

Automating Music Production with Music Information Retrieval

A thesis submitted for partial fulfillment of requirements
for graduation with research distinction in the
Department of Computer Science and Engineering
of The Ohio State University

By

Jordan V. Hawkins

The Ohio State University

March 2013

Abstract

Prior research in the field of Music Information Retrieval has yielded techniques for extracting musical information from digital audio, and made it possible to analyze human music production computationally. I hypothesize that a computer can be programmed to produce output similar to that of a musical artist on two production tasks performed by disk jockeys. The first, “mixing,” aims to create seamless transitions between songs in a playlist. The second, “mashup” creation, aims to overlay multiple similar tracks to create a new combined song. To automate these tasks, I first created example sets of my own mixes and mashups, and looked for patterns and relationships in the audio analysis data from the public music analysis API provided by The Echo Nest, Inc. I used my findings to write Python scripts that automatically perform the mixing or mashing tasks on any input audio files. The software was judged by a sample of individuals for its ability to produce output similar to that of a human DJ. Preliminary results support the claim that automatic music production processes can be convincing, but also show how programs perform poorly when processing unexpected input, suggesting that tasks are most easily replicated within specific, predefined artistic styles.

Dedication

This document is dedicated to generic pop music producers,
and the oversimplified music that I have completely overanalyzed.

Acknowledgements

Multiple individuals contributed their time and ideas to help facilitate this project. I am grateful for the selflessness with which they set aside their personal priorities to help me pursue this passion. The following people played an integral part in my progress over the past year, and deserve to be recognized as genuine advocates for undergraduate research.

I want to first thank Liz Riter for assisting me throughout my involvement with undergraduate research. My experience would not have been possible without her eagerness to answer questions and connect her students to the right faculty. Dr. Bradley Clymer was also essential to getting this project started by connecting me with Dr. Raphael Wenger to enroll in the proper CSE undergraduate research coursework. Dr. Clymer gave additional support again by volunteering to serve on the oral defense committee of this thesis. I would also like to thank Dr. Phillip Schniter for agreeing to advise this project, and for also serving on my oral defense committee.

I am lucky to have had many of my peers provide feedback or guidance during the development and testing of my software programs. I would like to recognize DJ Mike “Dansky” Smidansky for his advice programming the DJ music production tasks identified in this study. The eventual test phase of the software would not have been possible without the twenty anonymous human test subjects that helped rate the performance of the programs.

Finally, I would like to thank Dr. Michael Mandel for co-advising this research project. Dr. Mandel’s prior experience with Music Information Retrieval was invaluable when defining project objectives, as was his guidance while taking steps to complete

them. Without the help of exceptional advisors and helpful friends, this project would not have been possible.

Table of Contents

Abstract	2
Dedication	3
Acknowledgements	4
List of Tables	7
List of Figures	8
Chapter 1: Introduction	9
1.1 Motivation and Personal Background	9
1.2 Past Research in Music Information Retrieval.....	10
1.3 Project Objectives.....	11
1.4 Thesis Objectives	13
Chapter 2: Literature Review	14
2.1 Glossary of Musical Terms.....	14
2.2 Echo Nest Data Representation	15
2.3 Image Processing with Scikit-Image.....	16
Chapter 3: Automatic DJ	19
3.1 Development	19
3.2 Testing.....	27
3.2.1 Statistical Results.....	30
3.2.2 General Listener Feedback	35
3.3 Enhancements	37
Chapter 4: Automatic MashUp	39
4.1 Development	39
4.2 Testing.....	47
4.2.1 Statistical Results.....	49
4.2.2 General Listener Feedback	57
4.3 Enhancements	57
Chapter 5: Conclusion	60
References	62
Appendix A: Automatic DJ Python Script	63
Appendix B: Automatic DJ Flowchart	67
Appendix C: Automatic DJ Training, Validation Set Graphs	68
Appendix D: Human Listening Test Questionnaire	83
Appendix E: Automatic DJ Listening Test Results	85
Appendix F: Automatic MashUp Python Script	93
Appendix G: Automatic MashUp Flowchart	100
Appendix H: Automatic MashUp Listening Test Results	102

