RUFFLE: A USER-CONTROLLABLE MUSIC SHUFFLING ALGORITHM

Giorgio PRESTI (giorgio.presti@unimi.it) (0000-0001-7643-9915)[1],
Federico AVANZINI (federico.avanzini@unimi.it) (0000-0002-1257-5878)[1],
Adriano BARATÈ (adriano.barate@unimi.it) (0000-0001-8435-8373)[1],
Luca Andrea LUDOVICO (luca.ludovico@unimi.it) (0000-0002-8251-2231)[1], and
Davide Andrea MAURO (maurod@marshall.edu) (0000-0001-8437-4517)[2]

[1] *Laboratorio di Informatica Musicale (LIM), Department of Computer Science*, **Università degli Studi di Milano**, Italy
[2] *Department of Computer and Information Technology*, **Marshall University**, USA

ABSTRACT

Music shuffling is a common feature, available in most audio players and music streaming platforms. The goal of this function is to let songs be played in random, or constrained random, order. The results obtained by in-use shuffling algorithms can be unsatisfactory due to several factors including: the variability of user expectations to what constitutes a "successful" playlist, the common bias of being unable to recognize true randomness, and the tendency of humans to find nonexistent patterns in random structures. In this paper, a new shuffling algorithm called *Ruffle* is presented. *Ruffle* lets the user decide which aspects of the music library have to be actually shuffled, and which features should remain unchanged between consecutive extractions. First, an online survey was conducted to collect users' feedback about the characteristics used for shuffling. It is worth noting that, in general, the algorithm could address any metadata and/or audio extracted feature. Then, in order to test the algorithm on personal playlists, a Web version based on Spotify API has been released. For this reason, a second survey is marking an ongoing effort placed on validating the effectiveness of the algorithm by collecting users' feedback, and measuring the level of user satisfaction.

1. INTRODUCTION

The shuffle feature of a music player is a well known function that lets the user listen to each song of the personal catalog in a pseudo-random order. In [1] the shuffle mode is defined as "a control that paradoxically involves a renunciation of control on the part of the user".

People may use such a function for many reasons, including the reduction of boredom in listening, the need to keep the attention alive, the search for "serendipity" [2], etc. For a general overview of shuffle in music, see [3, 4].

As shown in [5], humans are hardly capable not only of identifying, but also of generating random sequences of numbers. This is mainly due to two phenomena: i) the *gambler fallacy* [6], that consists in thinking that, in a memory-less random draw, previous drawings have an influence on the actual ones, and ii) the *clustering illusion* [7], that consists in erroneously considering small clusters of samples from random distributions to be non-random.

An ambiguity when talking about the randomness of a playlist in natural language is whether to compute the order of the songs with or without regard of their metadata. In the latter case, music pieces can be seen as completely different objects drawn at random, even if they may share some common values for metadata; in the former case, the expected behavior is to travel the song space with the longest possible path, such that the distance of each consecutive song is maximized with respect to some properties. This can be seen as an interesting variant of the Travelling Salesman Problem, which is, in turn, a special case of the Longest Path Problem.

In order to manage the complex and multifaceted problem of how to shuffle songs in a playlist, a new algorithm, called *Ruffle*, has been developed. *Ruffle* generates a pseudo-random sequence that matches the user's preferences about what must be shuffled, and to what extent. It does so by exploiting the gambler fallacy to present the problem to the users, which makes their choices, that in turn (together with what has been previously drawn) will affect the probability of the remaining songs in the list to be drawn for the next play.

The rest of the paper is organized as follows: Section 2 provides the state of the art about industry standards and commercial applications integrating a shuffle function; Section 3 presents a survey designed to understand user preferences about playlist shuffling, also discussing the collected results in order to guide the design of a new approach; Section 4 focuses on *Ruffle*, a user controllable shuffle algorithm; Section 5 provides an early assessment of *Ruffle*, presenting a web-based implementation that interfaces with Spotify API, and the outcomes of a survey where participants tested its functionalities in an actual scenario; finally, Section 6 draws the conclusions.

2. STATE OF THE ART

Before presenting the state of the art, it is worth remarking the distinction between a shuffle function and a music recommendation system. A shuffle algorithm simply re-

orders a given list of songs already available in the user's collection. This operation can also be performed by taking into account some preferences about the desired outcome, but no user data out of the scope of a listening session is involved. Conversely, a recommendation (or suggestion) system aims at proposing content that is not necessarily already in the user's collection [8]. Moreover, this task is typically accomplished by modeling user's preferences and making inferences about musical tastes based on long-term behaviors. Some algorithms may combine the two approaches, e.g. by changing the shuffle strategy based on user's actions such as song skips, implicitly interpreted as a manifestation of dislike towards a particular genre or artist. For the sake of clarity, this paper deals with the shuffle problem only, without making any inference.

