economy 13, 4 (2020), 415–427. https://doi.org/10.1080/17530350.2019.1708780

[10] David Erlandsson and Jomar Perez. 2017. Listening Diversity Increases Nearly 40 Percent on Spotify. Spotify Insights (November 2017).

[11] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. 2008. Playlist Generation using Start and End Songs. In Proceedings of the 9th International Conference on Music Information Retrieval. 173–178.

[12] Eamonn Forde. 2017. They could destroy the album': how Spotify's playlists have changed music for ever. The Guardian (August 2017).

[13] Marco Furini, Filippo Geraci, Manuela Montangero, and Marco Pellegrini. 2010. STIMO: STIll and MOving video storyboard for the web scenario. Multimedia Tools and Applications 46, 1 (January 2010), 47–69. https://doi.org/10.1007/s11042-009-0307-7

[14] M. Furini, J. Martini, and M. Montangero. 2019. Automated Generation of User-Tailored and Time-Sensitive Music Playlists. In 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC). 1–6. https://doi.org/10.1109/CCNC.2019.8651820

[15] Filippo Geraci, Mauro Leoncini, Manuela Montangero, Marco Pellegrini, and M. Elena Renda. 2009. K-Boost: A Scalable Algorithm for High-Quality Clustering of Microarray Gene Expression Data. Journal of Computational Biology 16, 6 (2009), 859–873. https://doi.org/10.1089/cmb.2008.0201

[16] A. Germain and J. Chakareski. 2013. Spotify Me: Facebook-assisted automatic playlist generation. In 2013 IEEE 15th International Workshop on Multimedia Signal Processing (MMSP). 025–028. https://doi.org/10.1109/MMSP.2013.6659258

[17] Derek L. Hansen and Jennifer Golbeck. 2009. Mixing It up: Recommending Collections of Items. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, 1217–1226. https://doi.org/10.1145/1518701.1518883

[18] Shobu Ikeda, Kenta Oku, and Kyoji Kawagoe. 2017. Analysis of music transition in acoustic feature space for music recommendation. In Proceedings of the 9th International Conference on Machine Learning and Computing. 77–80. https://doi.org/10.1145/3055635.3056602

[19] I. Kamehkhosh, G. Bonnin, and D. Jannach. 2020. Effects of recommendations on the playlist creation behavior of users. User Modeling and User-Adapted Interaction 30 (2020). https://doi.org/10.1007/s11257-019-09237-4

[20] Peter Knees and Markus Schedl. 2013. A Survey of Music Similarity and Recommendation from Music Context Data. ACM Trans. Multimedia Comput. Commun. Appl. 10, 1, Article 2 (Dec. 2013), 21 pages. https://doi.org/10.1145/2542205.2542206

[21] N. Lin, P. C. Tsai, Y. A. Chen, and H. H. Chen. 2014. Music recommendation based on artist novelty and similarity. In IEEE International Workshop on Multimedia Signal Processing. 1–6. https://doi.org/10.1109/MMSP.2014.6958801

[22] Benjamin A. Morgan. 2020. Revenue, access, and engagement via the in-house curated Spotify playlist in Australia. Popular Communication 18, 1 (2020), 32–47. https://doi.org/10.1080/15405702.2019.1649678

[23] M. Pichl, E. Zangerle, and G. Specht. 2016. Understanding Playlist Creation on Music Streaming Platforms. In 2016 IEEE International Symposium on Multimedia (ISM). 475–480. https://doi.org/10.1109/ISM.2016.0107

[24] Martin Pichl, Eva Zangerle, Günther Specht, and Markus Schedl. 2017. Mining Culture-Specific Music Listening Behavior from Social Media Data. In IEEE International Symposium on Multimedia. IEEE Computer Society, 208–215. https://doi.org/10.1109/ISM.2017.35

[25] Andy Sarroff and Michael Casey. 2012. Modeling and Predicting Song Adjacencies in Commercial Albums. In Sound and Music Computing Conference. https://doi.org/10.5281/zenodo.850114

[26] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. International Journal of Multimedia Information Retrieval volume 7 (2018). https://doi.org/10.1007/s13735-018-0154-2

[27] B. Schwartz. 2009. The Paradox of Choice: Why More Is Less, Revised Edition. HarperCollins.

[28] Arnaja Sen, Dhaval Popat, Hardik Shah, Priyanka Kuwor, and Era Johri. 2018. Music Playlist Generation Using Facial Expression Analysis and Task Extraction. In Intelligent Communication and Computational Technologies. Springer Singapore, Singapore, 129–139. https://doi.org/10.1007/978-981-10-5523-2_13

[29] Andreu Vall. 2015. Listener-Inspired Automated Music Playlist Generation. In Proceedings of the 9th ACM Conference on Recommender Systems (Vienna, Austria) (RecSys '15). 387–390. https://doi.org/10.1145/2792838.2796548

[30] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. 2014. Exploration in Interactive Personalized Music Recommendation: A Reinforcement Learning Approach. ACM Trans. Multimedia Comput. Commun. Appl. 11, 1, Article 7 (Sept. 2014), 22 pages. https://doi.org/10.1145/2623372

[31] Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. 2019. An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. ACM Trans. Intell. Syst. Technol. 10, 5, Article 57 (Sept. 2019), 21 pages. https://doi.org/10.1145/3344257