List of Tables

Table 1: Training Set of manually selected demarcations	20
Table 2: Automatic DJ test input	28
Table 3: Audio clips of transitions used for human testing	29
Table 4: Average test results for Automatic DJ.....	29
Table 5: Automatic DJ ANOVA tests	31
Table 6: Automatic DJ ANOVA key conclusions	35
Table 7: Groups of component tracks used for mashup creation	41
Table 8: Test cases with fully verified beat and key information.....	47
Table 9: Mashup audio clips used in human testing.....	48
Table 10: Average test results for Automatic MashUp	48
Table 11: Automatic MashUp ANOVA key conclusions	56

List of Figures

Figure 1: The similarity matrix for Jimmy Buffet's Margaritaville.	17
Figure 2: Coin template	18
Figure 3: Larger image of many different coins	18
Figure 4: Match_template result	18
Figure 5: Loudness data for <i>When Love Takes Over</i> by David Guetta	21
Figure 6: Manual and automatic demarcations based on loudness data	23
Figure 7: Loudness data for <i>LE7ELS</i> by Avicii	25
Figure 8: Loudness data for <i>Don't Stop Til' You Get Enough</i> by Michael Jackson ..	26
Figure 10: Chroma, timbre, and loudness per beat	43
Figure 11: High-level visual depiction of template matching.	44
Figure 12: Four analysis section templates matched with another song	45
Figure 13: Automatic MashUp output audio clip.	46
Figure 14: Automatic MashUp ANOVA tests.	51

Chapter 1: Introduction

1.1 Motivation and Personal Background

Historically, large-scale analysis required a music researcher to interpret large amounts of sheet music, or listen to days of recorded songs. The task of looking for relationships or patterns in large libraries of music was inevitably arduous and time consuming. Because digital audio is encoded into a binary abstraction, software can process music computationally to deeply analyze large amounts of data more easily. As computers get faster and digital audio libraries become more extensive, researchers have an even greater opportunity to take a new approach to how music information is gathered.

Music Information Retrieval (MIR) is a field that combines signal processing and musicology to quantify digital audio features. My research investigates how these metrics can be used to look for patterns in music, identify common procedures in music production, and then automate the procedures to create original music. I hypothesize that unwritten algorithms already exist in music production, and that the creative process can be better understood if its procedures are identified in the form of software. There are millions of musicians worldwide, and this work may not only shed light on how music is already being produced, but also inspire a new approach to music production entirely.

My pursuit of this study stems from an interest in music composition that far precedes my knowledge of Music Information Retrieval. My experience in composing and recording original works fostered my understanding of music theory and inspired me to take a novel look at music production. My undergraduate coursework in computer

engineering laid the foundation to view digital audio as an enhanced musical format that can harness computational power to catalyze and improve musicological analysis.

1.2 Past Research in Music Information Retrieval

Prior MIR research has already advanced considerably toward the goal of quantifying specific audio features, some of which are described in the next chapter in greater detail. The Music Information Retrieval Evaluation Exchange (MIREX)¹ is an annual event organized to test the effectiveness of new MIR algorithms, which also motivates research to improve the accuracy of existing algorithms. The Echo Nest, Inc. implements this work to make data metrics available to software developers through its public music analysis API². Companies like Spotify³ and iHeartRadio⁴ have partnered with The Echo Nest to improve their music services by gaining greater understanding of their users' listening tastes. Some of the existing successful applications of MIR classify songs based on their metadata, such as artist or genre, while others rely on deeper analysis of intrinsic song content.

There are examples of MIR research that leverage quantified audio features to solve problems having to do with song similarity. Ellis and Poliner (2007) propose a method for using musical data metrics to identify different artist renditions of the same song. Tsai et al. (2005) present another methodology for querying music databases for songs with a specific melody. Bertin-Mahieux et al. (2010) investigate the development

¹ http://www.music-ir.org/mirex/wiki/MIREX_HOME

² <https://github.com/echonest/remix>

³ <https://www.spotify.com/us/for-music/>

⁴ <http://www.iheart.com/>

of a “codebook” of common song features to be used to identify similar features in large collections of music. These research efforts yielded helpful ideas when solving problems that involved the analysis and comparison of musical features.

Example Python modules exist that demonstrate the capabilities of The Echo Nest’s public music analysis API. One project⁵ uses analysis data to pull ten-second audio clips from multiple songs and concatenate them together into one track. Another program⁶ uses any song’s data metrics to find patterns and similarities within its structure, and then rearrange the structure into an extended version of the song. Existing software projects like these⁷ share useful methods for leveraging The Echo Nest’s development features to complete project objectives like those defined for this project.

