AutoMashUpper: Automatic Creation of Multi-Song Music Mashups

Matthew E. P. Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto

***Abstract*—In this paper we present a system, AutoMashUpper, for making multi-song music mashups. Central to our system is a measure of "mashability" calculated between phrase sections of an input song and songs in a music collection. We define mashability in terms of harmonic and rhythmic similarity and a measure of spectral balance. The principal novelty in our approach centres on the determination of how elements of songs can be made fit together using key transposition and tempo modification, rather than based on their unaltered properties. In this way, the properties of two songs used to model their mashability can be altered with respect to transformations performed to maximize their perceptual compatibility. AutoMashUpper has a user interface to allow users to control the parameterization of the mashability estimation. It allows users to define ranges for key shifts and tempo as well as adding, changing or removing elements from the created mashups. We evaluate AutoMashUpper by its ability to reliably segment music signals into phrase sections, and also via a listening test to examine the relationship between estimated mashability and user enjoyment.**

***Index Terms*—Audio user interfaces, creative MIR, music remixing, music signal processing.**

I. Introduction

A RECENT emerging area of activity within the music information retrieval (MIR) community is in the field of *creative-MIR* [1]. One of the main goals of creative-MIR is to open new possibilities for music creation, interaction and manipulation, facilitated by the ability to robustly analyze and interpret music signals. The facet of creative-MIR we explore in this paper relates to the content-based manipulation of music signals, specifically, that of automatic music mashups.

Music mashups are a combination of two or more songs, which, when mixed together, create entertaining musical results [2]. Mashups offer a way for users to re-engage with existing and familiar musical content by adding some extra, complementary musical components. Perhaps the most well-known way to create mashups is to take an "a cappella" stem (isolated vocal track) from one song and mix it together with another song, while adjusting the tempo so the two are temporally synchronized. Alternatively, mashups can be made by mixing complete songs together, which is the area of mashup creation we address in this paper. In this case, a larger number of musical components (i.e., vocals and all instrumentation) must fit together to create a pleasing musical result. On this basis, we consider this method for mashup creation to be more challenging, but also one that is more readily applicable to users' personal music collections.

For users wishing to listen to or create mashups by hand, several online resources exist. These include websites for uploading completed mashups [3], and those with multi-track recordings for remixing music [4]. The DJ Mix Generator website [5] contains a database of tempo and key signature information to allow users to discover songs for "harmonic mixing"–i.e., mixing songs together in the same tempo and compatible key signatures.

Regarding software tools for making the mashups and remixing music, these include both research-based and commercial systems. While quite limited in number, the research-based approaches address important technical aspects of music mixing and mashup creation, including interactive user interfaces [6], computational feasibility of time-stretching multiple songs in parallel [7] and the need for accurate tempo synchronization for user appreciation [8].

With regard to commercial systems, the Harmonic Mixing Tool [9] uses content based analysis to analyze a user's collection to allow a "harmonic fade" when mixing between songs. The Echonest Remix API has been used to create web services for remixing and manipulating music signals, including the Infinite Jukebox [10], Autocanonizer [11] and Wub Machine [12]. Perhaps the most similar system to the work presented in this paper, is Mixed in Key Mashup [13]. Central to this system is a measure of harmonic compatibility (using a similar model to [5]) which ranks songs according to how well they will sound mixed together, such that: "*A value of 100 means the mashup will be harmonic. A value of 0 means it will be dissonant.*" [13]. While this system automates the measurement of harmonic compatibility and beat-matching for temporal synchronization, users must construct the mashups themselves aligning songs by hand. In addition, the harmonic compatibility is a global property, which does not consider *where* within a harmonically compatible song the best match might occur. Furthermore, there are no parameters for users to customize the matching nor any apparent model of rhythm compatibility beyond beat-matching.

In this paper, we consider mashup creation from a multi-song perspective, where our proposed system, AutoMashUpper, allows users to create mashups by mixing different songs at distinct regions of an input song. To this end, we first segment

Manuscript received November 22, 2013; revised April 30, 2014; accepted July 28, 2014. Date of publication August 12, 2014; date of current version August 23, 2014. This work was supported by OngaCrest, CREST, JST . The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Laurent Daudet.

The authors are with the National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba 305-8568, Japan (e-mail: automashupper-ml@aist.go.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2014.2347135

the input song into phrase sections and then within each phrase section we estimate the "mashability" across a set of candidate songs. Our mashability measure considers three properties: i) harmonic compatibility, ii) rhythmic compatibility and iii) spectral balance. Novel to our approach is that mashability is calculated locally between songs, meaning the best matches across the duration of songs can be found. In addition we consider harmonic compatibility over a range of key shifts to expand the ways in which songs can be mashed together.

Having calculated the mashability per phrase section for a song, we create a ranked list from the set of candidate songs. We select the song with the highest mashability for each section, and then use time-stretching to temporally align the components of the mashup, pitch shifting for any required key transposition and amplitude scaling to match the loudness of the input section. To allow users to modify how mashability is calculated and to manipulate the mashup once it has been created, AutoMashUpper has an interactive user interface.

The main contributions beyond those in our previous work [14] include: i) a fast implementation for the calculation of harmonic similarity, ii) the addition of rhythmic and loudness based components for mashability estimation, iii) an objective evaluation of the phrase-level segmentation, and iv) a subjective evaluation of mashability.

The remainder of this paper is structured as follows. In Section II we provide a system-level overview of the underlying music analysis techniques for beat tracking, downbeat estimation and phrase segmentation. In Section III we address the measurement of mashability. In Section IV we demonstrate the user interface and several usage scenarios for AutoMashUpper. In Section V we present both objective and subjective evaluation of our mashup system. The paper concludes with discussion in Section VI and conclusions in Section VII.

II. Music Signal Analysis

The main goal of the music analysis stage is to determine the temporal framework around which a multi-song mashup can be created. To this end, we need to identify beat locations for later temporal synchronization and a set of section boundaries to partition the input song for multi-song mashup creation. We can consider these pre-processing aspects of the analysis as modular components for which any existing structural segmentation algorithm (e.g., [15]) and beat tracker (e.g., [16]) could be used. However, with the aim of presenting an overview of the entire AutoMashUpper system we now summarize the methods we use for this purpose. Specific to our proposed method, we estimate downbeats in addition to beat locations towards finding a *phrase-level* segmentation of the input. In this paper, we consider phrase sections to correspond to harmonically consistent regions of the input signal which begin on downbeats and whose duration is a complete number of bars (typically, 2, 4, or 8). Furthermore, we make the general assumption that the songs to be used in the mashups have approximately constant tempo and a fixed 4/4 time signature.

To generate the harmonic signal representations for phrase-level segmentation and the subsequent estimation of mashability, we use the NNLS Chroma plugin [17] within Sonic Annotator [18]. Given an input audio signal[1], we extract three outputs from the NNLS Chroma Vamp plugin: the global tuning, t, of the input, an 84-bin (7 octave), tuned semitone spectrogram, X, and a 12-dimensional chromagram (the distribution of energy across the chromatic pitch classes in a musical octave), C. All outputs are extracted using the default parameters.

A. Rhythm Analysis

1) Input Features: To act as input for the extraction of beat and downbeats, and subsequently used in the modelling of rhythmic patterns, we extract two onset detection functions [19] from the input audio signal. We use the approach of Robertson *et al.* [20] who employ Fitzgerald's harmonic/percussive separation technique [21] to generate a kick and snare drum emphasis onset detection functions, K_{DF}, and S_{DF} respectively. Each onset detection function has a temporal resolution of 11.6 ms.

2) Beat Tracking: To extract the beat locations, we use a simplified version of the approach in [22], to which we pass a single input feature: the sum of the kick drum onset detection function and the snare drum onset detection function extracted above. Since we assume the tempo is likely to be approximately constant, we extract a global estimate of the beat period. We pass this to the dynamic programming phase as proposed by Ellis [16] to determine the beat locations, γ.

3) Downbeat Tracking: To find the downbeat locations we extend the approach of Davies and Plumbley [23] inspired by [24]. The core of their approach was to measure the spectral difference between band-limited beat-synchronous spectral frames, and to determine downbeat as the beat transitions which consistently led to highest spectral change under the assumption of a constant number of beats per bar. We modify this method in two ways, first by median averaging the semitone spectrogram, X, into beat-synchronous frames, X_{γ}, and using this to measure spectral difference. Then, to this beat spectral difference function, we incorporate additional information from the kick and snare onset detection functions, extracted in Section II-A1. We use the hypothesis from [24] that kick drum events are more likely, and snare events less likely, to occur at downbeat locations and hence add the amplitude of the K_{DF} at each beat, and subtract the amplitude of the S_{DF} from the beat spectral difference function. In this way, we generate a downbeat detection function which incorporates both harmonic change and basic rhythmic information from which determine the downbeats, Γ, as in [23].

