AUTOMATIC PLAYLIST SEQUENCING AND TRANSITIONS

Rachel M. Bittner, Minwei Gu, Gandalf Hernandez, Eric J. Humphrey, Tristan Jehan, P. Hunter McCurry, Nicola Montecchio
Spotify Inc., USA

ABSTRACT

Professional music curators and DJs artfully arrange and mix recordings together to create engaging, seamless, and cohesive listening experiences, a craft enjoyed by audiences around the world. The average listener, however, lacks both the time and the skill necessary to create comparable experiences, despite access to same source material. As a result, user-generated listening sessions often lack the sophistication popularized by modern artists, *e.g.* tracks are played in their entirety with little or no thought given to their ordering. To these ends, this paper presents methods for automatically sequencing existing playlists and adding DJ-style crossfade transitions between tracks: the former is modeled as a graph traversal problem, and the latter as an optimization problem. Our approach is motivated by an analysis of listener data on a large music catalog, and subjectively evaluated by professional curators.

1. INTRODUCTION

DJs are modern artists that carefully select, sort, and combine recordings in order to enhance the music listening experience over simpler forms, such as albums or playlists. They traditionally create *mixes* or *sets* that flow seamlessly from one song to the next by sequencing styles, matching keys and tempos, and smoothly transitioning between musical ideas. Importantly, the ordering of tracks or samples and the quality of the transitions between them are fundamentally linked: it can be very difficult to create an enjoyable transition between songs that significantly differs in style, tempo, or key. Transitioning between a slow, smooth jazz piece and a high energy, fast electronic track, for example, will likely feel awkward or unnatural and create an abrupt change in the listening experience.

Though listening to DJ mixes is not a new phenomenon, modern music streaming services indicate that there is significant appetite among users for curating their own sets, having produced over 2 billion playlists in the last decade on Spotify alone. [1] To develop a vague sense of how many users might be aspiring "DJs" in the home or car, we find that roughly 1% of the public playlists available through Spotify's Web API contain "party" in the title. [2] Even through coarse extrapolation, this suggests that some 20M playlists are candidates for DJ-style production.

Therefore, given that so many users are actively exercising their curatorial skills, the steady advance of machine listening technology offers promise that the more technical challenges of creating a DJ mix could be achieved computationally. In this paper we focus specifically on the two hurdles faced in creating a DJ set from a given playlist: compute an optimal sequencing, and create song-to-song transitions between sequenced tracks. One of the challenges of building a model for these two tasks is defining how to evaluate performance. Because quality of a song sequence and of a song-to-song transition is highly subjective, we rely on user studies to evaluate the performance of our systems.

2. RELATED WORK

Several commercial products (e.g., Serato DJ [3] and Native Instruments' Traktor line [4]) are designed to assist DJs with digital mixing on a laptop. These are mainly tools for enthusiasts and professionals who already have experience in mixing, and as such these tools tend to replicate with software their original analog counterparts. Automatic audio analysis techniques are sometimes exploited to let the user sort playlists by tempo and key, however by design it is up to the DJ to make a final selection and decide on where to transition: the software's role is to assist with time-stretching and facilitating the execution of beat-aligned transitions. This paper is concerned with the automation of the entire experience, demanding less involvement by the users; examples of commercial software in this category include Algoriddim DJay [5] , Pacemaker [6] , and Serato Pyro [7] .

Sequential ordering is the primary concern of [6], that uses an audio similarity metric built on Gaussian models of MFCCs. However, the approach does not constrain the problem to a pre-selected set of songs and instead generates playlists from a large pool. In analyzing the order-

[1] https://press.spotify.com/us/about/

[2] `https://developer.spotify.com/web-api/playlist-endpoints/`
[3] `https://serato.com/dj`
[4] `https://www.native-instruments.com/en/products/traktor`
[5] `https://www.algoriddim.com/`
[6] `https://pacemaker.net/`
[7] `https://seratopyro.com/`

ing of songs in professionally-made DJ sets, [11] presents evidence that timbral factors play an important role in sequencing. In [3], consideration is given to "tempo trajectories" over time as a way of modeling human DJs' ability to structure the rise and fall of energy levels in the music as the sequence of songs progresses. Ishizaki, et al. [9] focus on making smooth tempo adjustments to songs with the goal of minimizing abrupt changes that could cause listener discomfort. In choosing optimal mixing regions between two songs, [8] employs section similarity metrics derived from chroma information, along with beat and tempo features. Similarly, [7] proposes a model for measuring the perceptual consonance for different transition regions given two tracks. In [15], a more complete DJ simulation method is proposed, which performs song selection, ordering and cross-fading for electronic music. A closely related problem is the automatic creation of musical "mash-ups", for which a number of algorithms have recently been proposed [5, 12].

