ISSN:0975-9646

Prateek Singh et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (5) , 2015, 4159-4162

Predictive Music Shuffling Algorithm

Prateek Singh, Aditya Batheja, Abhijit Chowdhury

CSE Department,
Guru Gobind Singh Indraprastha University
Sector-16C, Dwarka, New Delhi, Delhi 110078, India

***Abstract*— Shuffling is a method employed to randomize a data set to ensure an element of chance when elements from the data set are selected at random. Literature has established that computers are capable of generating an ideal shuffle; a bias-free random permutation of cards. Shuffling is also widely used in music players as a separate feature allowing listeners to randomize their playlists. However, an ideal or bias-free shuffle may prove to be counter-intuitive in this case as even in randomized selections, listeners tend to have inclinations towards a certain type of songs depending on their mood. In this paper, we propose a predictive shuffling algorithm that can provide automated dynamic based shuffling according to the user's preferences. This shuffling takes into account various parameters like genre, artist, play duration and release date and selects the next song on that basis.**

Keywords—Music Shuffling, Predictive Music Shuffling

I. INTRODUCTION

Shuffling or randomization is a sequencing of random variables describing a process whose outcomes do not follow a deterministic pattern, but follow an evolution described by probability distributions. A shuffling process looks to produce a random permutation of objects. This process, however, may not be completely haphazard. Randomization, although, looks to make a completely random selection but in certain cases, it needs to make more intelligent selections.

Randomization finds a lot of application in random experiments, gambling, song-shuffling, statistics and survey sampling. Most music players uses a minimal randomization algorithm known as Fisher-Yates algorithm. Fisher–Yates shuffling is similar to randomly picking numbered tickets out of a hat without replacement until there are none left. However, this does not account for any intuitive approach. Owing to the ever-increasing consumption of music and its varied availability, there is a dire need of a shuffling algorithm which works based on user's preferences.

The proposed approach works on the ID3 tags of a sound track. ID3 tags include artist, album, genre, song release date. Based on these, 'nearness' factor of all the songs in the music library are calculated respective to the song first played by the user. The higher the value of the nearness factor, higher are its chances of being played next.

II. COUNTER-INUITIVNESS OF NORMAL SHUFFLING

Consider a music library consisting of almost equals numbers of sound tracks pertaining to three different genres. Let there be 10 tracks of genre A, 11 tracks of genre B and 11 tracks of genre C. A simple unbiased random algorithm, when employed, would result in randomized data set such as one below:-

AACBBCBACABBCCACCCCABBACBACABABB

There are two shortcomings with this approach.

1. In the middle of the dataset, there are four tracks of genre C simultaneously placed. In case, the user does not want to listen to this particular genre, he has no choice except for skipping each of these tracks one by one. Thus, this approach is static.
2. Otherwise in the dataset, the player shifts from one genre to another without any relevance to user's activity. There is no way to know what genre the user wants to listen to more at this time.

Thus, the only shuffled patterns observed by the user using this approach are due to confirmation bias.

2.1 *Predictive Shuffling- Nearness Factor*

This approach requires forming links between different songs in the library. The mechanism used for forming such associations is called 'Nearness' factor. It is calculated based on the similarity in ID3 tags of the songs in the library to the song being played currently. ID3 tags consists of

- Artist
- Album
- Genre
- Release Date(whether same decade or not)

The nearness factor is assigned based on 1 or 2-point increment to the default value of 0 corresponding to each matched tag of the two songs. The songs with a high value of nearness factor are played first. The nearness factor is calculated based on the weight generator algorithm. The time for which the current song is played also influences whether the ones similar to this would be played next or not. Following algorithm is used to calculate the nearness factor.

Weight_Gen(currently_playing)

1. song.weight=0
2. If currently_playing.artist == song.artist
3. song.weight +=1
4. If currently_playing.album == song.album
5. song.weight +=1

6. If currently_playing.genre == song.genre
7. song.weight +=2
8. If currently_playing.decade == song.decade
9. song.weight +=1

2.2 Predictive Shuffliing-Interleaving Nearness factor with played duration

The algorithm depends on the first song being chosen by the user in order to calculate the 'nearness' of other songs to the chosen song. The duration for which the current track is played is directly related to its likability by the user. If the first song is changed immediately, another random song is played. If the song is played for a minimum time say t1, weight generator is called. A point in duration of the song is referred to as t2. t2 can be referred to as the time for which if the song is played succinctly established that it is liked by the user. If the duration crosses t2 and user requests a new song, the next song would be the one most similar to the song played.

