A User-Adaptive Automated DJ Web App with Object-Based Audio and Crowd-Sourced Decision Trees

Florian Thalmann
Centre for Digital Music, Queen Mary University of London
f.thalmann@qmul.ac.uk

Lucas Thompson
Centre for Digital Music, Queen Mary University of London
lucas.thompson@qmul.ac.uk

Mark Sandler
Centre for Digital Music, Queen Mary University of London
mark.sandler@qmul.ac.uk

ABSTRACT

We describe the concepts behind a web-based minimal-UI DJ system that adapts to the user's preference via simple interactive decisions and feedback on taste. Starting from a preset decision tree modeled on common DJ practice, the system can gradually learn a more customised and user-specific tree. At the core of the system are structural representations of the musical content based on semantic audio technologies and inferred from features extracted from the audio directly in the browser. These representations are gradually combined into a representation of the mix which could then be saved and shared with other users. We show how different types of transitions can be modeled using simple musical constraints. Potential applications of the system include crowd-sourced data collection, both on temporally aligned playlisting and musical preference.

1. INTRODUCTION

The Web Audio API allows us to incorporate increasingly sophisticated audio functionality into web applications and directly use the client's local resources for audio processing. While decreasing the need for server-side processing, this also enables more personalised and immediate user experiences. Delivering such experiences directly in the browser instead of as desktop applications has several advantages. They are immediately accessible by any user and from any device, due to the increased platform independence and the fact that no installation process is needed. Furthermore, the usage data necessary for such a customised and gradually improving experience can directly be gathered by the same application in real time, due to all user actions being carried out online. Finally, changes to the experience can directly be pushed to the user from the server, even while the application is being used. These kinds of applications have a significant potential for crowd-sourced data collection for audio and music research.

The current paper discusses the prototype of such an application, a web-based automated DJ software with the purpose of gaining some insight into music listening behavior. In particular, the goal is to investigate what kinds of transitions between any given arbitrary pair of successive musical pieces are generally favored by music listeners. The app is able to generate a number of standard transition types inspired by DJ practice and uses a customisable decision tree to determine the best type and configuration of transition for a user in a situation. All decisions are informed by high-level descriptors which are inferred from audio features directly extracted in the browser using the piper framework.[1] The internal representation of the music, mix, and transitions, in turn, are modeled using dynamic music objects.[2] The app features a minimal user interface that lets users drag and drop songs from their local private collection onto the browser and rate the decisions taken by the app with a simple star rating. An API collects all the high-level descriptors, decisions, and ratings, which can then be used to learn a personalised or improved decision tree, as well as to study differences in musical preference. With its minimal user interface, the app is aimed at amateur music listeners rather than professional DJs and its automatic mixing functionality can be decoupled and embedded into other music-based and potentially less or more interactive websites.

After a brief section on related work and DJ practice, we describe how a basic decision process can be derived for a given set of high-level descriptors which can be inferred from audio features. We also explain what kinds of data we collect during a session in order to learn customised decision trees. Then, we describe how mixes and transitions can be modeled as multi-hierarchical structures with constraints on their parameters. Finally, we conclude the paper with some details about the implementation and a preliminary experiment we used to assess the transition types and a sample decision process.

2. RELATED WORK

Within the field of Music Information Retrieval (MIR), there has been extensive work on music recommendation and the closely related topic of automatic playlist generation. In [3], a review of the playlist generation literature, it is highlighted that many approaches favour similar sounding tracks, resulting in rather homogeneous mixes. Similarly, the majority of proposed automatic DJ systems focus on transitioning patterns well-suited for homogenous playlists [14]. Most systems focus on simply beatmatching songs and linear volume crossfading between them [8, 6], with very slow tempo ramps [11, 7], and some added EQ or filter-

Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution**: owner/author(s).

Web Audio Conference WAC-2018, September 19–21, 2018, Berlin, Germany.

© 2018 Copyright held by the owner/author(s).

[1] https://github.com/piper-audio
[2] https://github.com/dynamic-music/dymo-core

ing techniques [4, 14]. Many systems also attempt to find optimal transition points, mostly based on local rhythmic or harmonic compatibility [4, 8, 11, 6], whereas only few consider genre-specific higher-level segmentation or areas of vocal activity [14].