3. SEQUENCING

Given a playlist, the goal of a sequencing algorithm is to order the tracks it contains in a way as to make the music "flow smoothly" from each song to the next. Cunningham et al. performed an in-depth study of how individual users sequence tracks, and concluded that the task is "more of an art than a science" [4]. Thus, the notion of flow and its attainment is ultimately an aesthetic phenomenon; a DJ may want the tempo to stay relatively constant or neighboring songs to be acoustically similar as a function of creative intent, as illustrated in Figure 1. If songs are to be cross-faded, proper sequencing can ensure that consecutive pairs of songs have similar keys and tempos, allowing for less abrupt transitions. Understandably, the scope of this work entails a more calculated approach than that of an expert DJ, and we identify artist-quality sequencing as a broader aim of this research area. It is important to note that this problem is related to, but different from the task of *generating* playlists, for example as in [2] – in this task we are given a list of tracks and the task is to reorder them, rather than to find a list of coherent tracks from a large corpus.

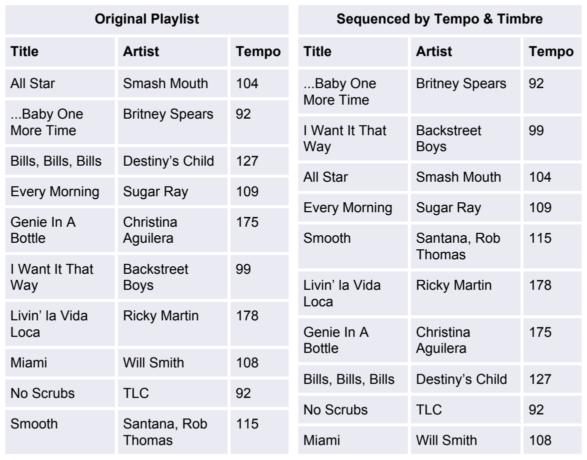
Examples of playlists sequenced using the proposed approach can be found online. [8] [9]

3.1 Method

The problem of sequencing tracks lends itself well to be formulated in a graph theory setting. The central step consists in mapping acoustic features into a Euclidean space so that songs that are fit to be sequenced next to each other are also close together in the feature space. Finding a good sequencing involves finding the shortest non-repeating path between all the songs.

[8] https://open.spotify.com/user/rabitt3/playlist/6a4lxKlqWZwKQgV3VhRMjX

[9] https://open.spotify.com/user/rabitt3/playlist/0Cl1BNwnWxmLkfUn8YQZVS



Original Playlist			Sequenced by Tempo & Timbre		
Title	Artist	Tempo	Title	Artist	Tempo
All Star	Smash Mouth	104	...Baby One More Time	Britney Spears	92
...Baby One More Time	Britney Spears	92	I Want It That Way	Backstreet Boys	99
Bills, Bills, Bills	Destiny's Child	127	All Star	Smash Mouth	104
Every Morning	Sugar Ray	109	Every Morning	Sugar Ray	109
Genie In A Bottle	Christina Aguilera	175	Smooth	Santana, Rob Thomas	115
I Want It That Way	Backstreet Boys	99	Livin' la Vida Loca	Ricky Martin	178
Livin' la Vida Loca	Ricky Martin	178	Genie In A Bottle	Christina Aguilera	175
Miami	Will Smith	108	Bills, Bills, Bills	Destiny's Child	127
No Scrubs	TLC	92	No Scrubs	TLC	92
Smooth	Santana, Rob Thomas	115	Miami	Will Smith	108

Figure 1: Example playlist sequencing by tempo and timbre.