B. Phrase Level Segmentation

Our motivation for a phrase-level segmentation is to partition the input song so that we can subsequently estimate mashability independently per section and then incorporate multiple songs into the eventual mashup. In our experience, we have found that pleasing results can be obtained when the transition between different songs in the mashup occurs on precise temporal boundaries between sections. On this basis, we make use of both previously extracted beat and downbeat times towards creating

[1]While the input audio signal for AutoMashUpper can be either mono or stereo, for the music analysis stage stereo signals mixed down to mono. However, for creating the final mashups, stereo versions are used.

a downbeat-synchronous representation for finding phrase section boundaries.

First, we retrieve the beat-synchronous version of the 84 bin semitone spectrogram, X_γ, extracted in Section II-A3. Then, to extract phrase section boundaries, we follow the self-similarity and novelty function based approach of Foote [25] for structural segmentation. As shown in [26], structural segmentation performance can be improved by time-lag embedding, where temporal frames are stacked together. To this end, we group sets of four consecutive beat frames (starting at each downbeat and without overlap) to create a ($84 \times 4 = 336$ bin) downbeat-synchronous stacked semitone spectrogram, X_Γ. We calculate a self-similarity matrix, D, between all pairwise combinations of downbeat-synchronous frames of X_Γ using cosine distance:

$$D(i,j) = 1 - \frac{X_{\Gamma,i} \cdot X_{\Gamma,j}}{||X_{\Gamma,i}||\ ||X_{\Gamma,j}||} \tag{1}$$

where $X_{\Gamma,i}$ and $X_{\Gamma,j}$ correspond to the ith and jth frames of the downbeat synchronous semitone spectrogram.

Having generated the self-similarity matrix, we then slide a Gaussian checkerboard kernel [25] (of size 16 downbeats) along the main diagonal to generate the novelty function which emphasizes section boundaries. We peak-pick [19] the novelty function to obtain an initial set of section boundary candidates. Following Sargent *et al.* [27], who impose regularity of section length into their structural segmentation method, we post-process the initial boundary locations according to an iterative method which we now summarize. We define a cost function relating to phrase section duration intervals which rewards durations of 2, 4, 8 and 16 downbeats and punishes others we consider less likely (e.g., 7, 9, 15, 17 downbeats). We then calculate a regularity score for each phrase boundary position independently shifted forward one downbeat, kept in its current position, and shifted back one downbeat. The phrase boundary shift which maximizes the increase in global regularity score is implemented, and the process is iteratively repeated until no further increase in the global regularity score can be obtained. Once this iterative process has completed, the downbeat phrase boundaries are mapped back to beat locations ready for the estimation of mashability. A graphical example of this procedure is shown in Fig. 1.

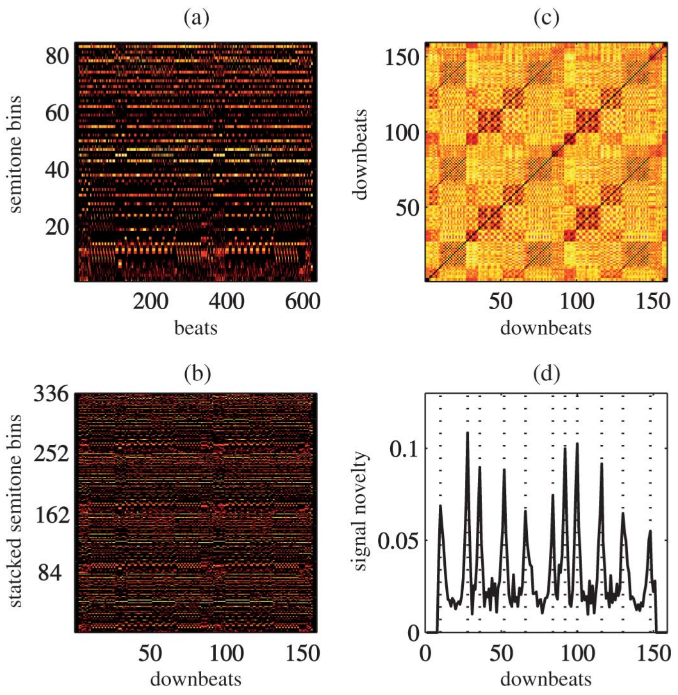


Fig. 1. Phrase-level segmentation overview. (a) a beat synchronous tuned semitone spectrogram. (b) a downbeat-synchronous spectrogram, where groups of four beat frames are stacked into bars. (c) a self-similarity matrix generated from the downbeat-synchronous semitone spectrogram. (d) a novelty function whose peaks highlight likely phrase boundaries. The vertical dotted lines show the detected phrase boundaries.

III. Mashability

Given a phrase section, p, of input song, i, and a set of candidate songs, n, in a music collection, the estimation of mashability, M, addresses two main points: i) the temporal location of the best matching point for each candidate song and ii) the way in which this best matching section must be transformed to match the input. We propose that songs which are well matched for mashup purposes will have the following properties:

1) Harmonic compatibility: central to the success of the mashup is that the two sections of the song complement one another in terms of harmonic structure. Our harmonic motivation follows directly from existing commercial tools for "harmonic mixing" [5], [9] and mashup creation [13] which place strong emphasis on finding songs in matching or related key signatures. For AutoMashUpper we do not consider key signature directly, but instead look for a measure of similarity between beat-synchronous chromagrams under a range of possible key transpositions.
2) Rhythmic compatibility: beyond the critical aspect of beat-matching for mixing songs together [8], we also consider the issue of rhythmic structure. Our aim here is to capture rhythmic information at a finer level than the beats towards the identification of rhythmically compatible regions of songs. In particular, we seek to avoid matching songs with conflicting rhythmic structures, e.g. straight feel vs swing feel.
3) Spectral Balance: as shown in [28 pp.53–54], equalization plays a key role when mixing songs together, allowing for different regions of songs to be layered on top of one another in the mix. Towards this goal of creating a balanced mix between songs, we incorporate a measurement of spectral balance between songs, favouring those with a complementary distribution of energy across low, middle and high bands.

The specific details of each component are described in the following sub-sections. However prior to this, we first consider the effect of global tempo. As shown in [8], when mixing music, it can be preferable to take tempo octave relationships into account. For example, if we consider a song at 70 bpm (beats per minute) and another at 140 bpm, it would be possible to use time-stretching to slow down the faster song by a factor of two to give it a tempo of 70 bpm and hence exactly matching the slower song. However due to the tempo octave relationship, we could simply mix the faster song with the slower song directly without the need for any time-stretching. While this is somewhat trivial for music with an exact 1:2 tempo ratio, if we consider the same original song at 70 bpm, and another at 130 bpm, by similar logic we should argue for *increasing* the tempo of the faster track to 140 bpm, rather than attempting to slow it down to 70 bpm, since this will represent the smaller time-scaling factor (and would hence minimize any artefacts introduced from time-stretching). Within the context of our mashup system, we can make all songs whose tempo is closer to another tempo octave of the input appear to be at the same tempo by sub-sampling the beat locations by a factor of two to halve the tempo, or interpolating by two to double it. In addition to changing the beats in this way we apply the same transformation to each of the beat synchronous features and representations which are used for mashability estimation and are described below.

A. Harmonic Matching

The first component of mashability we consider relates to the harmonic similarity calculated between beat-synchronous chromagrams. As in Section II-A3 we obtain a beat-synchronous chromagram, C_γ, by median averaging across beat windows. Next, we isolate the current phrase section, $C_{i,p}$, for input song, i, and phrase section p, with K beat frames (omitting the sub-script γ for clarity). For a given candidate song, n, in the music collection, with beat synchronous chromagram C_n we wish to identify the temporal location (i.e. starting beat) where harmonic similarity is maximized. However, in addition to looking at the current properties of the two songs, we also consider how the songs could match allowing for key-transposition (later implemented by pitch-shifting). In this way we can increase the possibility of finding good harmonic matches by considering the matching task in a kind of transform domain.

In our previous work [14], we implemented this approach by brute force, measuring the cosine similarity across a range of rotational shifts, q, of the input phrase chromagram,

$$H_n(q,k) = \frac{C_{i,p,q} \cdot C_{n,k}}{||C_{i,p,q}||\ ||C_{n,k}||} \tag{2}$$

where $C_{n,k}$ represents the region of the chromagram starting at beat k whose duration is the same as $C_{i,p}$. In practice we found this calculation to be computationally intensive, and computation time increased drastically in proportion to the number of songs over which the matching was undertaken and the range of rotational key shifts applied to $C_{i,p}$. Towards a more efficient solution, we recast the harmonic matching from 2-D filtering (image processing) perspective. We now treat $C_{i,p}$ as a *chroma patch* (or two-dimensional filter) to cross-correlate with C_n. In order for the chroma patch to cover the range of key shifts, we stack an exact copy of C_n, creating a $(24 \times K_n)$ matrix for K_n beats in C_n. Using this 2-D filtering operation (via MATLAB's multi-theaded conv2() function) means that we can search across all beat increments and harmonic shifts very efficiently.