If the duration is less than t2, there is a logic in place to assure that the user does not listen to the same kind of songs. The user can request new songs repeatedly and if the number of requests exceed the weight of the song last played, the algorithm plays the next song with a weight one less than the last played weight. This means that the song is little less similar to the earlier songs. For instance, such a song can be of the same genre, album and decade but a different artist, or a same artist, genre and decade but a different album. This allows the listener to listen to varied tracks. The algorithm recursively moves forward.

Function Main(song_from_user)

t - time when user made the song change request
t1 - time limit for immediate change
t2 - time limit for next change
current_song = the song being currently played
song_to_play = song that will be played next

1. If (change request is made at t<t1)
   song_to_play = Random_Play()
2. Else
3. Weight_Gen(current_song)
4. End-if
   If change request is made at t1<t<t2
5. song_to_play =Average_Play();
6. Else
7. If change request is made at t>t2 :
   song_to_play =Priority_Play();
8. End-if
9. End-if

Function Average_Play()

play_next_song = number of time next was pressed for current weight songs
current_weight = weight of currently selected song
songs_in currentweightpool = number of songs in current weight pool
1.play_next_song++;
2. If (play_next_song < current_weight)
3. If (songs_in_currenweightpool > 0)
4. Pool_Play(current_weight);
5. Else
6.current_weight--;
7.play_next_song=0;
8. If (current_weight equals 0)
9. Random_Play();
10.Else
11. If (songs_in_currenweightpool > 0)
12. Pool_play(current_weight);
13. Else
14. Average_Play();
15. End-if
16. End-if
17. End-if
18. Else
19. current_weight--;
20. play_next_song=0;
21. Pool_Play(current_weight-1);
22. If(current_weight equals 0)
23. Main(last_song_played);
24. Else
25. If (last_song_played-for<complete_duration)
26. Average_Play();
27. End-if
28. End-if
29. End-if

Function Random_Song()

1.Return any song from song_array();

Function Priority_Play()

1.If (play_next_song < current_weight)
2. If (songs_in_currenweightpool > 0)
3. Pool_Play(current_weight);
4. Else
5. current_weight--;
6. play_next_song=0;
7. If (current_weight equals 0)
8. Random_Play();
9.Else
10. If (songs_in_currentweightpool > 0)
11. Pool_play(current_weight);
12. Else;
13. Ammplay();
14. End-if
15. End-if
16. End-if
17. End-if

Function Pool_Play(int passed_weight)

1.song_array[] = array containing the complete user collection
2. Return any song from song_array[] whose weight=passed.weight;