A number of algorithmic approaches can be used to produce a randomly shuffled sequence of elements in a list.

The Fisher-Yates shuffle, whose original form dates back to 1938, represents a way for generating a random permutation of a finite sequence. The algorithm puts all the elements into an unordered set, i.e. a container that stores unique elements in no particular order, and establishes the next element in the sequence by randomly drawing one of them from the set until no elements remain. The algorithm produces an unbiased permutation, i.e. every permutation is equally likely. First described in [9] and originally conceived as a pencil-and-paper method, its computer implementation was documented in [10] and later published in [11]. A critical issue of the algorithm is that one of its steps requires to pick a random number, but only a high-quality unbiased random number source can guarantee unbiased results. The Fisher-Yates algorithm is at the base of the implementation proposed in [12].

The application domain represented by a music collection introduces some additional requirements. In fact, single music pieces often share some characteristics with others, and a good shuffle algorithm could try to maximize the distance considering also such similarities. For example, multiple songs in a dataset could be authored and/or performed by the same artist(s), could belong to the same genre, could share the same ensemble, etc.

This problem has been addressed in [13], where a form of *balanced shuffle* is proposed. The idea is to merge pre-shuffled playlists, each one made of pieces belonging to the same group, where a group contains elements sharing similar properties as it regards a given dimension. For the sake of clarity, let the chosen dimension be the genre: a group contains all classical pieces, another group all jazz pieces, and so on; each sub-playlist is shuffled; finally, a merge-and-mix operation is conducted to transform pre-shuffled sub-playlists into a single playlist.

Singh *et al.* [14] proposed a form of *predictive shuffling* that can provide automated dynamic-based shuffling according to the user's preferences by taking into account various parameters (e.g. genre, artist, play duration and release date) and selecting the next song accordingly.

An original approach described in the literature is so-called *responsive shuffling* [15], that benefits from temporal, spatial, and mental context awareness. Based on interactive soundscape concepts and wearable-computing technologies, this work proposes context-driven playlist shuffling for music listening in mobility.

Concerning documented shuffle functions in commercially-available services, Spotify [1] initially started from the Fisher-Yates algorithm, then evaluated the balanced shuffle described above, and finally moved to a method which is claimed to be inspired by image dithering [16].

Pandora [2] implements different forms of shuffle depending on user's privileges: premium subscribers have more flexibility with their shuffle options and can shuffle songs as well as the content on playlists or stations. The set of music features that can be considered comes from Pandora's *Music Genome Project*, an effort to "capture the essence of music at the most fundamental level" [17] employing over 450 attributes, called *genes*, to describe songs and a complex algorithm to organize them.

Although some of the shuffling algorithms mentioned above implement advanced features (such as considering a subset of metadata to vary or, conversely, to keep fixed among consecutive draws), such a mechanism is usually hidden from the user, with little to no room for controls, so the results may not match personal preferences.

3. TUNING TO LISTENERS' PREFERENCES

3.1 Survey design

In order to understand in detail users preferences and desiderata about the shuffle listening experience, an online survey was carried out.

The survey is made of three parts. The first part collects general information about the user such as the average daily listening time, the preferred player/platform, how often a shuffle function is used when organizing a playlist, and the level of satisfaction with the shuffle behavior.

The second part consists of an open question inviting users to reflect on which, according to their opinion, should be the aspects used in constraining a shuffle algorithm (e.g. preserving the same tempo, or genre), and what constitutes a "successful" playlist.

Finally, users are requested to express their preferences regarding specific parameters: 1. genre, 2. artist, 3. album, 4. BPM, 5. language, 6. publication date, 7. instrumental/-song, and 8. allowing repetitions.

Further comments are allowed as free text in the last page before submission. Surveys have been conducted in Italian and they are here translated to English.

3.2 Survey results and discussion

3.2.1 Users overview

In total, 84 answers were collected from Italian users. The subjects are 56% males, 42% females, and 2% not specified; 64% were 17-24 years old, 13% in the range 25-30, 15% in range 31-50, and 8% > 50.

Listening habits: 11% listens to less than 1 hour of music per day, 44% listens to 1-2 hours, 24% 2-3 h, and 21%

[1] https://www.spotify.com
[2] https://www.pandora.com

more than 3 h. Most used listening platforms are Spotify, Youtube, and personal libraries, in particular the preferred combinations were Spotify+Youtube 37%, Spotify only 23%, Youtube only 6%, and Spotify+Youtube+Personal 5%; the remaining 29% uses a variety of combination of the above services and others, such as, in preference order, Amazon Music, Apple Music, Google Play Music, Deezer, TIDAL, web or analog radio, Bandcamp.

About the use of the shuffle function, 29% rarely uses it, 18% sometimes, and 53% frequently. 24% are not satisfied by the shuffle function, 43% are neither satisfied nor unsatisfied, and 33% are satisfied.