3.1.1 *Constructing the Feature Space*

Several acoustic aspects of a song are exposed so that they can be combined differently:

- *acoustic vectors* are created by first using a convolutional neural network [16] trained to reproduce collaborative-filtering vectors in ($\mathbb{R}^{2048}$). The acoustic vectors are low dimensional embeddings ($\mathbb{R}^{2048} \Rightarrow \mathbb{R}^{8}$) of the output of the convolutional neural network, where the embedding was trained to minimize the Euclidean distance between artists. These features mostly capture the timbral character of a song.
- *key and mode* information from the Echonest analyzer is mapped into points in $\mathbb{R}^3$ so that adjacent keys in the circle of fifths and relative major/minor keys are equidistant, as pictured in Figure 2: Left.
- *tempo* (originally in beats per minute estimated from the Echonest analyzer) is represented in a base-2 logarithmic scale. In certain applications it is desirable to preserve tempo-octave invariance: in that case tempo is represented as a unit vector whose polar angle is mapped into a tempo octave, as in Figure 2: Right.

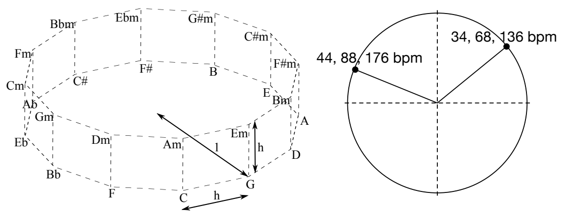


Figure 2: Left: key/mode mapping to Euclidean space. Right: octave-invariant tempo mapping to Euclidean space.

A feature vector is finally constructed by concatenating the above individual feature vectors, each feature is optionally weighted according to the application (e.g., in dance playlist, tempo coherence is important, thus tempo would have a large weight).

3.1.2 Solution as a Graph Problem

Let us consider the complete symmetric graph in which each song is associated to a vertex, and edges are weighted by the Euclidean distance between the corresponding songs' features.

A *Hamiltonian Path* is a path that visits each vertex in a graph exactly once. The optimal sequencing of a playlist corresponds to the *Shortest* Hamiltonian Path in the (complete) graph, a problem which is unfortunately NP-Complete (the total cost of an ordering is the sum of all the weights of the edges in the path). Several approximation strategies, shown below, have been considered; their computational cost is dominated by the construction of the weight matrix, quadratic in the length of the playlist.

A straightforward greedy approximation (which we denote by *HAM-1*) consists of iteratively selecting the closest non-visited vertex, starting from a given *seed* vertex. An improvement (*HAM-2*) can be made by selecting the closest non-visited vertex from either the tail or the head of the partial sequencing.

Empirically, both methods give satisfying results; the total cost of a *HAM-2* sequencing is virtually always lower (better) than its *HAM-1* counterpart, although the seed track does not end up as the head of the sequencing anymore (which could be itself a desirable feature). An undesirable artifact is the presence of poor track pairings at the tail of the sequencing for *HAM-1* and at both ends for *HAM-2*, due to the greedy nature of the algorithm.

A different solution is given by the Shortest Hamiltonian *Cycle*, an NP-complete problem (also known as the *Traveling Salesman Problem*) which however admits a polynomial 2-approximation [13]. The cost is usually higher than either of the greedy Hamiltonian Path solutions, but the resulting playlist will have smooth transitions, even when repeated in a loop, and is free from the head and tail artifacts described above.

3.2 Evaluation

To measure the effectiveness of the sequencing algorithm, we ran a pilot study in which professional curators blindly compared six sequenced vs. randomly sequenced playlists. Each of the six playlists contained 30 "Discover Weekly" playlists. The sequenced version of the playlist was created using *HAM-2* with acoustic vectors as features. For each of the six playlist pairs, the curators were instructed to (1) choose which playlist was sequenced better, and (2) list the pairs of tracks in each playlist that were deemed "abrupt" when played sequentially.

In the first task, for playlists 1, 2, and 5 the curators unanimously chose the sequenced playlist over the random playlist. For playlists 3 and 4, the curators were evenly split showing no preference, and for playlist 6, half preferred the sequenced, and half had either no preference or preferred the random playlist. Table 1 shows the average number of "abrupt" pairs of tracks across curators for each playlist. As expected there were more abrupt pairs in the random versions than in the sequenced versions. This is particularly drastic for playlist 5, probably due to the wider range of genres.