To illustrate the difference in computation time between the brute-force search method in [14] and the 2-D convolution approach, we measure the computation time (averaged over 100 iterations) for chroma patches of size 8 to 64 beats applied to songs of up to 1000 beats. As shown in Fig. 2 we see that under all conditions, the 2-D convolution approach is substantially faster with computation time measured in milliseconds rather than seconds. Furthermore, the 2-D convolution method is faster for smaller patch sizes, regardless of the number of beats in the

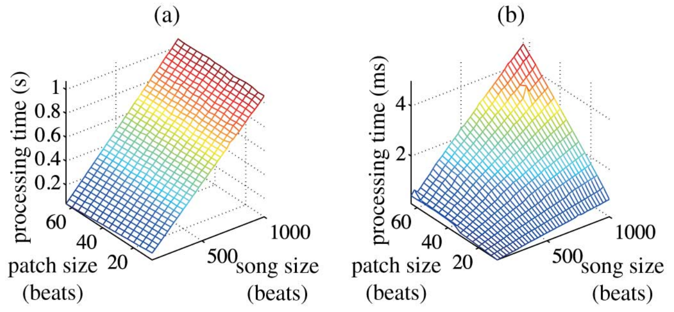


Fig. 2. Comparison of computation time for the brute force "for loop" approach from [14] with the proposed 2-D convolution method. (a) brute force "for loop" (b) proposed conv2 approach.

song. If we consider a full range of 12 key transpositions and approximately 400 beats in a song, this equates to close to 5000 possible solutions per phrase section of the input song. When searching over a small sized collection of 500 songs, this would give over 2 million possible matches per phrase section. Referring again to Fig. 2, we can see that the brute force approach is too slow to be considered practical.

To obtain a cosine similarity matrix (with values in the range [0, 1]), $H_n(q,k)$ as in (2) we must normalize the result of the 2-D convolution using the denominator from (2) and select the appropriate rows corresponding to the range of key shifts, e.g. $q \in -6, \ldots, +6$ semitones. Given $H_n(q,k)$ the final step is to extract the maximum harmonic similarity per beat $M_{H,n}(k)$, and to record the corresponding key shift $Q_n(k)$ necessary to realize this similarity value,

$$M_{H,n}(k) = \max_q(H_n(q,k)) \tag{3}$$

and

$$Q_n(k) = -\arg\max_q(H_n(q,k)), \tag{4}$$

where the transposition is negative since we later pitch-shift the region of the candidate song to match the input, which is the reverse harmonic matching process. A graphical example is shown in Fig. 3, where we can see that best matching section of the candidate song chroma occurs at a $+2$ semitone shift away from the input phrase section, which would subsequently require pitch shifting by -2 semitones to create the harmonic match. From the plot of $M_{H,n}(k)$ in Fig. 3(e), we can see that harmonic similarity varies across the duration of the song, where the region between 250 and 300 beats appears not to hold any good match to the input phrase across all possible key shifts.

B. Rhythmic Matching

For the rhythmic matching stage we follow the same principle as the harmonic matching. Here we generate a rhythmic representation for the input song and all candidate songs in the user's collection and measure the similarity between them. Our main source of rhythmic information is the kick and snare onset detection functions ($K_{\rm DF}$ and $S_{\rm DF}$) used in the estimation of beat times in Section II-A. In this way, we use an implicit representation of rhythm, rather than attempt to classify rhythmic patterns according labelled classes as in [29].

To capture rhythmic information which is independent of tempo we generate a beat-synchronous rhythmic representation

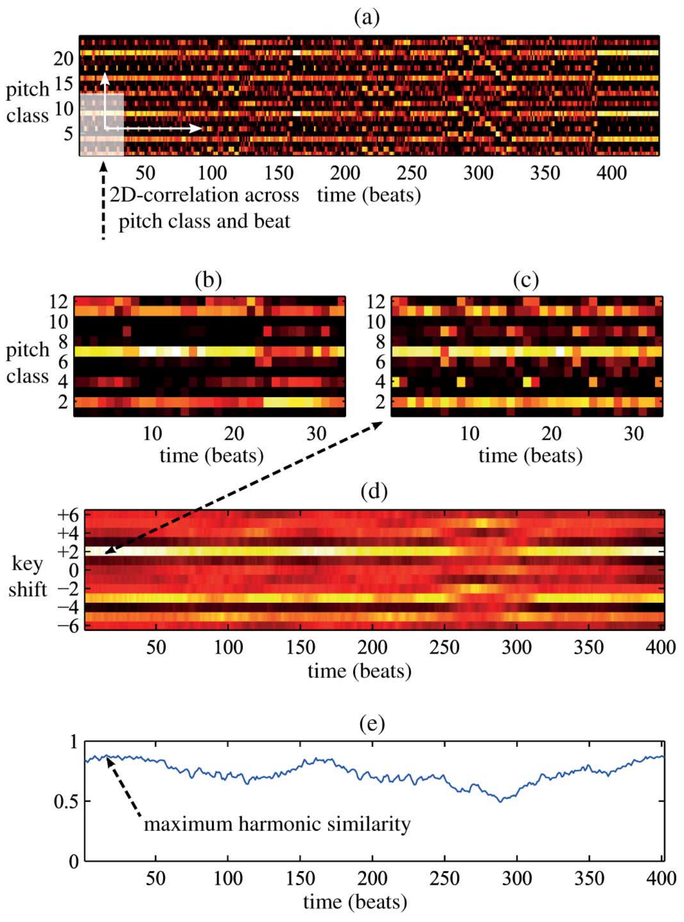


Fig. 3. Harmonic matching overview. (a) Stacked candidate song chromagram, (b) chromagram for a 32-beat phrase section the input song, (c) the corresponding best matching chroma patch (after a +2 semitone pitch shift) extracted as the global maximum of (d) the harmonic similarity matrix, and (e) the overall maximum harmonic similarity per beat shift, $M_{H,n}(k)$.

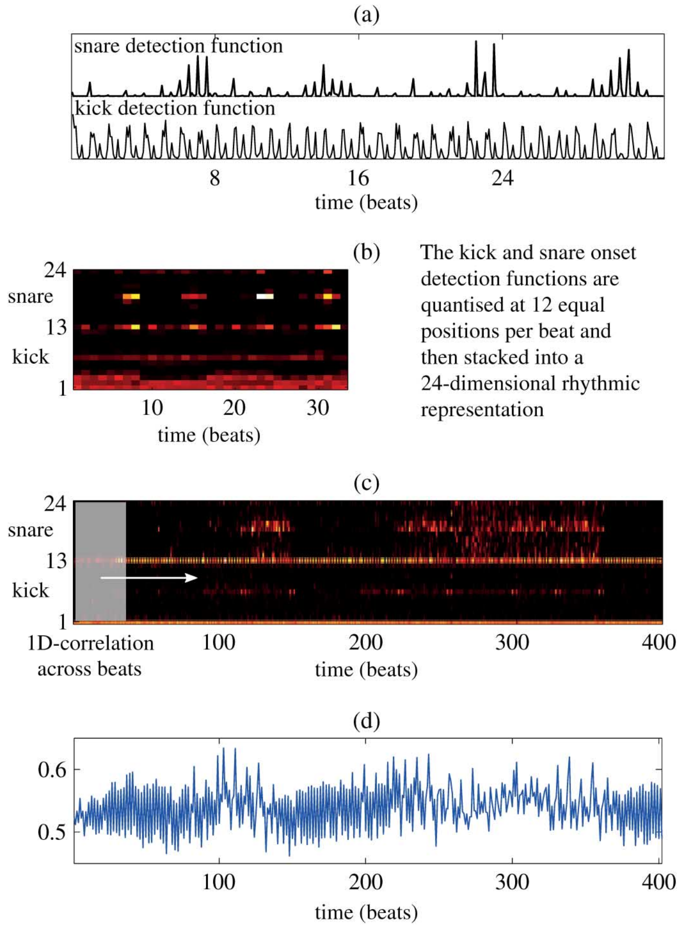


Fig. 4. Rhythmic matching overview. (a) Kick, K_{DF} and snare, S_{DF}, onset detection functions, (b) quantized rhythmic representation for input phrase section, (c) the beat synchronous rhythmic representation for the candidate song, and (d) the rhythmic similarity per beat shift, $M_{R,n}(k)$.

by sampling these two onset detection functions at sub-beat locations. To provide sufficient granularity to distinguish between rhythmic patterns in "straight" and "swing" feel, we sub-divide each beat into 12 equal positions, and record the amplitude of each onset detection function at these temporal locations. Then, per beat, we stack snare information on top of the kick information to provide a 24-dimensional rhythmic representation per beat, $R_{i,p}$, of size $(24 \times K)$ for K beat locations in phrase section p on input song i, and likewise calculate the same representation, R_n, for all songs, n, in the user's collection.

To determine the rhythmic matching component of the mashability, $M_{R,n}(k)$, we calculate the cosine similarity of the rhythmic patch, $R_{i,p}$, for phrase-section p of the input song i, with R_n across all beat shifts, k, for each song n in the user's collection. We follow the same fast matching approach for matching that we used for harmonic information, however in this case we need only search across the beat dimension,

$$M_{R,n}(k) = \frac{R_{i,p} \cdot R_{n,k}}{||R_{i,p}||\ ||R_{n,k}||}. \tag{5}$$

In this way, the rhythmic similarity measure, $M_{R,n}(k)$, will be close to unity for very similar rhythmic patterns, and nearer to zero for dissimilar patterns. An example of the rhythmic matching procedure is shown in Fig. 4.