Use of shuffle correlates weakly with age (spearman $R = 0.3$, $p < 0.005$), in particular almost all subjects with age > 50 rarely uses shuffle, while subjects with age < 50 present more heterogeneous behavior.

3.2.2 Open question about shuffling

Concerning the open question, the 84 participants provided a number of different comments. A manual clustering revealed three wide categories: [3]

- Randomness (21 comments);
- Music properties (48 comments);
- Suggestion systems (28 comments).

The comments regarding randomness may be summarized by the following 4 sentences:

R1 *All available songs should be shuffled as randomly as possible, without distinctions* (7 comments);

R2 *A song should not be re-played until all songs have been played* (5 comments);

R3 *Shuffle should produce very different sequences across listening sessions* (8 comments);

R4 *Shuffle outcome should be stored in case the listener is interested in navigating the playlist* (1 comment).

In regard to R1, Fisher-Yates should be the preferred choice, unfortunately none of the 7 users really meant it, since when asked whether a feature may be kept constant or variable across plays, or whether the feature must be irrelevant, almost all answered the artist and album must vary, the genre must stay constant, and they left only BPM, language, and publication year as actually random. This is no surprise, since it has already been discussed how humans, on average, have no precise insight on how randomness behaves.

Even if R2 may seem obvious, when explicitly asked about this aspect, not all subjects agreed (more on this in Sec. 3.2.3).

Note that R3 may seem to suggest something different from the equiprobability of the sequences that make Fisher-Yates an adequate algorithm, nevertheless, if we interpret this as an extension of the previous one (i.e. "A song should not be re-played until all songs have been played, *even across different sessions*"), then Fisher-Yates is still a valid approach.

The behavior described in R4 is usually implemented in modern platforms as "user history", or directly by design when the shuffle function works offline, and returns a playlist instead of one-by-one songs.

The comments regarding music properties may be summarized by the following 6 sentences:

M1 *Shuffle should be aware of music properties* (48 comments). In particular, the following were cited: Genre (19 mentions); Artist (10 mentions); Generic "properties" (9 mentions); BPM (4 mentions); Mood (3 mentions); Album (2 mentions); Key (2 mentions); Release year (1 mention); Song duration (1 mention); Instruments (1 mention); Timbre (1 mention).

M2 *All proposed songs should be strictly coherent in terms of one or more properties* (17 comments). In particular, it was specified in 3 comments that a user-selected song should set the baseline;

M3 *Some incoherence can be tolerated in order to avoid monotony* (4 comments);

M4 *Changes of music properties should be gradual* (8 comments);

M5 *Consecutive songs should be very different in terms of one or more properties* (4 comments);

M6 *It should be possible to choose between "coherent" and "incoherent" behavior* (2 comments).

Properties frequently cited in M1 confirm the validity of the items selected for the multiple choices version of the question (see Sec.3.2.3), except for the mood, which was not considered. The 9 mentions of some "generic property" may be implemented by adding audio features as well as metadata in the algorithm, nevertheless, not all players support this level of detail. It should be noted that some of these sentences are in contrast with each other (e.g. M2 and M5), thus a good shuffling feature should be tunable, as suggested in M6. Finally, M4 implies that it is worth considering the introduction of some sort of "memory" in the shuffle algorithm.

The comments regarding recommendation systems may be summarized by the following 6 sentences:

S1 *Priority should be given to frequently listened songs* (5 comments);

S2 *Some inference about musical taste is expected* (9 comments);

S3 *Some music discovery algorithm is expected* (4 comments);

S4 *Song skipping should be considered in future draw* (7 comments);

[3] Number of comments may add up to more than number of participants, since many participants provided multiple comments; moreover, also the expanded views of the clusters may add up to more than the total cluster comments count, since some comments are two-folded, e.g. comment "Artist and genre may stay constant" expresses an interest both in the "artist" and "genre" metadata, and the fact that the shuffle may produce coherent outcomes accordingly.

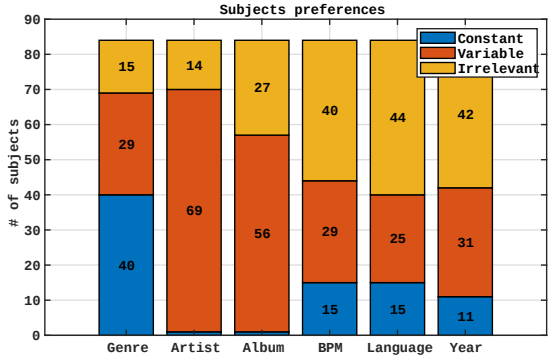


Figure 1: Hypothetical preferences expressed by subjects.

S5 *Context information (such as current time and geolocation) should be considered* (2 comments);

S6 *The algorithm should teach something to the user* (1 comment).

Sentences like S1-3 and S5 will be ignored, since they must rely on user information and recommendation algorithms.