1.3 Project Objectives

The objectives of this project are to investigate another application of MIR research that uses The Echo Nest’s metrics and machine learning techniques to automate two specific music production tasks currently performed by human disk jockeys. The first, “mixing,” aims to create seamless transitions between songs in a playlist. The second, “mashup” creation, aims to overlay multiple similar tracks to create a euphonious combined song. Each application is written as a separate software program in Python, and uses The Echo Nest’s Analysis API to obtain audio analysis data, and their Remix API to modify or edit the audio.

⁵ <https://github.com/echonest/remix/tree/master/examples/capsule>

⁶ <https://github.com/echonest/remix/tree/master/examples/earworm>

⁷ <https://github.com/echonest/remix/tree/master/examples>

The first program is called Automatic DJ, and is designed to imitate the performance of a live DJ. Existing software programs like MIXXX⁸ and Ultramixer 4⁹ include features that automatically transition between songs, but besides automatically fading volumes between songs, they do not imitate the mixing work of a real DJ. My program analyzes the musical content of every song in a playlist, reorders and modifies the audio files, and transitions between them with a beat-matched crossfade.

The second program, Automatic MashUp, uses song features of two input tracks to find compatible song sections to overlay and combine into a new mashup. No other software program like this has been developed successfully, and although the software from this project produces some capable output, I show how inaccurate analysis data has a large impact on the development of a successful algorithm.

Some additional software applications provide support as DJ tools during this project. Audacity¹⁰ is a free application that was used to shift the tempo and pitch for audio files. Apple's GarageBand¹¹ was purchased from Apple's App Store to edit audio files further, and to mix together new audio files.

⁸ <http://www.mixxx.org/>

⁹ <http://www.ultramixer.com/>

¹⁰ <http://audacity.sourceforge.net/copyright>

¹¹ <http://www.apple.com/ilife/garageband/>

1.4 Thesis Objectives

This thesis uses digital audio analysis data to explore its application in software programs that imitate music production tasks. I demonstrate how MIR techniques as implemented by The Echo Nest can be used to automate music creation, and reveal instances where the software's efficacy is ultimately limited by the occasional unreliability of analysis data. This thesis also explores the feasibility of automating music production, and identifies challenges associated with programming music production processes. The document closes with recommendations for future work based on subjective human listening tests and my own conclusions from this project.

Chapter 2: Literature Review

2.1 Glossary of Musical Terms

Music theory concepts and vocabulary from Randel (2003) are referenced throughout the text. The following is a brief review of some definitions and concepts. The next section in this chapter explains how The Echo Nest quantifies these items as concrete analysis data¹².

- **Beat:** A beat is a rhythmic unit of time. In a song with a constant tempo, all of the beats are the same duration.
- **Bar:** A bar is a sequence of beats that begin with an emphasized beat. This document only refers to bars with a length of four beats.
- **Chroma:** The chroma, or chromatic content, is the strength of the twelve different pitches {C, C#, D, Eb, E, F, F#, G, Ab, A, Bb, B}.
- **Chord Progression:** A chord progression is a repeating pattern of changing chromatic content prescribed to a specific sequence of bars.
- **Timbre:** Timbre is the quality that differentiates sounds with the same pitch and loudness.
- **Rhythm:** Rhythm is characterized by timing patterns in a musical piece, and is described in this document using timbre information.
- **Dynamics:** Dynamics is the term used to describe how the loudness of a song changes. It's commonly positively correlated with a song's intensity or energy.
- **Key:** A song usually only uses a subset of the twelve notes in the chromatic spectrum. A specific subset of pitches is defined as a Key.
- **Tempo:** The Tempo is the speed of the song in beats-per-minute (BPM).

¹² <http://echonest.github.com/remix/apidocs/>

- Section: A song section is a subset of beats that share some kind of commonality in their attributes. The musical works explored in this paper are commonly composed of three types of sections:
 - Verse: A verse is a section of a song typically characterized by softer dynamics, lower energy, and more thoughtful lyrics.
 - Chorus: Also known as a refrain, a chorus is a song section that is typically characterized by louder dynamics, higher energy, and repeating, catchy lyrics.
 - Bridge: The bridge is a section of the song that usually follows a chorus, and is similar to the chorus in that it is typically characterized by louder dynamics and higher energy. The bridge is usually just a variation of the chorus.
- Song Structure: A song’s structure is the holistic representation of all of its sections.

2.2 Echo Nest Data Representation

The Echo Nest quantifies all of the terms in their Analysis API. The following analysis terms connect these concrete representations with their abstract meanings.