Playlist	Genres	Random	Sequenced
1	*Folk Pop, Country*	2.8 (1.8)	1.2 (1.3)
2	*Underground Hip-Hop, Funk*	3.8 (4.3)	1.2 (1.3)
3	*Abstract Hip-Hop, Indietronica*	2.7 (2.0)	3.3 (2.2)
4	*Indietronica, Indie Rock*	2.8 (1.9)	2.8 (3.7)
5	*Jazz, Classical, House*	9.3 (1.5)	2.7 (1.2)
6	*Folk Metal, Death Metal*	4.00 (3.6)	3.50 (2.3)
	Average	**4.2 (2.5)**	**2.4 (1.0)**

Table 1: Average number of song pairs (out of a total of 29 pairs) marked as "abrupt" across curators. The standard deviation is indicated in parentheses.

4. TRANSITIONS

Various streaming services provide, as a toggleable feature, a simple fixed-length crossfading between tracks; this however does not take content into account. About 95% of the users of Spotify forgo the option, and use standard end-to-end playback. To motivate the inclusion of transitions in a playlist, an A-B test was run on 10% of users of Spotify, where DJ-curated transitions were added to several popular playlists for the test group. The results showed that the percentage of people who returned to the playlists per day was 1.4 percentage points higher for the test group than control, suggesting that the listeners enjoyed the playlists with DJ curated transitions more and were thus more likely to listen again.

The goal of the algorithm we present is to create interesting DJ-like transitions between pairs of songs, which could be offered as an enhanced alternative to the existing crossfade. This involves choosing where in each track the transition will occur given a fixed transition length (in units of number of beats).

4.1 Method

Given a pair of tracks and a target transition length, our method selects transition start and end points in both songs, and uses this information to render the transition. Transition locations are restricted to downbeats, and are heavily weighted to occur on section boundaries, such as at the intersaection of a verse and a chorus. Additionally, we assume that regions of tracks that have similar timbre and pitch distributions will yield the smoothest transition. In this work, we only consider music in quadruple meter.

A symmetric crossfade, depicted in Figure 3, is arguably the most basic kind of transition: $t_1^{(A)}$ and $t_1^{(B)}$ denote the fade out start and end points in track 1, and $t_2^{(A)}$ and $t_2^{(B)}$ denote the fade in start and end points in track 2; the duration of the *transition region*, the interval $[t_i^{(A)}, t_i^{(B)}]$, for track 1 and 2 need not be equal.

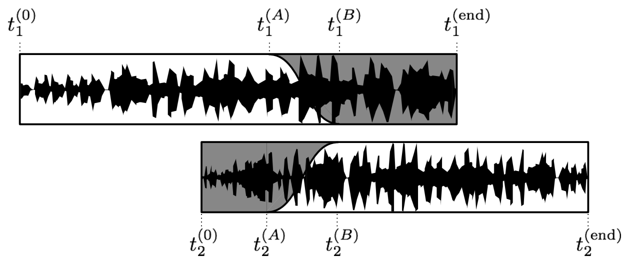


Figure 3: Sample crossfade transition. $t_1^{(A)}$ and $t_1^{(B)}$ mark the start and end points of the fade out for track 1. Similarly, $t_2^{(A)}$ and $t_2^{(B)}$ mark the start and end points of the fade in for track 2.

4.1.1 *Features*

Unless otherwise stated, each of the following features are computed for each track using the Echo Nest Analyzer, which is largely based on [10]. Let $\mathbf{b}$ be a list of estimated beat positions in seconds. Given the beat positions of each track, we compute several different types of event locations, each on the same time grid as the estimated beats. Let M be the set of indices of $\mathbf{b}$ which are downbeats. Similarly, let S be the set of indices of $\mathbf{b}$ which are section boundaries, and D be indices which are "drop" points. Section boundaries are computed using the method described in [14], and the "drop" point estimation is described in Section 4.1.2.

When choosing transition points, not all beats are created equal: the best transition points occur at strong structural boundaries. Each type of event location has a different level of structural significance in the track. The strongest structural boundaries, if they exist, are at the drop points. The next strongest points are section boundaries, followed by downbeats. Ideally, all drop points are section boundaries, and all section boundaries are downbeats, but this may not be the case.

In addition to these event locations, we compute several beat-synchronous features. Let N be the number of beats. Timbre features $\mathcal{T}$ are a (12 x N) matrix describing the spectral shape of each beat, and the chroma features C are a (12 x N) matrix giving the pitch class distribution for each beat. Loudness features ℓ and "vocalness" features v give the loudness and probability of vocals for each beat, and are each size (1 x N). Intuitively, transition regions with low loudness can often sound awkward and abrupt, and when vocals are present we risk overlapping vocals with the other track, or cutting over mid-sentence.