C. Spectral Balance

To address the third of the mashability components we use the Replay Gain method [30] and previously extracted beat locations per song, to derive a coarse beat-synchronous spectral representation, L, which contains the beat-synchronous perceptual loudness in three spectral bands: a low band (up to 220 Hz), a mid-band (220-1760 Hz), and a high-band (frequencies above 1760 Hz). In our approach for assessing spectral balance between $L_{i,p}$ (the spectral representation of phrase section, p) and L_n for song n we look for a simple measure of spectral flatness by summing the coarse spectral representations and taking the mean across the beat dimension,

$$\beta_k = \frac{1}{K_p} \sum_{k=1}^{K_p} L_{i,p} + L_{n,k}. \tag{6}$$

We then normalize β_k to sum to unity, and find the standard deviation of the resulting spectral profile,

$$M_{L,n}(k) = 1 - \mathrm{std}(\beta_k). \tag{7}$$

In practice, $M_{L,n}(k)$ will be close to unity when the spectral profile is flat and nearer to zero when it is uneven. A graphical example is shown in Fig. 5.

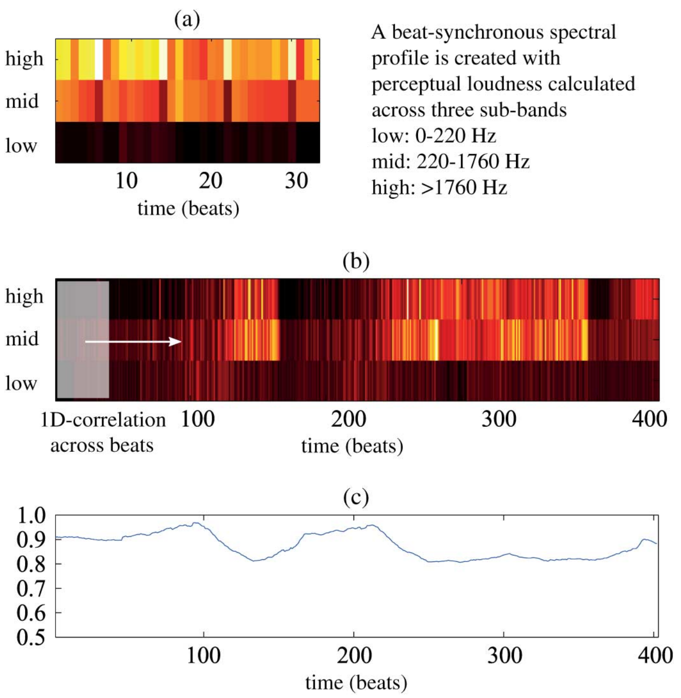


Fig. 5. Spectral balance overview. (a) beat synchronous spectral representation for input phrase section, (b) the beat synchronous spectral representation for the candidate song, and (c) the spectral balance per beat shift, $M_{L,n}(k)$.

D. Mashability Estimation

We derive the overall beat-wise mashability, $M_n(k)$, per song, n, as a weighted combination of the harmonic similarity, rhythmic similarity and spectral balance. The values for the weights, $w = \{w_H, w_R, w_L\}$ can be set in the user interface, as described in Section IV, but in our informal testing, we found acceptable results with the following values, $w_H = 1$, $w_R = 0.2$ and $w_L = 0.2$, wherein we give greatest emphasis to the harmonic matching parameter. We obtain a weighted mashability per beat increment, k, in song n, as follows:

$$M_n(k) = w_H M_{H,n}(k) + w_R M_{R,n}(k) + w_L M_{L,n}(k). \quad (8)$$

In addition to these three local features we incorporate one further component to reward songs whose tempo, T_n is within a user specified range, η, (default value 0.3) around the input song tempo, T_i, such that

$$M_n(k) = \begin{cases} M_n(k) + \alpha, & \text{if} |1 - |(T_i/T_n)|| \leq \eta \\ M_n(k), & \text{otherwise} \end{cases} \quad (9)$$

where $\alpha = 0.2$ was found to give acceptable results.

For each phrase section, p, of the input song, i, we now have a continuous vector $M_n(k)$ which indicates the mashability for every beat increment, k, in every song n, along with the corresponding required key shift $Q_n(k)$ from (4). To determine, *which* song, *where* in that song, and *how* to transform it, we first find the maximum mashability per song,

$$M_{\max}(n) = \max_k(M_n(k)) \quad (10)$$

and record the corresponding beat,

$$k_{\max}(n) = \arg\max_k(M_n(k)) \quad (11)$$

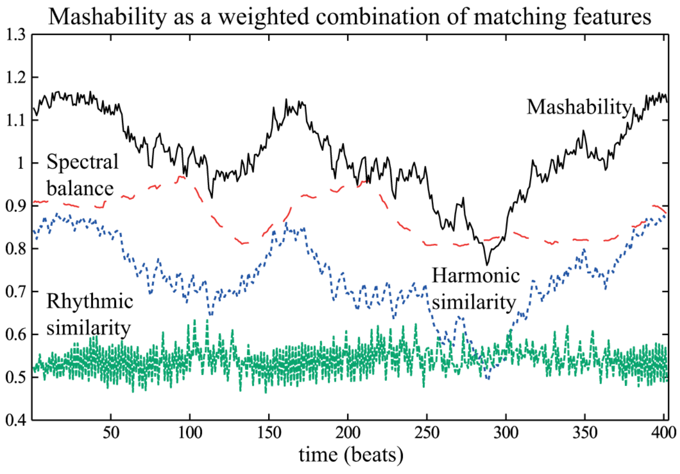


Fig. 6. Mashability per beat shift in a candidate song. Due to the high emphasis given to the harmonic matching component, mashability bears a strong resemblance to this feature. However, the emphasis of the mashability can be changed by user modification of the weights.

which can be used to recover the corresponding key shift,

$$q_{\max}(n) = Q_n(k_{\max}). \quad (12)$$

For all songs, n, we now have a corresponding mashability value, and by sorting $M_{\max}(n)$ in descending order, we can rank the mashability per phrase section of the input.

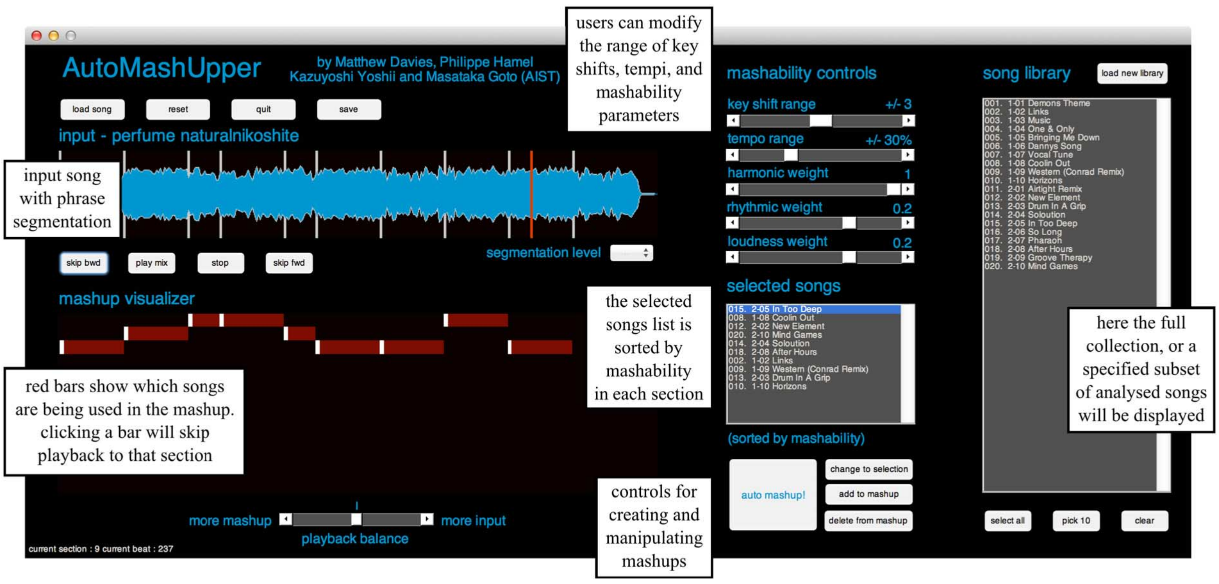
A graphical example of the mashability for a given song is shown in Fig. 6. Here we can see the strong correlation with the harmonic matching function due to the high weighting we give this in comparison to the other features. It is interesting to note that the separate features which contribute to mashability have different properties. The rhythmic similarity rapidly varies as the rhythmic patterns change from being beat aligned to mis-aligned, whereas the spectral balance evolves much more slowly across the duration of the song. The peaks in the harmonic similarity function correspond to time points in which chord changes are aligned and occur around every eight beats.

E. Mashup Creation

The creation of the mashup is split into three steps: i) temporal alignment (or "beat-matching") using time-stretching, ii) harmonic alignment and tuning adjustment using pitch-shifting, and iii) loudness adjustment to create a balanced mix. For time-stretching and pitch-shifting we make use of the open-source library Rubberband [31], whereas for loudness compensation we use the Replay Gain method [30].