The feature described in S4 is somehow borderline between shuffling and suggestion, it may be worth considering it in future works, and let this work focus on pure shuffling.

S6 is also related to a recommendation algorithm, nevertheless it provides a hint toward a shuffle algorithm that lets you pick a path in the songs feature space. Yet another feature that can be explored in the future.

3.2.3 Multiple-choice questions about shuffling

As a first question, subjects were asked to tell, for each metadata (genre, artist, album, BPM, language, and release year) if that should remain constant among plays, if it should vary, or if should be irrelevant. Figure 1 shows an overview of the answers. The sum of "constant" and "variable" preferences has been interpreted as a score of the wish to have control over it. All metadata received a score greater or equal to 50%, except for language, which is slightly less than 50%.

In more detail, 49 unique combinations of answers were given: 31 combinations were selected only once, 11 combinations appeared twice, and only the remaining 7 combinations were selected three or more times (these are visible in Table 1). This considerable variability, with some frequent choices, is two-folded: on one side the presence of frequent choices can suggest that some template behavior for shuffle algorithms may be useful, but on the other side, to meet the needs of most users, a fine tuning mechanism seems desirable.

To further investigate if a pattern is present when tolerating some small differences in the answers, answers to this question were clustered using linkage hierarchical clustering method. Specifically, the hamming distance was selected to compute distances between pairs of answers, and weighted average distance was used as linkage. The tree was cut so to retain most of the groupings of Table 1 without creating neither many small clusters nor large heterogeneous clusters. The resulting cluster's medioids are visible in Table 2.

Clustering revealed that most of the answers appearing only once can be considered very similar to the frequently given combinations, especially 1,2, and 7. The only new cluster (8) can be considered as a truly random version of 2, and one answer (with ID 9) resulted to be so different from the others to be considered the only true outlier.

In conclusion, 92% of the answers can be traced back to the 7 most populated clusters, and in particular clusters 1 and 2 captures almost half the total preferences.

Regarding the possibility to influence the shuffling algorithm by limiting the reproduction of vocal or instrumental tracks, 75% of users agreed.

On the possibility of repeating the same song twice in the same listening session, 79% of answers were negative, 15% and 5% of users considered it acceptable only after a long time or even after a short time, respectively, and the remaining 1% considered this feature to be irrelevant.

3.3 Lessons learnt

The survey highlighted a set of *desiderata* for the shuffling algorithm, which can be synthesized as follows:

D1 Songs should not be replayed until the whole song list has been played;

D2 Available properties such as metadata and audio features should be considered;

D3 A tuning mechanism between constant and variable behavior should be provided;

D4 The system is supposed to have a short-term memory to ease gradual changes in properties;

D5 There should be the ability to seed the algorithm by picking a reference song.

Answering D1 and D5 is trivial, while D2 and D3 are at the core of *Ruffle*. D4 has been implemented by introducing a *memory parameter* β which acts on the *responsiveness* of the system.

4. THE *RUFFLE* ALGORITHM

The rationale behind *Ruffle* is to update after each draw the probability of other songs to be drawn, according to songs properties and user settings. In particular, a similarity δ between the drawn song and the remaining ones is calculated for each considered property, and a probability weight is associated to the songs, which is either directly or inversely proportional to δ based on user decision.

4.1 Variables

Let $\boldsymbol{x}_N$ be a set of N songs:

$$\boldsymbol{x}_N = \{x_1, x_2, \cdots, x_n \mid 0 < n \leq N\}\,; \qquad (1)$$

ID	Genre	Artist	Album	BPM	Lang.	Year	Count	Mnemonic
1	V	V	V	V	V	V	8	Forced randomness
2	C	V	V	I	I	I	6	Genre exploration
3	I	I	I	I	I	I	4	True randomness
4	V	V	V	I	I	I	4	Enhanced randomness
5	C	V	V	V	C	I	3	Cultural niche
6	V	V	V	V	I	V	3	Tolerant randomness
7	V	V	V	C	I	C	3	Memorabilia DJ

Table 1: Most selected combinations. Some mnemonic names are given (V: Variable; C: Constant; I: Irrelevant).