4.1.2 *Drop Point Estimation*

The goal in drop point estimation is to find the points in a track where the "drop" happens. The term "drop" is typically used in the context of specific types of electronic dance music, and refers to the point(s) in time where a drastic change in the song occurs following a large build. In our context, we are looking for points in a song where an exceptionally interesting event occurs. Rather than take a content-based solution similar to [17], we use a crowd-sourced approach following from the work described in P. Lamere's blog [10]. Lamere computes the points where users moved (scrubbed) the playhead while listening to a track. Typically users tend to move the playhead towards the most interesting points in the track. Figure 4 (top) shows an example of the aggregated playhead scrubbing data (blue) for *Skrillex: "First of the Year"*. The large peak occurring around 66 seconds accurately marks the first big drop, and the second smaller peak around 145 seconds marks the second big drop.

To identify these peak locations, we use a standard peak picking approach from the onset detection literature [1]: an adaptive threshold (shown in green) is computing using a median filter, then a detection function subtracts the adaptive threshold from the normalized scrub ratio and selects its peaks, as shown in Figure 4 (bottom). Choosing the closest downbeat that occurs before each resulting peak gives us our final drop index D. Note that in Figure 4 there is a small peak near the start of the track which is not a significant musical point. We correct for this by removing peaks that occur within the first 15 seconds of the track.

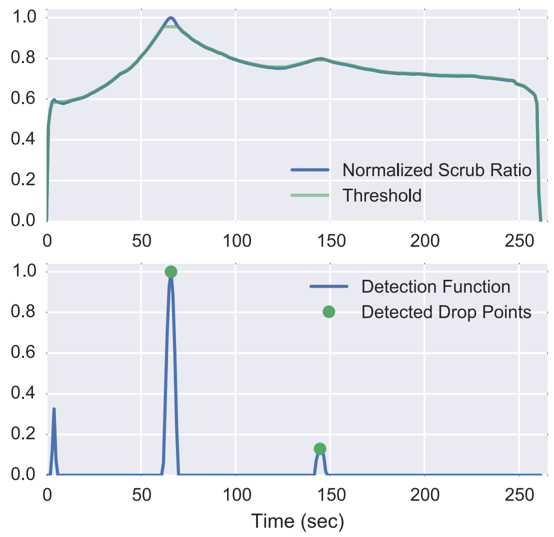


Figure 4: Drop point estimation intermediate steps for *Skrillex: "First of the Year (Equinox)"*. **Top**: Normalized scrub ratio and adaptive threshold. **Bottom**: Detection function and detected drop points. The first peak in the detection function is not a drop point because it occurs within the first 15 seconds of the track.

[10] `http://musicmachinery.com/2015/06/16/the-drop-machine/`

4.1.3 Selecting Transition Points

The procedure for selecting transition points between track 1 (T_1) and track 2 (T_2) of length n beats is outlined in Algorithm 1. The functions `beats` and `features` are described in Section 4.1.1.

Let $\mathbf{t}_1$ and $\mathbf{t}_2$ be the set of transition point candidates from which $t_1^{(A)}$ and $t_2^{(A)}$ will be selected. Since we are given a transition duration (in units of number of beats), $t_1^{(B)}$ and $t_2^{(B)}$ can be determined from the values of $t_1^{(A)}$ and $t_2^{(A)}$. Initially, we set $\mathbf{t}_1 = M_1$ and $\mathbf{t}_2 = M_2$.

We prune these sets to ensure that the transition points happen in reasonable portions of the track, removing obvious "bad" regions. The pruning is performed using the following rules:

- $t_1^{(B)}$ occurs before the fade out, $t_2^{(A)}$ is after the fade in
- $t_1^{(B)}$ occurs within the last 25% of the track, $t_2^{(A)}$ occurs within the first 20% of the track.

After pruning, the remaining points in $\mathbf{t}_1$ and $\mathbf{t}_2$ are considered valid candidates. These pruned sets are the output of the `candidates` function.