- Analysis data: This term is used later in the paper, and refers to the analysis data returned by The Echo Nest’s Analysis API to describe abstract song attributes like chroma, timbre, loudness, and more.
- Segments: Segments are short audio clips of musical “events” or notes, for each of which The Echo Nest uses to calculate analysis data. If a larger time quantifier, like beats or bars, is referenced as having analysis data, the analysis data is actually an average of the segments that comprise the beat or bar.
- Confidence: Each analysis metric is accompanied by a confidence value to express the confidence with which The Echo Nest estimates that specific musicological quantity. In my experience, however, confidence values do not accurately indicate the correctness of a value.

- **Sections:** The Echo Nest calculates and provides a list of demarcated sections for a song. The specification for how these sections are calculated is not clearly provided, however, so I do not use this feature in the Automatic DJ development phase when deeper analysis is necessary. The abstract definition and this concrete list definition are both used in the paper, so it is important to understand the difference between the definitions.
- **Beats, Bars, and Sections Lists:** All three of these time markers are represented as sequences of tuples. Each tuple contains a start time, a duration time, and a confidence value.
- **Local Context:** The local context of a beat is its index within its respective bar. For example, the local context of beat 1 is 0, and the local context of beat 4 is 3. The local context values for a sequence of four bars is $\{\{0,1,2,3\}, \{0,1,2,3\}, \{0,1,2,3\}, \{0,1,2,3\}\}$.

2.3 Image Processing with Scikit-Image

Chapter 4 briefly presents song attributes in the format of two-dimensional images. This section cites a previous application of image processing to a song-similarity problem, and provides additional information about the Scikit-Image python module's template matching procedure¹³ is used to find similarity between songs.

The image processing methodology used in this document is based on a technique used by Bartsche & Wakefield (2001) to locate the refrains in an audio recording. A two-dimensional similarity matrix of the song recording with itself reveals similar song sections as dark diagonal lines in the image. The figure below shows an example of such a similarity matrix.