ID	Genre	Artist	Album	BPM	Lang.	Year	Count	Mnemonic
1	V	V	V	V	V	V	19	Forced randomness
2	C	V	V	I	I	I	21	Genre exploration
3	I	I	I	I	I	I	7	True randomness
4	V	V	V	I	I	I	9	Enhanced randomness
5*	C	V	I	V	C	I	6	Refined cultural niche
6	V	V	V	V	I	V	5	Tolerant randomness
7	V	V	V	C	I	C	11	Memorabilia DJ
8	C	I	I	I	I	I	5	Genre strolling
9	C	I	V	C	I	V	1	Genre DJ

Table 2: Cluster's medioids. Some mnemonic names are given (V: Variable; C: Constant; I: Irrelevant).

let $\boldsymbol{c}_L$ be an execution queue of $L \leq N$ songs, initially empty:

$$\boldsymbol{c}_L = \{c_1, c_2, \cdots, c_l \mid 0 \leq l \leq L\}; \tag{2}$$

let $\boldsymbol{x'}_R$ be a subset of $\boldsymbol{x}_N$, composed of $R \leq N$ remaining songs, initialized as $\boldsymbol{x'}_R \equiv \boldsymbol{x}_N$:

$$\boldsymbol{x'}_R \subset \boldsymbol{x}_N; \tag{3}$$

let x_Δ be the last song inserted in the queue, such that

$$x_\Delta \in \boldsymbol{x}_N \quad \& \quad x_\Delta \notin \boldsymbol{x'}_R \quad \& \quad 0 < \Delta \leq N. \tag{4}$$

let $\boldsymbol{a}_{nM}$ be a set of M attributes associated to a song $x_n \in \boldsymbol{x}_N$:

$$\boldsymbol{a}_{nM} = \{a_{n,1}, a_{n,2}, \cdots, a_{n,m} \mid 0 < m \leq M\}; \tag{5}$$

let $\boldsymbol{s}_M$ be a set of M user-controllable settings, such that each setting $s_m \propto P(a_{n,m} \equiv a_{\Delta,m})$:

$$\boldsymbol{s}_M = \{s_1, s_2, \cdots, s_n \mid 0 < m \leq M \quad \& \quad 0 \leq s_n \leq 1\}. \tag{6}$$

each setting is meant to be set to 0 in order to force variation of the corresponding attribute between two consecutive draws, and 1 to force the attribute not to change. A value of 0.5 emulates random variations of the attribute.

Finally, let $0 \leq \beta \leq 1$ be a *memory* coefficient of the system, indicating to what extent previous draws will influence the next ones; $\beta = 0$ will only consider the comparison between x_Δ and the examined song, while $\beta > 0$ will also consider the previous results, with the limit case of $\beta = 1$, where the comparison is made only with the first song extracted.

4.2 Similarity function

Since all attributes have different nature and units, it is not possible to rely on a single similarity function. Songs attributes can be categorical or scalar values: for example, author and album title are categorical, while BPM, year, and audio features are scalar. In both scenarios, similarity is expected to be either 0 or 1. Specifically, in the former case similarity can be expressed as:

$$a \times b := \begin{cases} 0, & \text{if } a \neq b \\ 1, & \text{if } a = b \end{cases}. \tag{7}$$

In the latter case, scalar values are compared in order to obtain a continuous distance measure, then the discrete similarity value is computed by considering a distance threshold (whose value depends on the attribute type) and setting similarity to 0 if the continuous distance is above the threshold, and to 1 otherwise.

4.3 Weighting function

At each draw, each song $x_n \in \boldsymbol{x}_N$ is paired to a probabilistic weight p_n, so to have a set $\boldsymbol{p}_N$ of values computed as:

$$p_n = \begin{cases} 0, & \text{if } x_n \notin \boldsymbol{x'}_R \\ \beta \cdot p_n + (1-\beta) \cdot \tau_n, & \text{otherwise} \end{cases} \tag{8}$$

with

$$\begin{aligned} \tau_n &= \textstyle\prod_{m=1}^{M} (2 \cdot |s_m + \delta_{n,m} - 1| + \epsilon), \\ \delta_{n,m} &= a_{n,m} \times a_{\Delta,m}. \end{aligned} \tag{9}$$

Here $\delta_{n,m}$ represent the similarity between x_n and x_Δ along the m^{th} attribute, while τ_n can be interpreted as follows.

The most important part in the definition of τ_n is the argument of the product, as it dictates how the attribute a_n influences the probabilistic weight; the term $|s_m + \delta_{n,m} - 1|$ takes the same value of the similarity $\delta_{n,m}$ if the corresponding setting value is $s_m = 1$; it becomes $1 - \delta_{n,m}$ when $s_m = 0$; and it converges to 0.5 regardless of the similarity value as s_m approaches 0.5. The additive term $0 < \epsilon \ll 1$ is a very small value needed to avoid all-zeros in $\boldsymbol{p}_N$, while the multiplication factor of 2 has been introduced in order to produce a neutral contribution for $s_m = 0.5$ and an amplification when needed.

Finally, note that the update of p_n in Eq. (8) is based on the previous value of p_n, with the β parameter acting as a weight influencing the *responsiveness* of the change.

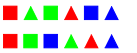
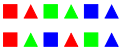
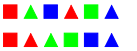
Description	Shape	Color	Output
Random	0.5	0.5	
Alternate	0.0	0.5	
Sort	1.0	0.5	
Interleave	0.0	1.0	
Mix	0.0	0.0	
Group	1.0	1.0	

Table 3: Example of different settings, with 6 colored shapes. β is not considered to simplify the understanding of the base behavior.