In actual DJ practice, however, more diverse mixes and a greater variety of transition types are common devices for adding musical interest or an element of surprise in order to keep the crowd engaged. Also, DJs naturally have their own style and signature techniques, such as quick song switching, fast tempo ramps, or juxtapositions of multiple songs, especially since the emergence of digital technology. Few projects have attempted to address these varieties. In [6], a binary rating-based personalisation option is provided to circumvent mix homogeneity, allowing the system to adapt to the users preference for particular mixing of tracks. However, the user ratings simply override the similarity ratings predefined by the system. In the commercial app space, Serato's Pyro[3] offers similar functionality, incorporating a number of transition types for mixing between songs with varying levels of compatibility. Due to allowing tracks from streaming services such as Spotify, transition points are often constrained to the start / end points of tracks or from the currently playing point to the start of the next cued track. Recent work from Spotify [2] frames playlist sequencing as a DJ mix, and uses aggregate user data on play-head scrubbing behaviour to find interesting transition points, incorporating beat detection and time-stretching of individual beats to render transitions with over a ramping tempo, gradually beat-matching the tracks.

For the reasons outlined above, in this paper, we focus on including a greater number of transition types, as well as song successions of varying compatibility, which, at the potential expense of stylistic consistency and robustness, enables increased user customisability. However, despite being central to the practice, in the context of this paper we omit the detection of optimal cue points as the system is designed to transition as soon as possible after a user drops a song, which is particularly useful in the experimental setup described later on.

3. DJING AS A DECISION PROCESS

Traditionally, a pair of turn-tables for playing vinyl records, and a mixer for blending the outputs of the two playing records, are the basic tools used by a DJ. The task of mixing one track into another usually occurs over some time window, and can utilise temporal alignment, effects, and processing techniques for modifying aspects of either track. We use the terms *current* and *cued* to refer to the track being transitioned from and to, respectively. More recently, software applications make the realtime manipulation of tracks easier, as well as the process of *cueing* tracks, and executing more complex.

For the purposes of creating an application for mixing tracks, it is necessary to model fundamental techniques used by DJs to blend a variety of tracks into a continuous stream. In the first instance, we focus only on a small set of basic transition techniques commonly used when performing, and present in commercial DJ software, which cover a wide range of mixing scenarios:

[3]https://seratopyro.com

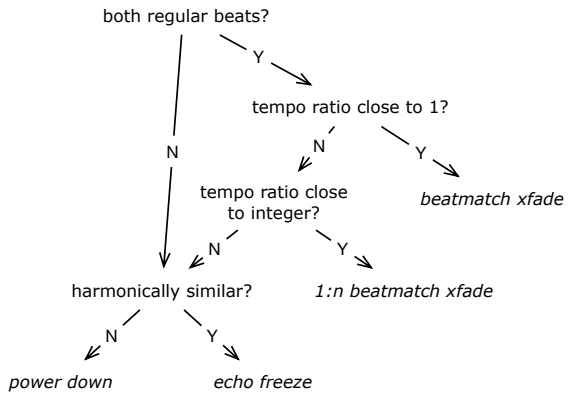

Figure 1: An example of a simple decision tree for four binary (yes/no) features and four transition types.

- slam / cut / switching - transition immediately, usually on the downbeat of the cued track
- cross-fade - smoothly fade-out the current track and fade-in the cued track
- beat-match - adjust the playback rate of the cued track for the beats to be synchronous
- echo-out / echo-freeze - apply an echo effect to the current track and stop it, creating a wash as the cued track comes in
- power-down - the record slows down drastically in a short time, resulting in a distinctive downward pitch slope

These can be framed as the primitives for blending tracks, or the basic units of composition for more complex transitions. As previously stated, these are typically used alongside some signal processing across time to achieve more sophisticated transitions. In the context of creating an automatic system, these can be combined with feature extraction for deriving harmonic and rhythmic information from the current and cued track, to create more musical transitions, such as echo-out-and-drop-on-the-one (first beat of next track, perhaps at a harmonically compatible section), and using metrical information and time-stretching or changing playback rate, beat-matched cross-fading can be achieved.