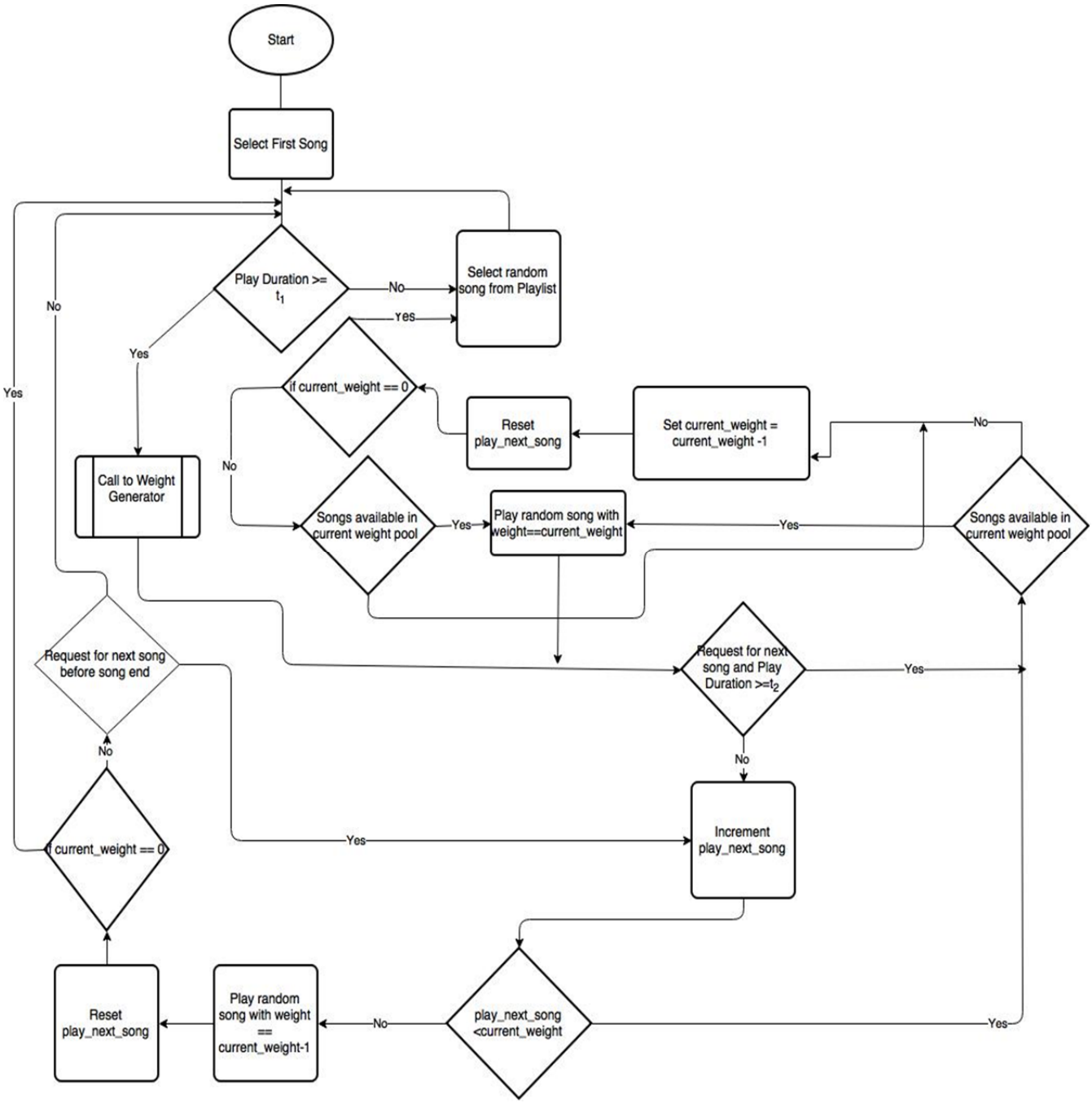

Fig. 1 Flow Chart Of Predictive Shuffling Algorithm

III. Results And Conclusions

A predictive shuffling algorithm is proposed for an individual's music library. The algorithm tends to iterate through the collection based on the user's likings of the song being currently played. This is established through a concept of Nearness factor which links all the songs to the current song based on the comparison of the ID3 tags. Further, it is realized that the time for which any song is played by the user is also an indication how well it is deemed by the user. Incorporation of the time along with the nearness factor renders a unique approach to the shuffling of the songs in the library.

IV. Future Scope

As per the current proposition of the algorithm, the time thresholds t1 and t2 are indicative of how much a song is liked by listener. These values should be quantified in a unique manner for every song track. The weight assigned to the track based on ID3 tracks can be made for dynamic. This can be achieved using Machine Learning and Fuzzy logic in order to make the algorithm more qualitative and data-driven. Another important factor can be the ability of the user to like or dislike a given track. This can then be incorporated into the algorithm to inlfuence the song's weight.

REFERENCES

[1] McFee, Brian (2012), Machine learning approaches to music similarity, Retrieved from http://escholarship.org/uc/item/8s90q67r
[2] Music Genome Project http://www.pandora.com/about/mgp
[3] Fisher, Ronald A., Yates, Frank (1948) , Statistical tables for biological, agricultural and medical research (3rd ed.)