4.4 Algorithm and discussion

set $x_\Delta =$ (manual or random choice)
set $\boldsymbol{c}_L = \varnothing$
remove x_Δ from $\boldsymbol{x'}_R$
while $\boldsymbol{x'}_R \neq \varnothing$ && $|\boldsymbol{x'}_R| < L$:
    compute $\boldsymbol{p}_N$
    draw x_α from $\boldsymbol{x}_N$ according to $\boldsymbol{p}_N$;
    add x_α to $\boldsymbol{c}_L$;
    remove x_α from $\boldsymbol{x'}_R$;
    set $x_\Delta = x_\alpha$.

To picture the possible results of this algorithm, consider the case in which *Ruffle* is used to shuffle colored geometric shapes, where the available attributes are color and shape. Starting with a set of 6 elements (3 squares and 3 triangles, forming 3 pairs of colors) the outcomes depicted in Table 3 are possible.

One known drawback of this approach is that all songs that do not manage to fit in the shuffled stream will concentrate at the end of the playlist. This can be resolved by communicating to the user that no more songs are compliant with the settings once the maximum of remaining weights falls under a certain threshold, or can be ignored if the user wants to play the available library completely.

Another possible drawback is that low values of the product terms in τ_n (i.e. close to 0) have a stronger influence when multiplied together than high values (i.e. close to 2). Future implementations will consider a logarithmic type of function for the computation of τ and will assess if it can actually improve the outcomes.

A future evolution could be the generalization of the weight function to non-binary similarity values, in order to take advantage of the continuous nature of scalar attributes and provide more sensitivity to the algorithm.

Finally, a deterministic version of Ruffle is possible, if the draw in the while loop is done by looking for $\text{argmax}_n(\boldsymbol{p}_N)$, nevertheless this case is out of the scope of this paper.

4.5 Possible optimizations

In Eq. (8), and in the pseudo-code above, the computation of $\boldsymbol{p}_N$ is carried out on all $\boldsymbol{x}_N$ songs just for the sake of clarity, of course the computation of $\boldsymbol{p}_N$ can be restricted only to those songs in $\boldsymbol{x'}_R$, without loss in generality.

Beside this, note that since the present work focuses on the validation of the algorithm in its original form, no heuristic optimizations were implemented. Nevertheless some suggestions are provided in the remainder of this section.

Since Eq. (8) is computed ${}^{N^2+N}/_2$ times, the complexity of the algorithm ends up to be $\mathcal{O}(N^2)$. Of course this is not ideal for large music libraries.

In case of $\beta = 1$ (i.e. only the first song is considered), there is no need to compute weights more than once, thus reducing the complexity to $\mathcal{O}(N)$.

Aside from this special case, other improvements may be implemented by storing lookup tables for categorical features such as genre, artist, album etc. even if this does not strictly reduce complexity.

An interesting heuristic may amount to initially shuffle the list with Fisher-Yates, then compute the weights only until a song's weight exceeds a certain threshold. The threshold can be computed as slightly less than the maximum weight possible, which is the product of all $\boldsymbol{s}_M$, in that case the song is picked for next play. If no song reaches the threshold, weights are ready for a regular draw. This heuristic is not ideal for low values of β, since in this case weights are not guaranteed to be up to date.

An alternative to the previous heuristic (assuming that $\beta \ll 1$) is to shuffle the list with Fisher-Yates, then compute the weights only for a fixed number of songs following the one played, and limit the draw in this sliding window.

Finally, consider that real time is not a constraint of the algorithm, since weights can be updated while the user is listening to the song.

5. ASSESSMENT

To assess the validity of *Ruffle* as a usable shuffling algorithm, it has been implemented as a tool to shuffle Spotify playlists, in order to let users evaluate the algorithm before answering a survey.

5.1 Implementation

The tool is based on the Spotify Web API, and is implemented using the vue.js[4] framework. After logging in to their Spotify account, users are able to see the list of their saved tracks, together with a section devoted to load single playlists' content. Using the Web API, for every track the system retrieves 18 properties: 1. track title, 2. artists, 3. album title, 4. genres (associated to artists), 5. release year, 6. duration, 7. key, 8. mode, 9. time signature, 10. acousticness, 11. danceability, 12. energy, 13. instrumentalness, 14. liveness, 15. loudness, 16. speechiness, 17. valence, and 18. tempo. Users can see the values of these properties in the track details. The properties are automatically computed by Spotify, and there is no control or direct knowledge over the algorithms that are implemented for this task.

A sidebar is used to change the settings, i.e. the β and the s_m values, with a set of sliders; a number of presets are available, representing the 5 most selected clusters of Table 2, together with 2 configurations never chosen in previous tests.

[4] https://vuejs.org/

After choosing a preset or manually adjusting single values, users can run the *Ruffle* algorithm, obtaining a reordering in the list of tracks.