3.1 Modeling a Basic Decision Tree

Based on these observations we decided to formalise what DJs do as a simple decision tree, which the application uses to decide on how to perform a transition for a given pair of successive musical pieces. For the tree to be intuitively understandable to humans, we devised a set of high-level features that correspond to simple musical questions that DJs may ask when selecting a next song as well as choosing a way to transition to that song. Each parent node in the tree corresponds to such a question whereas the outgoing edges correspond to the available set of answers. Finally, the leaf nodes correspond to chosen transition types and potential configuration parameters. Figure 1 shows an simple example of such a tree.

	Descriptors	
Type	*Individual*	*Comparative*
Temporal	tempo t	tempo ratio tr
		tempo multiple tm
	regularity r	regularity ratio rr
Dynamic	loudness l	loudness ratio lr
	dynamicity d	dynamicity ratio dr
Harmonic	key k	key distance kd

Table 1: A sample set of descriptors inspired by DJ practice.

In order to ensure that the system is flexible and open ended, we decided to allow for both the set of available transitions with their parameters, and the set of features used for the decisions to be extensible and custom-defined. Similarly, the corresponding decision tree is replaceable and can involve decisions based on both qualitative and quantitative features, e.g. if the cued track has a tempo > 130, do this. The following sections describe the initial sets of features and transitions we chose for the application prototype.

3.2 Descriptors Inferred from Audio Features

To simplify the investigative study as well as the initial learning process, we devised a limited number of high-level features or descriptors. Each decision feature can be derived in a simple way from a set of standard audio features available in many MIR libraries. We calculate each feature for the *individual* songs in a succession, as well as *comparatively*. The members of the first class describe aspects of an individual track, whereas the ones of the latter two compare two tracks. Local comparative features describe a temporal compatibility of two tracks, e.g. the onsets of the harmonically most similar parts. Table 1 shows a compilation of the initial set of features chosen for the prototype. A similar set of features was for example chosen in [10], where listeners' preference for simple successions of songs was measured.

All temporal descriptors here can be derived from the output of a beat detector, which consists in the beat durations b_i for a given track. $t = 60/mean(b_i)$ (see table for names and variable names) and $r = var(b_i)$. tr, rr are simply derived from the respective values for t, r of the cued song divided by the ones for the current song, and $tm = tr \mod 1$. The dynamic descriptors are calculated analogously, with the input being a vector of loudness values l_j for the duration of a track, obtained from a corresponding feature extractor. $l = mean(l_j)$ and $d = var(l_j)$, whereas lr, dr are obtained as above. $k \in \{0, \ldots, 11\}$ can be obtained directly from a key feature extractor, or be calculated as the mode of a temporal sequence of key values. kd is the directed distance on the circle of fifths from the first to the second key.

3.3 Learning a Custom Transition Preference

The app prototype contains a basic decision tree as described above, which is used in the initial data collection phase. However, already with relatively little data for a given user or for the entirety of users, the system will be able to learn a customised or improved decision tree. In this section, we describe the nature of the data collected and how it can be used to learn different aspects of the system.

For every new song dropped onto the app and the subsequently generated transition, the user is able to give a numerical rating (currently 1-5 stars). These ratings, along with the descriptors and the decision taken, are forwarded to a specially devised API and stored in a database for further use. Each record thus has the following structure:

$$user\_id, rating, descriptors, process, transition, params$$

process is the decision process used in the particular case, which can currently be random or informed by the decision tree (see Section 6 below), *transition* is the chosen transition type (which may include custom ones), and *params* is an array values chosen for the parameters of the chosen transition, e.g. duration, offset, or effect level (see below).

Once enough training data has been collected for a user, the system queries the API to obtain the data records and learns a user-specific tree using the CART algorithm[4]. The system currently simply selects all well-rated records (4-5 stars) and devises the transition types as supervisory classes and the descriptors as training input objects.