The beat-matching stage takes two inputs, the sequence of beat times in each phrase section, p, of the input song, i, and beat times from the best matching section from the song in the music collection. The input beat times act as a target sequence of anchor points for which Rubberband calculates a dynamic time-stretching factor such that the best matching section is temporally aligned to the input.

The harmonic alignment between the two sections, implemented via pitch-shifting in Rubberband, requires two pieces of information. First, the required number of semitones and second, a tuning correction. They can be combined to give a pitch correction factor, $f = t_n/t_i \times 2^{(q_{\max}(n)/12)}$, where t_n and t_i are the tunings (in Hz) of the best matching song and input



users can modify the range of key shifts, tempi, and mashability parameters

input song with phrase segmentation

red bars show which songs are being used in the mashup. clicking a bar will skip playback to that section

the selected songs list is sorted by mashability in each section

controls for creating and manipulating mashups

here the full collection, or a specified subset of analysed songs will be displayed

Selected songs:

015. 2-05 In Too Deep
008. 1-08 Coolin Out
012. 2-02 New Element
020. 2-10 Mind Games
014. 2-04 Soloution
018. 2-08 After Hours
002. 1-02 Links
009. 1-09 Western (Conrad Remix)
013. 2-03 Drum In A Grip
010. 1-10 Horizons

Song library:

001. 1-01 Demons Theme
002. 1-02 Links
003. 1-03 Music
004. 1-04 One & Only
005. 1-05 Bringing Me Down
006. 1-06 Dannys Song
007. 1-07 Vocal Tune
008. 1-08 Coolin Out
009. 1-09 Western (Conrad Remix)
010. 1-10 Horizons
011. 2-01 Airtight Remix
012. 2-02 New Element
013. 2-03 Drum In A Grip
014. 2-04 Soloution
015. 2-05 In Too Deep
016. 2-06 So Long
017. 2-07 Pharaoh
018. 2-08 After Hours
019. 2-09 Groove Therapy
020. 2-10 Mind Games

Fig. 7. Screenshot of the AutoMashUpper user interface. Additional descriptions of the functionality are overlaid.

respectively as estimated by NNLS [17], and $q_{\max}(n)$ is the required semitone shift. If $q_{\max}(n) = 0$ and the ratio of the tunings is small (i.e., within 0.5% of one another) then we bypass this processing stage.

After pitch-shifting and time-stretching we scale the amplitude of the best matching section to match that of the input. For this we estimate the perceptual loudness of both sections using Replay Gain [30] and scale the amplitude of the best matching section so the two are equal. We employ each of these methods on section-by-section basis in the input song and create a composite audio file containing the mashup accompaniment, which can be subsequently mixed down to create the completed mashup.

The computational demands of the complete mashup creation process depend on several factors. For mashability estimation this includes the number of phrase sections and their size, the number of beats per candidate song, as well as the total number of candidate songs. The computational cost for time-stretching and pitch-shifting increases in proportion with the number of sections, and hence overall length, of the input song. As a guide, we have found the complete mashup process to take less than 30 s for a 4 minute input song with 10–15 candidate songs for our prototype version of AutoMashUpper running in MATLAB.

IV. AutoMashUpper

A. User Interface

To allow users to interact with the mashup creation process, we have created a user interface for our system, AutoMashUpper, of which a screenshot is shown in Fig. 7.

The left hand side of the interface is used for displaying the audio waveform of the input and its section boundaries, the songs used in the mashup, as well as system level functionality and playback controls. Both the waveform display and mashup visualizer are interactive and clicking either one during playback will jump to the start of the clicked section. At the bottom of the left hand side, we include a new feature, a balance control to enable users to adjust the mix between the input and mashup in real time.

On the right hand side of the interface is a list box which is used to display the set of pre-analyzed songs which can be included in mashups, either by individual selection with the mouse, choosing ten random songs (with the "pick 10" button) or selecting all. Above this list box is an option to load a different song library. We use this as a method for selecting a folder with pre-grouped songs, e.g., all from the same artist or of a particular musical style.

In the centre panel we have the interactive controls for mashability estimation. Each of these sliders can be modified to create mashups with different emphasis, and to encourage users to experiment with mashup creation. Below these controls is another list box, which shows the subset of songs selected from the main library listbox, and then beneath this are the tools available to first create the mashup and then subsequent manipulation of the result. This manipulation includes deleting, changing, or adding songs to the created mashup.

B. Modes of Operation

The standard usage we expect for AutoMashUpper is for a user to load a song of their choice, select a set of songs from which to make the mashup, and then modify it according to their taste afterwards. Through experimentation with the system, we have found several alternative ways to use AutoMashUpper to make interesting musical results.

- **Album/Artist mode**: we manually select songs for the mashup from a single album or artist. Our testing has shown this mode can give subtle results due to high similarity in vocal timbre and overall music production. Furthermore by processing all songs in an album in this way, it is possible to create an entire remix/mashup album, which can offer listeners a new way to re-engage with familiar material.
- **Style mode**: in a similar way to the Album/Artist mode, we can exclusively use songs from a given musical style as source material for the mashup. For example, taking a J-Pop song and creating a mashup using only Drum and Bass music or vice-versa.

- **Forced-mashup mode**: while the quality of the mashups created by AutoMashUpper is somewhat dependent on the large search space created in the estimation of mashability, we can alternatively use AutoMashUpper in a much more restricted search mode, where we give just one song to be used in the mashup along with the input. Here, AutoMashUpper must "force" a match between the signals, and offers the potential for unexpected results when combining very different pieces of music, for example a modern piece of electronic dance music and a 1960s jazz ballad.
- **Musician mode**: here, we take the conventional method of using an a cappella vocal stem in a mashup, but instead of vocals, we use a recording of an isolated musical instrument. To this end, we have obtained interesting results using recordings of solo slap bass playing, and creating automatic slap bass accompaniment mashups. This approach is not limited to bass guitar, and we can envisage AutoMashUpper being used by musicians who could record some unaccompanied playing on a musical instrument and then use it as material for a mashup. This would allow them to hear their playing used in a new way, and give them the opportunity to "re-learn" the new arrangement. Alternatively, the inclusion of a mashed-up bass part could allow a composer to complete composition by reuse of existing material.

Some example videos demonstrating AutoMashUpper using Creative Commons music content can be accessed at: http://staff.aist.go.jp/m.goto/AutoMashUpper/.

V. Evaluation

To evaluate AutoMashUpper, we address two aspects of the system. First, we demonstrate the efficacy of the phrase-level segmentation which underpins the multi-song mashup creation process via a comparison against ground truth annotations. Then we undertake a listening test to subjectively assess the mashability estimation component of AutoMashUpper.

A. Objective Analysis

1) Test Database: To evaluate the phrase-level segmentation component of AutoMashUpper we use the copyright-cleared RWC Music Database [32] which contains beat, downbeat and structural segmentation annotations [33]. Given our primary interest in creating mashups from pop music, we restrict the analysis to the Popular Music Database which contains 100 songs (RWC-MDB-P-2001 No. 1-100). The majority of songs have a 4/4 time-signature and approximately constant tempo. Although this dataset may not be considered the most challenging, it matches the type of content we expect to be used to create mashups. While the task of structural segmentation does not exactly match our goal of phrase segmentation, we consider structural boundaries also to be phrase boundaries and on this basis, consider this evaluation task to be appropriate for our purpose.

2) Segmentation Performance: While many evaluation methods [34] exist for measuring the performance of structural segmentation algorithms, the majority attempt to address two factors, the temporal accuracy of the segment boundaries and

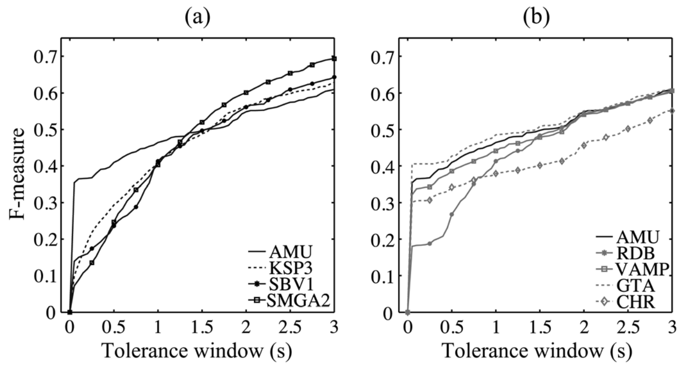


Fig. 8. Evaluation of structural segmentation performance on the RWC-POP database showing F-measure as a function of tolerance window size. (a) AutoMashUpper (AMU) compared to the three unique top performing algorithms in MIREX 2012: SMGA2 [36], KSP3 [35] and SBV1 [37]. (b) AMU compared to different parameterizations. RDB: random downbeat estimates, VAMP: using the QM-VAMP plugin [18] to estimate beat and downbeats, GTA: ground truth beat and downbeat annotations and CHR: using a chromagram for the self-similarity matrix.

the labels given to these sections. Our interest in this paper is only in the temporal localization of the segment boundaries, and with this in mind, we pursue an evaluation strategy which ignores section labels. In [27], the F-measure was used with two tolerance windows, a narrow window of 0.5s and a wider window of 3.0s. Since our goal is for high temporal precision in boundary detection, we calculate the F-measure as a function of tolerance window size. We start with a tolerance window of ± 0s and increase it by 0.05s up to a maximum tolerance window size of ± 3.0s.