The re-ordered tracks can be also saved in a new playlist.

5.2 Evaluation survey

The survey is composed of 4 main parts: a briefing section, containing information about *Ruffle*, and instructions on how to use the implemented Spotify Playlist Shuffler. Please note that it was explicitly asked to focus the attention to the shuffling outcomes rather than the usability of the prototype, since the latter is not the focus of the paper.

The second part repeats some of the questions of the first survey aimed at describing the sampled population, i.e. sex, age, average daily music listening time, usage of shuffle function and satisfaction about the currently used shuffle.

The third part focuses on general impressions on the *Ruffle* algorithm, asking questions about the usefulness of: the available properties, the algorithm itself beside available properties, and the β parameter. Users were also asked to evaluate how much they liked *Ruffle*, and if they would use it if implemented in a music player.

In the last part users were asked to evaluate how frequently they would use each of the presets present in the prototype (note that the "true randomness" preset can be considered as a baseline, since it is equivalent to Fisher-Yates), they were also asked to enter the preferred settings they experienced, and if they would likely change settings frequently, use a finite (small) set of presets, or just use a "set and forget" approach. Finally they were asked to enter free comments if they had any.

The last two parts were aiming at validating what has been observed in Section 3 and evaluating the acceptance of the *Ruffle* algorithm.

5.3 Results

In total 23 users were tested. This survey and the one described in Section 3 have been administered to two different populations, nevertheless, the distributions of answers to the first section were substantially similar.

The algorithm itself (beside available properties) has been marked as useful 17 times, not useful 1 time, and neither useful nor useless 5 times. Similarly the Beta parameter has been marked as useful 17 times, not useful 1 time, and neither useful nor useless 5 times. Furthermore, 19 users liked *Ruffle* (versus 1 that did not liked the algorithm, and 3 which were neutral about it), and 22 over 23 said they would use it in real applications, thus demonstrating that *Ruffle* is indeed a desirable feature. The properties marked as useful are reported in Table 4, together with the number of votes received.

Properties such as *theme* (intended as "Christmas songs" etc.), *instruments*, *mood*, and *language* were manually inserted by subjects.

This not only provides a detailed view of what has been observed in Section 3, but also refines results in a real world scenario, since answers to the preliminary test were given in an hypothetical scenario. In particular it can be seen which audio features are more interesting for users, sometimes even more than metadata.

Attribute	Votes	Attribute	Votes
Artists	18	Loudness	5
Genres	14	Mode	5
Danceability	13	Time Signature	3
Energy	13	Valence	3
Instrumentalness	13	Duration	2
Album Title	9	Language	2
Release Year	9	Liveness	2
Tempo	7	Instruments	1
Speechiness	6	Mood	1
Acousticness	5	Theme	1
Key	5		

Table 4: Properties marked as useful by *Ruffle* users.

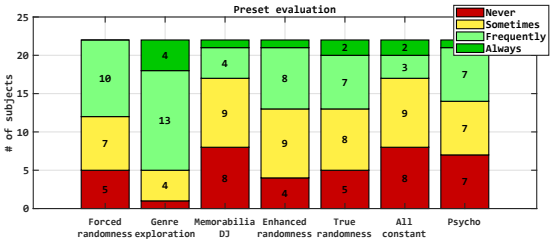


Figure 2: Answers to the question "Would you use this preset in your shuffling sessions?"

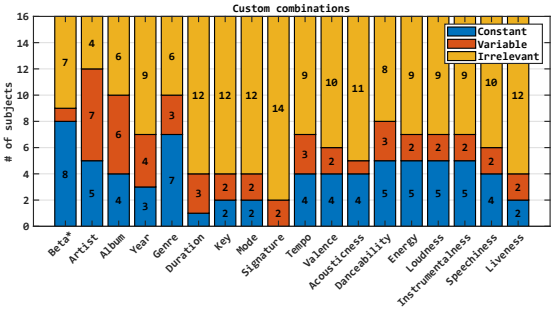


Figure 3: Composition of reported custom presets (for β only, blue ≈ 1, yellow ≈ 0.5, red ≈ 0).

As reported in Figure 2, the most cited configurations of Table 2, when selected by users, revealed to be perceived slightly differently from the expectations. In particular, only "Genre Exploration" clearly outperformed the 2 presets expected to be disliked, which indeed were.

From another perspective, Figure 3 shows how the preferred settings are distributed. This picture highlights how the expressed preferences differed from the "Genre Exploration" preset. This seems to contrast with the outcome of the previous question, but it is worth stressing that the presented presets were working only on 5 principal metadata (Genre, Artist, Album, BPM, and Release Year), while the manual settings were those made available by Spotify and previously listed. This may suggest that, in case of very basic settings, "Genre Exploration" is the most useful approach; nevertheless, when the possibility to finely tune preferences is offered to the user, system customization is a very appreciated feature.