In addition to the decision tree, in future work, the system will also be able to learn, via linear regression, optimal parameter values for each of the transition types, given a specific feature.

4. REPRESENTING THE MUSIC, MIXES AND TRANSITIONS

We use the Dynamic Music Objects format [12] for the definition and playback of all involved musical structures. It consists of a semantic web based format for the representation of constrained multi-hierarchical musical structures. Each object has a certain type, can be annotated with features and parameters, and can have a number of parts, which are again objects following the same definition. Every object can also have multiple parents, which lets us to define useful structures. The types define the way an object and its parts are navigated when played back, e.g. as a sequence, simultaneously, at random, permuted, and so on.

For the present purpose, the format allows us to represent temporally segmented audio files hierarchically and automatically annotate the segments with any available audio features. Via a simplified programming interface, these structures are converted into semantic web triples and stored in an in-browser triple store which can then be queried throughout the decision and playback process. For example, when calculating the higher-level descriptors described above we can simply query for feature values at a musical level of interest, such as bars or beats. These annotated values can also be used when setting up specific transitions which depend on local feature values, e.g. when defining a ramp between the tempos of the current and the cued songs, as explained below. The format allows us to easily model various transition techniques by defining constraints between parts of the musical structure, as simple logical expressions. At runtime, when triggered, these constraints are then automatically asserted by the *dymo-core* library.

4.1 Mixes as Multihierarchical Structures

Musical pieces are commonly viewed as a hierarchical or even multi-hierarchical structures with their sections, peri-

[4]https://github.com/mljs/decision-tree-cart

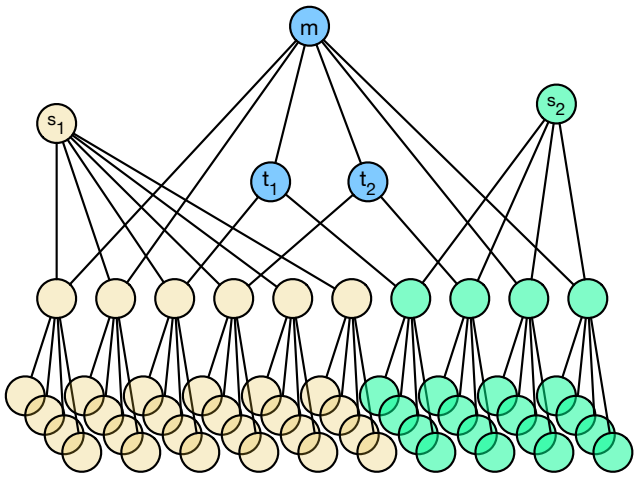

Figure 2: A simplified representation of a sample mix m from one piece s_1 to the next s_2, where the mix features bars from both pieces as well as combined transition parts t_1, t_2.

ods, phrases, slurs, bars, beats, and onsets or events being examples of hierarchical levels, which may be conflicting and thus multiply hierarchical. For our purpose it suffices to model a few of these levels, depending on the set of features chosen. For the features compiled in Table 1, for example, it suffices to have three levels, including the piece (root node), bar, beat levels. In terms of Dynamic Music Objects, each piece loaded into our player is represented as a sequence of bars, which are in turn sequences of beats. Such a structure can directly be passed to the player, which then simply plays the piece, seemingly in its original form, unnoticeably concatenated, beat by beat.

Given multiple such representations, we can build a new structure on top of them, which may reuse and recombine their elements in arbitrary ways. This allows us to build a DJ mix by defining a new root node of type sequence, representing the whole mix, to which we then gradually add the parts of the pieces we intend to play. These parts can again be of various types and may vary depending on the type of transition we decide to add. Figure 2 shows a simplified representation of a mix containing parts of two pieces. Depending on which root object we pass to the player, we can get it to either play the individual original songs, or the created mix, or all at the same time.

4.2 Transition Types and Constraints

Transitions can be modeled in a similar way, by concatenating and recombining parts of the current and cued song. In the current implementation, whenever a transition is decided on, the rest of the current song is removed and a transition part is added, followed by the rest of the cued song. This process is repeated for every new cued song. Most transitions have parameters, which define certain aspects of their nature, e.g. offset in the cued song, or duration of the transition.