For each pair of points in $\mathbf{t}_1$ and $\mathbf{t}_2$, we compute pairwise comparisons along a series of different features over the entire overlapping region. For a transition of length n beats, the overlapping region begins at beats i and j, and ends at beats $i + n$ and $j + n$. In Algorithm 1 beginning at line 9, we use the notation $\mathcal{T}_1[i : i_n]$ to denote features within the region beginning at beat i and ending at beat i_n. Let Λ be the combined transition point cost matrix, where one axis represents the beat indices of track 1 and the second of track 2. Let Λ_x be the transition cost matrix for a particular feature comparison x. For timbre and chroma features, we compute $\Lambda_\mathcal{T}$ and Λ_C as the Euclidean distance between the features directly (Algorithm 1 lines 9, 10). Λ_ℓ (line 11) is computed as the sum of the average inverse loudness for each track, giving regions that are loud in both tracks a low transition cost. Similarly, Λ_v is the sum of the average "vocalness" probability, to assign transitions that both have vocals a high cost. Finally, we penalize transitions that do not end on a drop or a second boundary in both tracks (lines 13, 14), with a score of 2 if neither track's region ends on a boundary, and a score of 1 if only one track's region ends on a boundary.

Each feature's individual cost matrix Λ_x is standardized so that the minimum cost is 0 and the maximum cost is 1. The final cost matrix Λ is computed as a weighted sum of each feature's cost matrix after standardization. An example of each of feature's standardized matrix is shown in Figure 5, and the weighed combination is shown in Figure 6. The final transition points $t_1^{(A)}$ and $t_2^{(B)}$ are chosen as the times corresponding to the minimum cost entry in Λ.

Algorithm 1 Picking Transition Points

```
1:  procedure TRANSITION-POINTS($T_1, T_2, n$)
2:      $\mathbf{b}_1 \leftarrow \texttt{beats}(T_1), \mathbf{b}_2 \leftarrow \texttt{beats}(T_2)$
3:      $\mathcal{T}_1, C_1, \ell_1, v_1, M_1, D_1, S_1 \leftarrow \texttt{features}(T_1, \mathbf{b}_1)$
4:      $\mathcal{T}_2, C_2, \ell_2, v_2, M_2, D_2, S_2 \leftarrow \texttt{features}(T_2, \mathbf{b}_2)$
5:      $\mathbf{t}_1 \leftarrow \texttt{candidates}(T_1, M_1, S_1, D_1, \ell_1)$
6:      $\mathbf{t}_2 \leftarrow \texttt{candidates}(T_2, M_2, S_2, D_2, \ell_2)$
7:      for $i \in \mathbf{t}_1, j \in \mathbf{t}_2$ do
8:          $i_n \leftarrow i + n \; j_n \leftarrow j + n$
9:          $\Lambda_\mathcal{T}[i, j] \leftarrow \texttt{norm}(\mathcal{T}_1[i : i_n] - S_2[j : j_n])$
10:         $\Lambda_C[i, j] \leftarrow \texttt{norm}(C_1[i : i_n] - C_2[j : j_n])$
11:         $\Lambda_\ell[i, j] \leftarrow \texttt{avg}\,(2 - (\ell_1[i : i_n] + \ell_2[j : j_n]))$
12:         $\Lambda_v[i, j] \leftarrow \texttt{avg}(v_1[i : i_n]) + \texttt{avg}(v_2[j : j_n])$
13:         $\Lambda_D[i, j] \leftarrow 1_{i_n \notin D_1} + 1_{j_n \notin D_2}$
14:         $\Lambda_S[i, j] \leftarrow 1_{i_n \notin S_1} + 1_{j_n \notin S_2}$
15:     end for
16:     $\Lambda \leftarrow [\Lambda_\mathcal{T}, \Lambda_C, \Lambda_\ell, \Lambda_v, \Lambda_D, \Lambda_S]$
17:     for $k \in \Lambda$ do
18:         $k \leftarrow \texttt{standardize(k)}$
19:     end for
20:     $\Lambda \leftarrow \texttt{weightedAvg}(\Lambda_\mathcal{T}, \Lambda_C, \Lambda_\ell, \Lambda_v, \Lambda_D, \Lambda_S)$
21:     $i^*, j^* \leftarrow \texttt{argmin}(\Lambda)$
22:     $t_1^{(A)}, t_2^{(A)} \leftarrow \mathbf{b}_1[i^*], \mathbf{b}_2[j^*]$
23:     return $t_1^{(A)}, t_2^{(A)}$
24: end procedure
```

4.2 Rendering Transitions

Transitions are rendered such that the beats in the two tracks occur at the same time. In virtually every case, the tempos are not perfectly in sync, each beat is timestretched such that the tempo slowly changes from the tempo of track 1 to the tempo of track 2. For an N beat transition, if the nth beat in track 1 has duration d_1 and the beat in track 2 has duration d_2, the total duration of the new nth beat is $d_{\text{out}} = \frac{N-n}{N} d_1 + \frac{n}{N} d_2$. To achieve this, the nth beat in track 1 is time stretched by a factor of d_1/d_{out}, and the n^{th} beat in track 2 by d_2/d_{out}.