We compare the performance of AutoMashUpper against the three best performing (unique) systems submitted to the 2012 MIREX structural segmentation task, KSP3 [35], SMGA2 [36] and SBV1 [37]. We did not run these algorithms ourselves, but instead we made use of the online resource of algorithm outputs[2]. To gain a deeper understanding of our proposed segmentation algorithm we also evaluate four different approaches for obtaining beat and downbeat information prior to phrase segmentation:

- RDB: which uses automatically extracted beat locations as in Section II-A2, but a random first downbeat from the first four beat estimates and then every fourth beat after that as a downbeat.
- VAMP: which uses the QM-VAMP bar and beat tracking plugin from Sonic Annotator [18] to provide beat and downbeat locations.
- GTA: here we use the ground truth beat and downbeat annotations.
- CHR: we use automatically extracted beat and downbeat locations but build the self-similarity matrix in Section II-B using the beat synchronous chromagram, C_γ, instead of the semitone spectrogram, X_γ.

The performance of each segmentation algorithm as a function of tolerance window is shown in Fig. 8.

The most immediate result of looking at the comparison against the MIREX methods in Fig. 8(a) is the large difference in accuracy for very small tolerance windows. Our approach has an F-measure of 0.35 even for a tolerance window of

[2]https://github.com/jpauwels/mirex-tools

± 0.05 s, and for this very small tolerance window, it far outperforms the current state of the art on this dataset. Comparing performance of AMU and RDB in Fig. 8(b) we can see that the use of estimated (rather than random) downbeat locations has a considerable impact on performance when the goal is to find precise temporal boundaries. Furthermore, compared to GTA, we see that segmentation performance for very small tolerance windows is marginally improved by using the ground truth beat and downbeat annotations. With all other aspects equal, we can therefore infer that beat and downbeat estimation by AMU was accurate and also slightly outperformed the freely available VAMP implementation.

In terms of the input representation for the self-similarity matrix, we can see that the use of the chromagram, whose performance is given by CHR, is less effective than the semitone spectrogram, suggesting that some useful structural information is lost by folding the seven octave semitone spectrogram into a single octave chromagram. Comparing again to the MIREX methods in Fig. 8(a), we see that once the size of the tolerance window exceeds 1.0 s, the remaining algorithms begin to outperform AutoMashUpper, with SMGA2 the most accurate once a wide tolerance window of 3.0 s is permitted. This lower performance at large tolerance windows may be the result of our emphasis on detecting shorter phrase sections. For example, the median detected phrase length across the database for AMU was 10.3 s compared to the SMGA2 algorithm it was 15.6 s and 14.3 s for the ground truth annotations.

B. Subjective Analysis

1) Listening Experiment: The aim of the listening experiment was to explore the relationship between user enjoyment of a mashup and mashability, with the hypothesis that user enjoyment would be positively correlated with mashability. While it would be theoretically desirable to have listeners listen to mashups of entire songs, and hence evaluate the complete result of AutoMashUpper, several issues make this choice impractical. First, it would be difficult to ask users to make a single judgement about their overall enjoyment of a mashup with multiple songs and sections. Second, listening to many different full songs would make for a long and potentially tiring experiment for participants. Third, the estimated mashability is empirically related to the phrase section duration–i.e., the likelihood of very high harmonic similarity is much greater for a short section of 8 beats than one which is 64 beats containing multiple chord changes, hence it is not trivial to meaningfully relate mashability between phrase sections of different lengths. In this sense the absolute value of mashability is only comparable between equal section lengths. Nevertheless, the relative ordering of mashability within each individual section can be explored. Therefore, in order to retain as much control as possible over the stimuli, we restrict ourselves to a set of musical excerpts of isolated, individual phrase sections of a fixed duration of 32 beats.

To create the dataset used in the listening experiment, we randomly selected 12 musical excerpts of duration 32 beats (using automatically estimated phrase section boundaries), from a collection of 90 songs. Then, for each excerpt, we calculated and ranked the mashability to all the other songs in the collection using the default weighting between the different components, as described in Section III-D. We then created three mashed-up

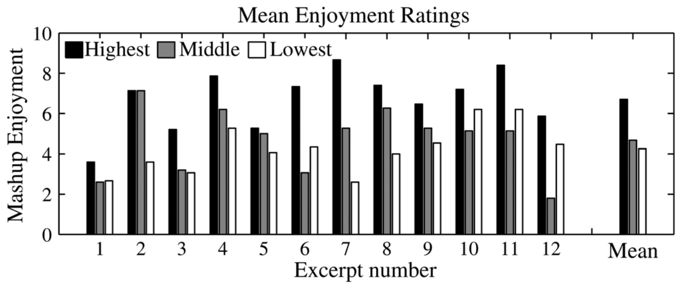


Fig. 9. Mean user enjoyment ratings per listening test excerpt. The black bars refer to the top ranked mashup, the grey bars are the middle ranked mashup and the white bars represent the lowest ranked mashups.

versions using: i) the top ranked mashability, ii) the song closest to the mean mashability, and iii) the lowest ranked mashability, to give a total of 36 musical stimuli. Our aim is to approximate a typical usage scenario for AutoMashUpper, where the best result would be the automatic result given by AutoMashUpper, and that the lower ranked results would represent weaker recommendations.

The experiment was run as follows: for each new excerpt the participants first listened to the un-mashed input by itself, after which they could listen to the mashup (i.e., the mix of the two songs), with repeated listening of both sound files permitted. Participants were then asked to input their rating on a 0–10 scale to indicate how much enjoyed the result of mashup compared to the original; in effect how successful they felt the mix of the two songs was. On this basis we felt it was very important to allow listeners hear the original as well to avoid enjoyment ratings based purely on musical taste of the listeners–a factor for which we could not control.

To allow the participants to familiarize themselves with the operation of the listening test interface and to set the playback volume to a comfortable level, each participant undertook a short training phase prior to completing the main experiment. In total, a set of 15 participants were recruited to take the listening test. Since we hope that mashups can be appreciated irrespective of musical training, we did not use this as a criterion for selecting participants, however we did ensure that the participants understood what the term "music mashup" meant. To prevent order effects, the musical stimuli were presented in a different random order for each participant. Participants were not paid for taking part in the experiment.

2) Results: On completion of the listening test, participants were asked their general opinion about the mashups. Many were very positive, stating that some mashups were highly enjoyable to listen to and represented interesting musical combinations. Several participants commented that for certain excerpts they recognized and enjoyed the original excerpt very much, and then could not enjoy the resulting mashup as it disrupted a song they did not want altered. In terms of potential negative aspects of the mashups, some (but not all) participants stated they did not enjoy overlapping vocal parts, while others commented on bass parts being incompatible.

To examine the results of the listening test we now inspect the mean ratings per excerpt, as displayed in Fig. 9. In all but one of the cases (Excerpt 2), the top ranked mashup is rated higher than the middle and lowest ranked mashups. The overall mean enjoyment ratings across all excerpts for the top, medium

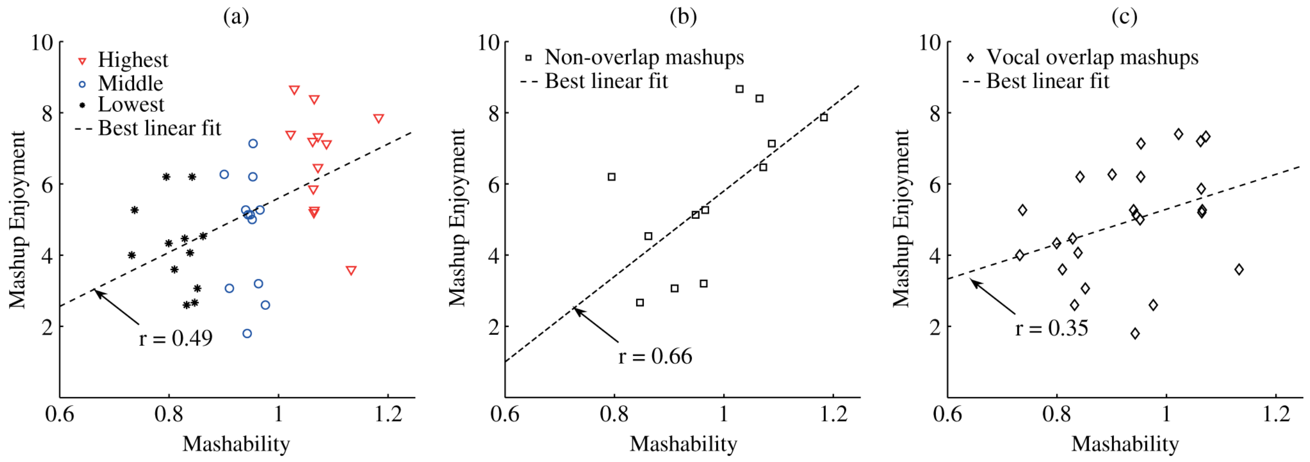


Fig. 10. Scatter plots of estimated mashability vs mean user enjoyment ratings for (a) all the three classes of mashup, (b) mashups without vocal overlap and (c) mashups with vocal overlap. The best linear fit is shown for each plot along with the Pearson correlation coefficient.

and lowest ranked mashups where 6.7, 4.7 and 4.3 respectively. Performing a paired t-test on the ratings revealed a highly significant difference between the top and middle ranked mashups ($p < .005$) and no significant difference between the middle and lowest ranked mashups ($p = .65$). This result endorses the overall effectiveness of the ranking according to our model of mashability. However, we should also note that even if the relative order of preference among the participants in the order is as we expected to be, not all top-ranked mashups were rated high in terms of enjoyment. For example, the top ranked mashup for Excerpt 1 was rated 3.6 despite having high mashability. Listening back to the mashup we can identify prominent overlapping vocals between the two excerpts in addition to an "out of phase" feeling caused by the unexpected behavior of chord changes not occurring on downbeats in the candidate song.