Figure 2 also contains a simple transition part where the two pieces are playing at once. In this case, the transition is modelled using a few conjunction objects, which unite the two pieces at the bar level. Without any audio parameter automation, this transition sounds rather crude. We can introduce such automation by adding constraints to the structure, which will be triggered at given points.

Depending on its type, the transition part is modeled differently. The simplest case is the *slam*, which basically means no transition, where the current bar of the current song is directly followed by the first relevant bar of the cued song, depending on the offset. For the *echo freeze*, the echo parameter of the last beat or bar of the current song is set to 1, a silent part of varying duration is inserted, followed by the new song.

While these transitions simply consist in adding bars to the mix sequence, the remaining transitions make use of additional object types and constraints. For the *crossfade*, we can simply add a conjunction of two sequences of bars of comparable duration from each of the songs, complemented by a constraint which ties the value of a ramp to amplitudes of the bars as follows:

$$Amplitude(b_i^1) = r, \text{ and } Amplitude(b_j^2) = 1 - r$$

where $r \in [0, 1]$ is the value of the ramp and b_i^1 and b_j^2 are the bars of the current and cued song. The *beat-matched crossfade*, in turn, consists in a sequence of pairs of bars (b_i^1, b_i^2) with the additional three constraints for tempo interpolation:

$$t = (1 - r) * t_1 + r * t_2$$

and beat-maching:

$$TimestretchRatio(b) = t/60 * DurationFeature(b)$$
$$DurationRatio(b) = 1/TimestretchRatio(b)$$

where r is the ramp value, t_1, t_2 the tempos of the current and cued tracks, and b all beat objects involved in the transition. The *power down* can be generated with an analogous ramp on playback rate, but without any overlapping of the tracks' bars.

5. WEB APP IMPLEMENTATION

As a proof of concept, we implemented a simple browser-based prototype with the purpose of collecting some initial data in order to evaluate the starting point and initial decision tree, as well as to train the system. The prototype consists of a main Angular 6 web app[5] that communicates with a Node API which collects and serves all the data in a MongoDB instance. The automatic DJ functionality is implemented as a separate node module written in TypeScript, which can be embedded into any web application.[6]

The interface to the node module is very simple. It includes a class `AutoDj` with three public functions: `isReady()` which returns a Promise that resolves once the module is initialized, `getBeatObservable()` which returns an RxJS Observable which emits an incremented number whenever a new beat is being played (can for example be used for animations), and `transitionToSong(audioUri: string)` which transitions to the song at the uri passed as an argument. Anything else is done internally and automatically, including loading the audio, extracting features, calculating higher-level descriptors, deciding on a transition, gradually adding to the mix structure, and playing it back. The constructor

[5]https://dynamic-music.github.io/fast-dj
[6]https://www.npmjs.com/package/auto-dj

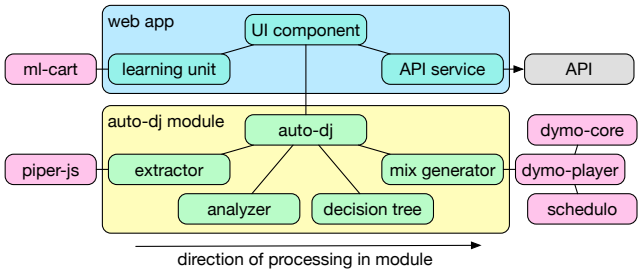

Figure 3: The current structure of the prototype.

of `AutoDj` currently takes a few optional arguments, including a custom decision tree (specified as a JSON structure), a decision mode such as `Tree`, `Random`, or `Default`, a default transition type, and a custom feature service which may provide pre-extracted features for speeding up the process.

The internal structure of the module, shown in Figure 3 consists of a *feature extractor* (the default feature service), which extracts features in the browser for each given track using *piper-js* (see below), an *analysis unit* which calculates and buffers the high-level descriptors, a *decision unit* which can contain various decision processes, and a *mix generator*, which contains templates for all the transition types, adds hierarchical song structures to the *dymo-core* triple store, and creates the mix object as described above, which is then navigated by the *dymo-player*,[7] a playback module optimized with Web Workers and based on our dynamic scheduling module *schedulo* built around Tone.js.[8] The main `AutoDj` class connects and communicates with all other units.