4.3 Evaluation

A selection of rendered transitions were evaluated by subjective human review. We randomly picked 48 pairs of tracks from a selection of popular music across multiple dance genres, using tempo constraints when picking the tracks to make sure the tempo difference between pairs was no more than 5 bpm.

For each of the pairs, we asked four professional curators to listen to the transition all the way through at least once and rate the quality. For subjective measurement, the overall quality is described as Good (3), OK (2) and Bad (1). Additionally, curators were asked to describe any potential problems they noticed within the transitions, such as "beats do not align" or "key clash". The results are shown in Tables 2 and 3, respectively.

A fairly large number (15%) of transitions were marked as "Bad" because the "beats do not align". Since we constrain transitions to align along estimated beats, we conclude that the "beats do not align" transitions occur as a result of errors in the beat estimation algorithm. Transitions labeled as "transitioning mid-vocals" are also likely a result of errors in our vocal activity detection algorithm. In

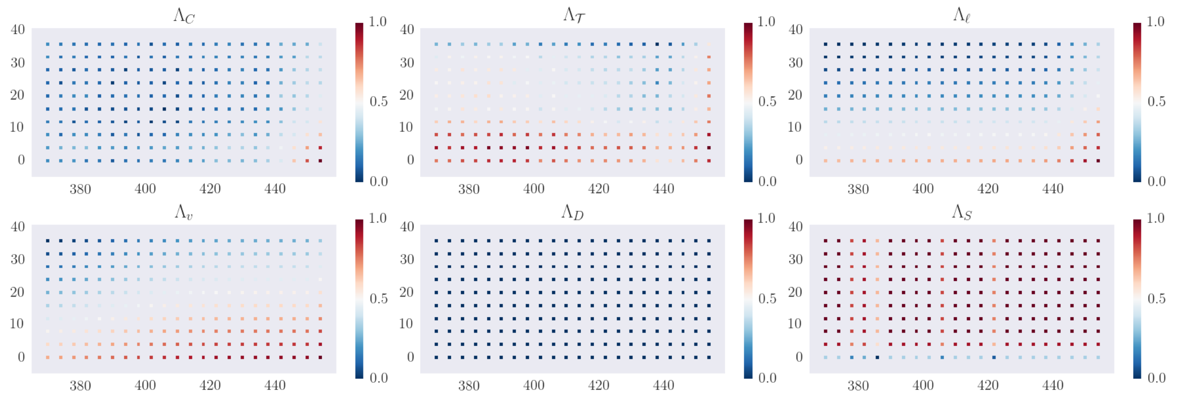


Figure 5: Transition matrices for each feature for a pair of songs. The x-axis show beat indices in Track 1, and the y-axis for Track 2. Many index pairs have no score because they are not part of the set of candidates. Dark blue points indicate good transition pairs for the feautre, while red indicates a poor pair. In this example, no drops were detected, so Λ_D is a uniform matrix.

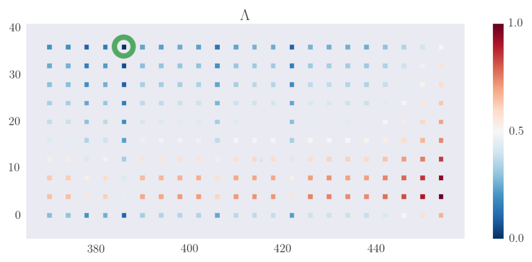


Figure 6: Weighted combination Λ of the individual feature matrices in Figure 5.The x-axis show beat indices in Track 1, and the y-axis for Track 2. The point with the lowest cost is circled in green.

Rating	Percentage
3 - Good	64.13%
2 - OK	28.26%
1 - Bad	7.61%
Average (Std) Rating	2.56 (0.38)

Table 2: Average percentage of quality rating for all track pairs and average rating of song pairs in rendered transition test set. The standard deviation is indicated in parentheses.