In terms of the more general relationship between mashability and user enjoyment, we can look at the scatter plot in Fig. 10(a). A Pearson correlation coefficient of $r = 0.49$, ($p < .005$), indicates a statistically significant correlation between mashability and user enjoyment on this dataset. It is important to note that some mashups were rated high in terms of enjoyment even though they were ranked low according to mashability. In the overall context of AutoMashUpper this means lower ranked results may still be acceptable, and given the subjective nature of mashup enjoyment, this is a predictable outcome. Perhaps what is more critical in terms of the success of our system is that the majority of top ranked results are not deemed unenjoyable by listeners. In light of these results we cannot, and nor do we seek to, claim that the top ranked song will definitely be the best according to user preference, e.g., compared to other songs with high mashability, merely that high mashability appears to be a good indicator of an acceptable result.

However, given the comment by some participants of vocal overlap being detrimental to mashup enjoyment, we now look in more detail at the results of the listening test. In Figs. 10(b) and (c) we can observe the relationship between mashability and enjoyment ratings for mashups manually separated to have no vocal overlap and those with vocal overlap respectively. Inspection of the plots immediately reveals a much stronger correlation for non-vocal overlap mashups compared to those with vocal overlap. While this could be taken as an indication that vocal overlap should be avoided in mashups, we can see that in Fig. 10(c) that several vocal overlap mashups were rated high in terms of enjoyment. On this basis we believe that attempting to remove all vocal overlap could eliminate potentially enjoyable mashups. From a user perpsective, we can hypothesize that two listener groups may exist, those who dislike vocal overlap and those for whom it is not a problem. In this way, such a preference could be indicated in the user interface.

VI. Discussion

Through our evaluation we have shown that the phrase-segmentation is able to reliably identify structural segmentation boundaries with precise temporal localization and to a degree which outperforms current state of the art methods. In terms of the subjective aspect of the evaluation, we have also shown that within these phrase sections, higher mashability is generally related to higher enjoyment by listeners of the end result. While we recognize that this constitutes an evaluation of AutoMashUpper at one level *below* the final result–its ability to produce automatic multi-song mashups, we believe that targeting the fully automatic result would overlook the important aspect of user-interaction.

Not only is it possible for users to interact and experiment with the mashup creation process by specifying mashability parameters, changing candidate songs and modifying the results, we allow this functionality in the user interface to explicitly encourage them to do so. We believe this interaction is particularly important for users to feel that they have contributed in making the musical result; that they are active participants rather than passive observers in the music content creation process. In future work, we intend to undertake a user study to evaluate these interactive aspects of AutoMashUpper, where issues such as ease of use and subjective enjoyment of complete music mashups can be addressed.

We consider AutoMashUpper to be a piece of "assistive" technology, designed to help users to create new musical content, especially when they might lack musical composition skills themselves. As noted by Shiga [2] a key musical skill required in mashup creation is in music *listening* rather than a direct ability in composition. Therefore, we can consider the role of AutoMashUpper to be one of providing a user with a set of good matches according to our objective mashability model, and then to allow the user to make their own subjective choices and refinements. In this way we hope to determine a useful balance between the objective aspects (e.g., similar harmonic and rhythmic structure) and the subjective elements (user preference and song familiarity) in mashup creation.

An important aspect in appraising the value of AutoMashUpper is to be aware of its limitations and to recognize when and why "mistakes" happen as well as rating their relative importance. Given the large number of different elements used in AutoMashUpper, one might assume that in order to obtain an objectively pleasing musical result it would be necessary for all elements to be correct, e.g., beat times, downbeats, phrase boundaries, tuning etc. While this is certainly desirable, we have found that some elements of the analysis are more critical to success than others. For example, if the beat times are incorrectly estimated, this can create an unpleasant feeling when the resulting mashup is not fully beat-synchronized, and is related to the user discomfort analysis for mismatched tempi in [8]. Furthermore, the estimation of tuning, which is central both to the generation of the harmonic signal representations (semitone-spectrogram and chromagram) and the scaling factor for pitch-shifting and tuning correction, plays a critical role in the resulting mashup. That is, even with all other aspects evaluated accurately, an "out-of-tune" mashup will generally not make for a pleasant listening experience. Other aspects however, such as mis-estimation of downbeats or phrase section boundaries may not be as critical since mashability is calculated across all beat shifts of candidate songs meaning such errors can be absorbed.

In terms of song properties which can adversely affect the resulting mashup, we have found these often relate to sung sections in the music. While setting a wide 'key shift' range in the AutoMashUpper interface increases the amount of possible solutions and therefore raises the chance of finding a good objective match, this can be problematic for vocal regions. This is perhaps most notable for increases in pitch for female vocals and decreases in pitch for male vocals, where the results can sound unnatural. In this sense, there is a greater chance of a natural musical result when using instrumental music–at least for those songs transformed in the mashup process. Similarly, the issue of vocal overlap was raised by some participants as a factor which created confusing results. A possible solution would be to incorporate a vocal activity detection module [38] into the estimation of mashability and simply discard any potential mashups with vocal overlap. However, given the diverse range of ratings for vocal overlap mashups obtained in the listening experiment (including some rated very highly), such a measure could exclude potentially valid and enjoyable mashups. Therefore we believe that a more complex model of vocal activity and interaction is required to properly capture this aspect. Since the chromagram is not as well-suited a signal representation for identifying vocal regions as MFCCs or melodic based features, we should explore additional features for inclusion in a more complete measure of mashability.

VII. Conclusions

In this paper we have presented a system for creating automatic and semi-automatic multi-song mashups. Our work has been driven by two main motivations: first, from a creative perspective, we have sought to demonstrate there are numerous musical possibilities and potential advantages to using MIR techniques for mashup creation; and second, from a research perspective, to highlight the significant potential mashups and music manipulation hold for future creative-MIR research. Our main contribution has been to show that by searching for mashability in a transform-domain which takes potential changes in key and rhythm into account, we can expand the range of mashups beyond what is possible with existing approaches. In terms of the next steps for the AutoMashUpper system, we plan to expand its use for songs which do not fit the strict assumptions concerning song properties, e.g. constant tempo and fixed time-signature. Furthermore we believe there is wide scope for incorporating other music analysis techniques, including vocal detection and source separation [39]. In future work we will conduct a user study to further explore the mashup creation possibilities with AutoMashUpper, and to identify new features and functionality to expand the creative possibilities for automatic mashup creation.

References

[1] X. Serra, M. Magas, E. Benetos, M. Chudy, S. Dixon, A. Flexer, E. Gómez, F. Gouyon, P. Herrera, S. Jordà, O. Paytuvi, G. Peeters, J. Schlüter, H. Vinet, and G. Widmer, Roadmap for Music Information ReSearch Creative Commons BY-NC-ND 3.0 license, 2013.

[2] J. Shiga, "Copy-and-persist: The logic of mash-up culture," *Critical Studies in Media Communication*, vol. 24, no. 2, pp. 93–114, 2007.

[3] Mashstix [Online]. Available: http://mashstix.com

[4] Mixter [Online]. Available: http://ccmixter.org

[5] DJ Mix Generator [Online]. Available: http://www.djprince.no/site/DMG.aspx

[6] N. Tokui, "Massh!: A web-based collective music mashup system," in *Proc. 3rd Int. Conf. Digital Interactive Media Entertainment and Arts*, 2008, pp. 526–527.

[7] G. Griffin, Y. E. Kim, and D. Turnbull, "Beat-sync-mash-coder: A web application for real-time creation of beat-synchronous music mashups," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process. (ICASSP)*, 2010, pp. 437–440.

[8] H. Ishizaki, K. Hoashi, and Y. Takishima, "Full-automatic DJ mixing with optimal tempo adjustment based on measurement function of user discomfort," in *Proc. 10th Int. Soc. Music Inf. Retrieval Conf.*, 2009, pp. 135–140.