Everything else is done in the web app, including aggregating the user data, communicating with the API, and learning the decision trees. The UI of the web app is currently kept as minimal as possible, as can be seen in Figure 4. It guides the user through a cycle of successive drag-and-drop, analysis, transitioning, and rating stages. A new track is currently cued as soon as possible, while sticking to the bar structure, which accelerates the experimental data collection process. For other kinds of studies and public use, the interface will be made more flexible later on (see future work). Every new rating is sent to the API which forwards them to a database, which can be then be queried for data evaluation and the learning stage.

5.1 Piper and In-Browser Music Analysis

Recent advances in web platform standards such as SharedArrayBuffer,[9] Atomics,[10] and WebAssembly [5], allow for sophisticated signal processing techniques to be realistic in interactive web applications. The use of WebAssembly and related toolchains have recently been used to bring existing audio effects [9] and feature extraction methods [13] to browser applications. We leverage these technologies to bring in well-known methods for beat detection, chroma etc, but also to do additional audio processing of audio in a non-blocking fashion for functionality not provided by the Web Audio API.

[7]https://github.com/dynamic-music/dymo-player
[8]https://tonejs.github.io
[9]https://hacks.mozilla.org/2017/06/a-cartoon-intro-to-arraybuffers-and-sharedarraybuffers/
[10]https://hacks.mozilla.org/2017/06/avoiding-race-conditions-in-sharedarraybuffers-with-atomics/

Figure 4: The minimal user interface after a transition has been rated.

In connection with feature extraction, an interesting advantage of using decision trees is that, depending on their structure, they can significantly optimise the feature extraction process, since not for every path all features need to be extracted. For instance, for the tree in Figure 1, if the current track has already been determined to be irregular, the only additional feature needed for the decision is the key of both tracks, after which the transition can be started immediately and additional feature extraction can be scheduled while the new track is already playing.

6. A PRELIMINARY EXPERIMENT

To assess the current state of the system we conducted a brief first experiment where we let four participants use the app for 15 minutes each, with the same set of 200 highly varied anonymised songs.[11] The purpose of the experiment was to compare the outcomes of the decision process with randomly generated transitions. The app was set up to choose with a 50% likelihood between the suggestion by the preset decision tree, and a transition selected at random. Data about a total of 81 transitions were collected during the experiments, where transitions based on the tree were rated higher with a p-value of 0.080346 in an independent t-test. Figure 5 shows a plot of the distributions for both randomly and decision-tree generated transitions. The high variation in both the ratings of the informed and random transitions may result from the fact that a random transition can have a surprising or entertaining effect, whereas informed transitions may not work for a given pair of songs, due to the variation in reliability of the high-level descriptors. The users were satisfied with the simplicity of the app and enjoyed hearing what it decided to do, and some stated that they would like to try out the app with their own music collections.

7. CONCLUSION AND FUTURE WORK

Although the app introduced in this paper started out as a demonstrator for a combination of several technologies,

[11]created by sampling the Moodplay dataset to obtain songs as equally distributed as possible on the two-dimensional arousal-valence plane [1].

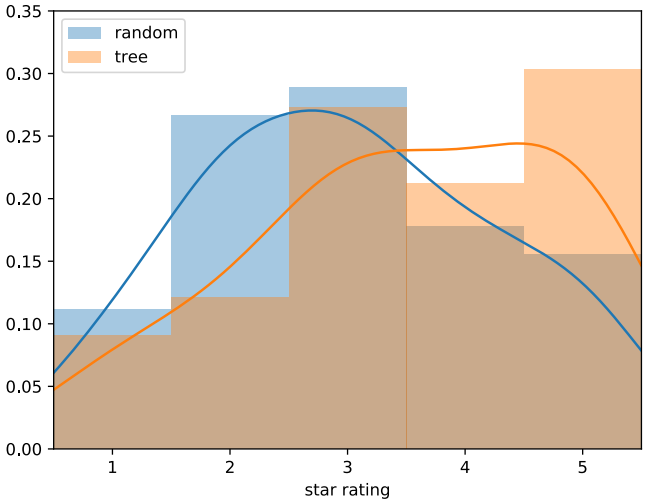

Figure 5: Histograms and Gaussian kernel density estimates for the ratings of tree-informed and random decisions collected during the brief study.