Reason	Percentage
Beats do not align	15.22%
Not on downbeat	2.17%
Key clash	0%
Awkward transition points	2.17%
Transitions mid-vocals	6.52%
Contrasting Songs	4.34%

Table 3: Average percentage of song pairs marked as the stated reason for bad quality transitions by curators.

both of these cases, as beat tracking and vocal activity detection algorithms improve, these transition quality issues should be mitigated. An interesting finding is that “key clash” is not marked as problematic by any of the curators for a single transition in either transition types.

5. CONCLUSIONS

This paper has presented systems for automatically sequencing and generating DJ-style transitions for a playlist of songs. Both systems were evaluated with the help of professional curators. Beat and downbeat tracking errors were found to be the primary bottleneck in the subjective performance of automated transitions.

A possible alternative approach for tackling the sequencing and transitioning problems entails the usage of Machine Learning approaches. Given a large number of (carefully curated) playlists and transition points between them, one might attempt to directly learn the mapping of low-level audio representation of recordings into their optimal sequencing and transitions. Such an methodology is certainly fascinating, and represents in fact a future research direction. However, the experiments above prove how just a few *interpretable* features are suitable for this problem to a remarkable extent. We chose then to investigate an approach that is heuristic in nature, but whose particular behavior can be customized by the user in an extremely intuitive manner (e.g., weighting acoustic similarity more than key and tempo might be preferred when constructing a playlist for a radio show, while the reverse is true in the case of a dancing playlist).

Finally, this work has focused on specific genres of music – namely “party” music. The constraints we imposed may not be necessary or sufficient for other genres of music, for example rap. However, the same framework could be applied substituting different features in the optimization problem. The exploration of how to apply this model to other genres is left as future work.

6. REFERENCES

[1] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *Speech and Audio Processing, IEEE Transactions on*, 13(5):1035–1047, 2005.

[2] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 714–722. ACM, 2012.

[3] Dave Cliff. Hang the DJ: Automatic sequencing and seamless mixing of dance-music tracks. *HP LABORATORIES TECHNICAL REPORT HPL*, 104, 2000.

[4] Sally Jo Cunningham, David Bainbridge, and Annette Falconer. 'more of an art than a science': Supporting the creation of playlists and mixes. In *ISMIR*, pages 240–245, 2006.

[5] Matthew EP Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto. Automashupper: Automatic creation of multi-song music mashups. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(12):1726–1737, 2014.

[6] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. Playlist generation using start and end songs. In *ISMIR*, pages 173–178, 2008.

[7] Roman B Gebhardt, Matthew EP Davies, and Bernhard U Seeber. Psychoacoustic approaches for harmonic music mixing. *Applied Sciences*, 6(5):123, 2016.

[8] Tatsunori Hirai, Hironori Doi, and Shigeo Morishima. Musicmixer: Computer-aided dj system based on an automatic song mixing.

[9] Hiromi Ishizaki, Keiichiro Hoashi, and Yasuhiro Takishima. Full-automatic dj mixing system with optimal tempo adjustment based on measurement function of user discomfort. In *ISMIR*, pages 135–140, 2009.

[10] Tristan Jehan. *Creating music by listening*. PhD thesis, Massachusetts Institute of Technology, 2005.

[11] Thor Kell and George Tzanetakis. Empirical analysis of track selection and ordering in electronic dance music using audio feature extraction. In *ISMIR*, pages 505–510, 2013.

[12] Chuan-Lung Lee, Yin-Tzu Lin, Zun-Ren Yao, Feng-Yi Lee, and Ja-Ling Wu. Automatic mashup creation by considering both vertical and horizontal mashabilities. In *ISMIR*, pages 399–405, 2015.

[13] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[14] B. McFee and D. P. W. Ellis. Analyzing song structure with spectral clustering. In *ISMIR*, 2014.

[15] Jaume Parera. Dj codo nudo: a novel method for seamless transition between songs for electronic music. Master's thesis, Universitat Pompeu Fabra, Barcelona, 2016.

[16] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651, 2013.

[17] Karthik Yadati, Martha Larson, Cynthia CS Liem, and Alan Hanjalic. Detecting drops in electronic dance music: Content based approaches to a socially significant music event. In *ISMIR*, pages 143–148, 2014.