[9] Fraunhofer, IDMT. Harmonic Mixing Tool [Online]. Available: http://www.idmt.fraunhofer.de/en/Service_Offerings/technologies/e_h/harmonic_mixing_tool.html

[10] Echonest. The Infinite Jukebox [Online]. Available: http://labs.echonest.com/Uploader/index.html

[11] Echonest. The Infinite Jukebox. The Autocanonizer [Online]. Available: http://static.echonest.com/autocanonizer/

[12] The Wub Machine [Online]. Available: http://thewubmachine.com

[13] Mixed in Key. Mashup. [Online]. Available: http://mashup.mixedinkey.com/HowTo

[14] M. E. P. Davies, P. Hamel, K. Yoshii, and M. Goto, "AutoMashUpper: An automatic multi-song mashup system," in *Proc. 14th Int. Soc. Music Inf. Retrieval Conf.*, 2013, pp. 575–580.

[15] J. Paulus, M. Müller, and A. Klapuri, "Audio-based music structure analysis," in *Proc. 11th Int. Soc. Music Inf. Retrieval Conf.*, 2010, pp. 625–636.

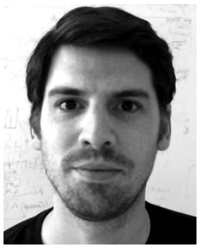
[16] D. P. W. Ellis, "Beat tracking by dynamic programming," *J. New Music Res.*, vol. 36, no. 1, pp. 51–60, 2007.

[17] M. Mauch and S. Dixon, "Approximate note transcription for the improved identification of difficult chords," in *Proc. 11th Int. Soc. Music Inf. Retrieval Conf.*, 2010, pp. 135–140.

[18] C. Cannam, M. O. Jewell, C. Rhodes, M. Sandler, and M. d'Inverno, "Linked data and you: Bringing music research software into the semantic web," *J. New Music Res.*, vol. 39, no. 4, pp. 313–325, 2010.

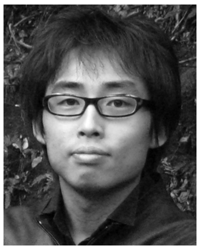
[19] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A tutorial on onset detection in music signals," *IEEE Trans. Speech Audio Process.*, vol. 13, no. 5, pp. 1035–1047, Sep. 2005.

[20] A. Robertson, A. Stark, and M. E. P. Davies, "Percussive beat tracking using real-time median filtering," in *Proc. 6th Int. Workshop Mach. Learn. Music (MML 13)*, 2013, pp. 71–75.
[21] D. Fitzgerald, "Harmonic/Percussive separation using median filtering," in *Proc. 13th Int. Conf. Digital Audio Effects (DAFx-10)*, 2010, pp. 10–13.
[22] M. E. P. Davies, M. D. Plumbley, and D. Eck, "Towards a musical beat emphasis function," in *Proc. IEEE Workshop Applicat. Signal Process. Audio Acoust. (WASPAA '09)*, 2009, pp. 61–64.
[23] M. E. P. Davies and M. D. Plumbley, "A spectral difference approach to extracting downbeats in musical audio," in *Proc. 14th Eur. Signal Process. Conf. (EUSIPCO)*, 2006.
[24] M. Goto, "An audio-based real-time beat tracking system for music with or without drum-sounds," *J. New Music Res.*, vol. 30, no. 2, pp. 159–171, 2001.
[25] J. Foote, "Automatic audio segmentation using a measure of audio novelty," in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2000, pp. 452–455.
[26] R. J. Weiss and J. P. Bello, "Identifying repeated patterns in music using sparse convolutive non-negative matrix factorization," in *Proc. 11th Int. Soc. Music Inf. Retrieval Conf.*, 2010, pp. 123–128.
[27] G. Sargent, F. Bimbot, and E. Vincent, "A regularity-constrained Viterbi algorithm and its application to the structural segmentation of songs," in *Proc. 12th Int. Soc. Music Inf. Retrieval Conf.*, 2011, pp. 483–488.
[28] M. J. Butler, *Unlocking The Groove: Rhythm, Meter, and Musical Design in Electronic Dance Music*. Bloomington, IN, USA: Indiana: Indiana Univ. Press, 2006.
[29] C. Uhle and C. Dittmar, "Drum pattern based genre classification of popular music," in *Proc. 25th AES Int. Conf.: Metadata for Audio*, 2004, pp. 189–195.
[30] D. Robinson, "Perceptual model for assessment of coded audio," Ph.D. dissertation, Dept. of Electron. Syst. Eng. , Univ. of Essex, Essex, U.K., 2002 [Online]. Available: www.mp3-tech.org/programmer/docs/Robinson_thesis.pdf
[31] C. Cannam, Rubber band library [Online]. Available: http://breakfastquay.com/rubberband/
[32] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka, "RWC music database: Popular, classical, and jazz music databases," in *Proc. 3rd Int. Conf. Music Inf. Retrieval*, 2002, pp. 287–288.
[33] M. Goto, "AIST annotation for the RWC music database," in *Proc. 7th Int. Conf. Music Inf. Retrieval*, 2006, pp. 359–360.
[34] H. M. Lukashevich, "Towards quantitative measures of evaluating song segmentation," in *Proc. 9th Int. Conf. Music Inf. Retrieval*, 2008, pp. 375–380.
[35] F. Kaiser, T. Sikora, and G. Peeters, "Large-scale cover song recognition using the 2D fourier transform magnitude," in *Music Inf. Retrieval Eval. eXchange (MIREX)*, 2012 [Online]. Available: http://www.music-ir.org/mirex/abstracts/2012/KSP3.pdf
[36] J. Serrà, M. Müller, P. Grosche, and J. L. Arcos, "The importance of detection boundaries in music structure annotation," in *Music Inf. Retrieval Eval. eXchange (MIREX)*, 2012 [Online]. Available: http://www.music-ir.org/mirex/abstracts/2012/SMGA2.pdf
[37] G. Sargent, F. Bimbot, and E. Vincent, "A music structure inference algorithms based on morphological analysis," in *Music Inf. Retrieval Evaluation eXchange (MIREX)*, 2012 [Online]. Available: http://www.music-ir.org/mirex/abstracts/2012/SBV1.pdf
[38] H. Fujihara, M. Goto, J. Ogata, and H. G. Okuno, "LyricSynchronizer: Automatic synchronization system between musical audio signals and lyrics," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 6, pp. 1252–1261, Dec. 2011.
[39] A. Ozerov, E. Vincent, and F. Bimbot, "A general flexible framework for the handling of prior information in audio source separation," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 4, pp. 1118–1133, May 2012.



Matthew E. P. Davies received the B.Eng. degree in computer systems with electronics from King's College London, U.K., in 2001 and the Ph.D. degree in electronic engineering from Queen Mary University of London, U.K., in 2007. From 2007 until 2011, he was a post-doctoral researcher in the Centre for Digital Music, QMUL. In 2011 he joined the Sound and Music Computing Group at INESC TEC in Porto, Portugal. In 2013, he worked in the Media Interaction Group, National Institute of Advanced Industrial Science and Technology (AIST), Japan, after which he returned to INESC TEC. His research interests include rhythm analysis in musical audio, evaluation methodology in MIR, and creative-MIR applications.

Philippe Hamel obtained his Ph.D. in computer science from the Université de Montréal in 2012. He then continued his research at AIST in Japan as a post-doctoral researcher. Since 2013, he has been a software engineer at Google working on improving music recommendation for Google Play Music. Prior to his work in computer science, he studied physics at Université de Montréal where he obtained a M.Sc. in theoretical physics and a B.Sc. in mathematics and physics. His main research interests are machine learning and its application to music information retrieval. One of his goals is to find ways to obtain better and richer representations of music audio. His recent work has been focused on artificial neural networks, deep learning, signal processing and collaborative filtering. He is interested in music information retrieval problems such as automatic tagging, music classification, music similarity and music recommendation.



Kazuyoshi Yoshii received the Ph.D. degree in informatics in 2008 from Kyoto University, Japan. He is currently a Senior Lecturer at Kyoto University. He has received several awards including the IPSJ Yamashita SIG Research Award and the Best-in-Class Award of MIREX 2005. His research interests include music signal processing and machine learning. He is a member of the Information Processing Society of Japan (IPSJ) and Institute of Electronics, Information and Communication Engineers (IEICE).

Masataka Goto received the Doctor of Engineering degree from Waseda University in 1998. He is currently a Prime Senior Researcher and the Leader of the Media Interaction Group at the National Institute of Advanced Industrial Science and Technology (AIST), Japan. Over the past 22 years, Masataka Goto has published more than 200 papers in refereed journals and international conferences and has received 38 awards, including several best paper awards, best presentation awards, the Tenth Japan Academy Medal, the Tenth JSPS PRIZE, and the Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology (Young Scientists' Prize). He has served as a committee member of over 90 scientific societies and conferences, including the General Chair of the 10th and 15th International Society for Music Information Retrieval Conferences (ISMIR 2009 and 2014). In 2011, as the Research Director he began a 5-year research project (OngaCREST Project) on music technologies, a project funded by the Japan Science and Technology Agency (CREST, JST).