including the piper framework, dynamic music objects, and semantic web technologies, there are various further directions that the work may lead us. First and foremost, the opportunity of releasing the app online and letting users use the app with their own music collections, will enable us to collect a larger amount of transition ratings along with the high-level features for each track pair.

With these data we will be able to experiment with different learning methods and learning goals. For example, in addition to learning the transition types based on features and ratings, future versions will also learn transition parameters, which include duration, effect level, or offset, as described above.

Then, more transition techniques can easily be incorporated, such as *beat masher*, *loop roll*, *chopping*, *juggling*, *teasing*, *reverse cueing*, or *looping*, and give users more control over which set of transitions the app will selected from. At some point, users could also define their own transition types via a dedicated interface and subsequently share them with other users. Due to the used internal representation, the entire mix could also easily be saved and shared with others.

The interface could also be modified to accept lists of files to be cued, which would then play autonomously for a while, which would also benefit from an implementation of a cue point detection logic, as it exists in other applications. By giving user the choice to use their own music and decide on a succession, the transition ratings could also be used for studying playlisting, i.e. choosing the best suited track among a few options, as it was done in [10], however, with a focus on a potential ease of transition rather than a mere sequence.

8. ACKNOWLEDGMENTS

This paper is supported by EPSRC Grant EP/ L019981/1, Fusing Audio and Semantic Technologies for Intelligent Music Production and Consumption. Mark B. Sandler acknowledges the support of the Royal Society as a recipient of a Wolfson Research Merit Award.

9. REFERENCES

[1] M. Barthet, G. Fazekas, A. Allik, F. Thalmann, and M. B Sandler. From interactive to adaptive mood-based music listening experiences in social or personal contexts. *Journal of the Audio Engineering Society*, 64(9):673–682, 2016.

[2] R. M. Bittner, M. Gu, G. Hernandez, E. J. Humphrey, T. Jehan, H. McCurry, and N. Montecchio. Automatic playlist sequencing and transitions. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*, pages 442–448, 2017.

[3] G. Bonnin and D. Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):26, 2015.

[4] D. Cliff. Hang the dj: Automatic sequencing and seamless mixing of dance-music tracks. *Hp Laboratories Technical Report Hpl*, 104, 2000.

[5] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 185–200. ACM, 2017.

[6] T. Hirai, H. Doi, and S. Morishima. Musicmixer: computer-aided dj system based on an automatic song mixing. In *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, page 41. ACM, 2015.

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

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

[9] S. Letz, Y. Orlarey, and D. Fober. Compiling faust audio dsp code to webassembly. In *Proceedings of 3rd Web Audio Conference, London*, 2017.

[10] E. Liebman, M. Saar-Tsechansky, and P. Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 591–599. IFAAMAS, 2015.

[11] H.-Y. Lin, Y.-T. Lin, M.-C. Tien, and J.-L. Wu. Music paste: Concatenating music clips based on chroma and rhythm features. In *ISMIR*, pages 213–218. Citeseer, 2009.

[12] F. Thalmann, A. Perez Carillo, G. Fazekas, G. A. Wiggins, and M. Sandler. The mobile audio ontology: Experiencing dynamic music objects on mobile devices. In *Tenth IEEE International Conference on Semantic Computing*, Laguna Hills, CA, 2016.

[13] L. Thompson, C. Cannam, and M. Sandler. Piper: Audio feature extraction in browser and mobile applications. In *Proceedings of 3rd Web Audio Conference, London*, 2017.

[14] L. Vande Veire. From raw audio to a seamless mix: an artificial intelligence approach to creating an automated dj system. Master's thesis, Ghent University, 2017.