Unfortunately, since the reporting of a preferred personal

setup was optional, only 12 answers were collected, not enough to perform a clustering. Nevertheless, it is interesting to see that all answers were unique, reporting different combinations of preferences.

Concerning the preferred style of settings, the need of only one preset to be "set and forget" has been chosen 5 times, the need for fine tuning each session has been chosen 5 times, and the need for a finite set of custom presets 13 times, thus being the most desirable scenario.

Among free user comments, it is worth mentioning the request to be able to select the first song for the playlist. This aspect, already considered in the discussed version of *Ruffle*, will be soon implemented in the online prototype, too.

Other remarks concerned control values: between 0 and 0.5, they have been perceived as not very useful, and difficult to discern, while they become more significant in the upper part of the scale. In this case, an exponential parameter control may solve the issue: the lower part would be compressed in less space, leaving the upper part more sensitive to changes. In terms of GUI, if the parameter is interpreted as the "amount of homogeneity", a slider in the upper bound position recalls steadiness, while randomness is expected to be closer to the lower bound (with the mid position being considered a midpoint between randomness and steadiness).

6. CONCLUSIONS AND FUTURE WORKS

This paper presented a novel algorithm to shuffle music playlists by giving the user the possibility to configure the dimensions to consider in the calculation of pieces similarity. On each dimension, feature values can be ignored (thus not influencing the process), be distanced as much as possible, or conversely act as piece aggregators.

In order to let the reader test the algorithm, a solution publicly available via web has been released. Such a framework, working on Spotify personal playlists, is available at `http://ruffle.lim.di.unimi.it/`.

Concerning future work, we are planning to perform a fine tuning of the formulas employed for scalar values such as year, tempo, etc. Besides, we aim to implement some features that may improve user's experience, such as the possibility to set the size of the generated playlist (e.g., a shuffled list made of n songs or lasting m minutes).

7. REFERENCES

[1] M. G. Quiñones, "Listening in shuffle mode," *Lied und populäre Kultur/Song and Popular Culture*, pp. 11–22, 2007.

[2] T. W. Leong, F. Vetere, and S. Howard, "The serendipity shuffle," in *Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future*, 2005, pp. 1–4.

[3] D. Powers, "Lost in the shuffle: Technology, history, and the idea of musical randomness," *Critical studies in media communication*, vol. 31, no. 3, pp. 244–264, 2014.

[4] K. R. M. Sanfilippo, N. Spiro, M. Molina-Solana, and A. Lamont, "Do the shuffle: Exploring reasons for music listening through shuffled play," *PLOS ONE*, vol. 15, no. 2, pp. 1–21, 02 2020. [Online]. Available: https://doi.org/10.1371/journal.pone.0228457

[5] M. Bar-Hillel and W. A. Wagenaar, "The perception of randomness," *Advances in applied mathematics*, vol. 12, no. 4, pp. 428–454, 1991.

[6] R. Croson and J. Sundali, "The gambler's fallacy and the hot hand: Empirical data from casinos," *The Journal of Risk and Uncertainty*, 2005.

[7] G. Smith, *Standard deviations: Flawed assumptions, tortured data, and other ways to lie with statistics*. Abrams, 2014.

[8] O. Celma, *Music recommendation and discovery*. Springer, 2010.

[9] R. A. Fisher and F. Yates, *Statistical tables: For biological, agricultural and medical research*. Oliver and Boyd, 1938.

[10] R. Durstenfeld, "Algorithm 235: random permutation," *Communications of the ACM*, vol. 7, no. 7, p. 420, 1964.

[11] D. Knuth, "Seminumerical algorithms," *The art of computer programming*, vol. 2, 1981.

[12] G. P, G. Gupta, S. H, S. Naseera, A. A. Siddiqui, G. G, G. B. Amali, and S. Gonjari, "Music playlist manager using fisher-yates shuffling algorithm and sorting," *World Wide Journal of Multidisciplinary Research and Development*, 2017.

[13] M. Fiedler, "The art of shuffling music," http://keyj.emphy.de/balanced-shuffle/, 2007, online; accessed 11 March 2021.

[14] P. Singh, A. Batheja, and A. Chowdhury, "Predictive music shuffling algorithm," *International Journal of Computer Science and Information Technologies*, vol. 6, 2015.

[15] R.-H. Liang and Z.-S. Liu, "Towards responsive music shuffling," https://www.researchgate.net/publication/228789429_Towards_Responsive_Music_Shuffling, accessed: 2020-10-14.

[16] L. Poláček, "How to shuffle songs?" https://engineering.atspotify.com/2014/02/28/how-to-shuffle-songs/, 2014, online; accessed 11 March 2021.

[17] M. Castelluccio, "The music genome project," *Strategic Finance*, pp. 57–59, 2006.