¹³ http://scikit-image.org/docs/dev/auto_examples/plot_template.html

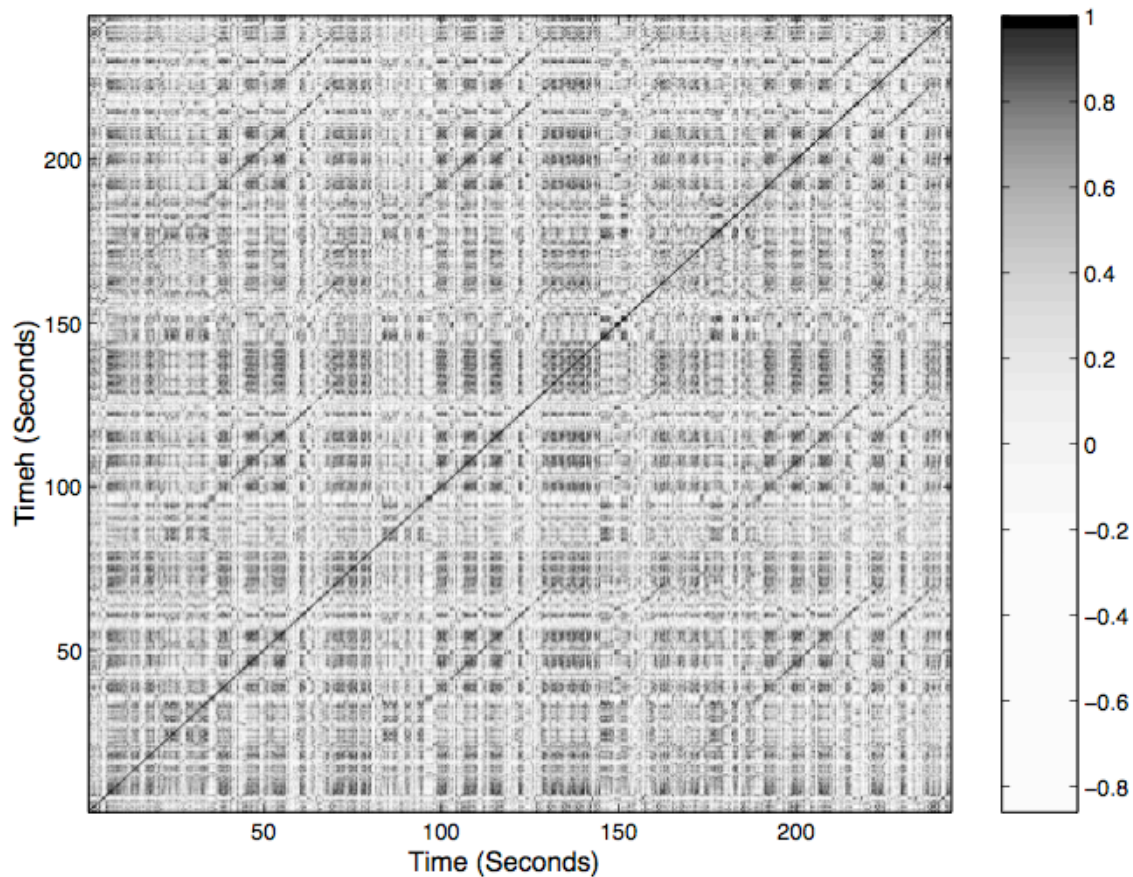


Figure 1: The similarity matrix for Jimmy Buffet's Margaritaville.

The dark diagonal line specified on the image shows where the chromatic similarity between regions of approximately 30 to 70 seconds and 100 to 140 seconds is strong enough to indicate that a refrain occurs during those times. The template matching procedure used in this project also calculates similarity in a two-dimensional fashion, but differs in that it first breaks song sections into smaller sized “template” images before calculating similarity between two recordings.

Scikit-Image provides a `match_template` function based on a method by Lewis (1995) that calculates normalized cross-correlation to find instances of a smaller, “template” image within a larger image. In these example photos from Sci-kit Image

(2013), the location of a single coin is shown within a larger image of many coins as the result of such a template matching procedure. The pixel with the highest correlation value, designated by the red circle in the picture, specifies the smaller coins location in the larger image.



Figure 2: Coin template

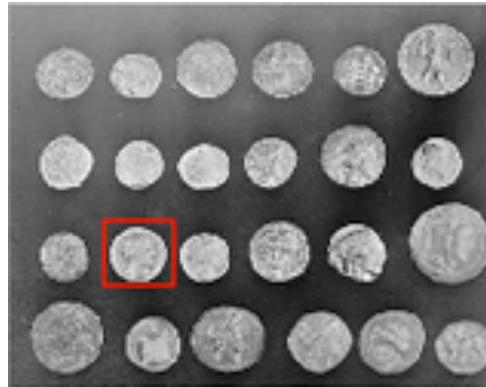


Figure 3: Larger image of many different coins

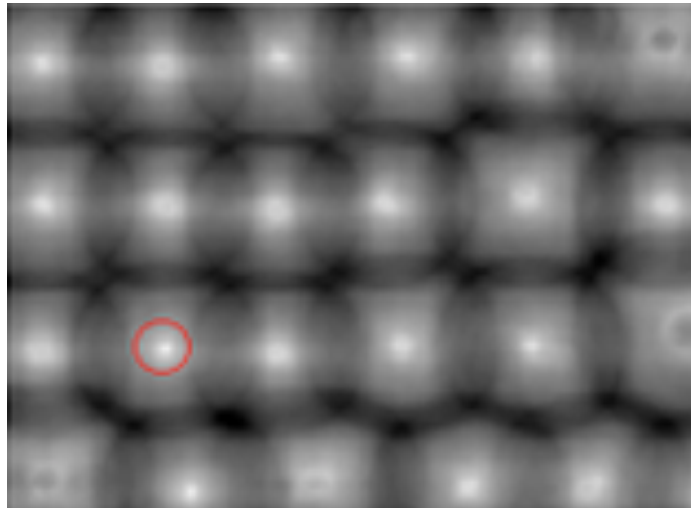


Figure 4: Match_template result

This method is referenced again in the text when finding similarity between song sections and their attributes.

Chapter 3: Automatic DJ

3.1 Development

In this section, program specifications for Automatic DJ are defined, and then certain design decisions are explained. The complete Python script is included in Appendix A, and a visual workflow the program's operation is included in Appendix B.

The specifications for this program are first determined by defining basic characteristics common to disk jockey performances. A typical DJ playlist promotes a certain mood or atmosphere, such as that of a high-energy dance venue. The individual tracks are also deliberately arranged and mixed together to create seamless transitions that make the playlist a unified whole. Four basic considerations outlined by Smidansky (2012) most commonly determine the arrangement:

- Tempo – Songs of similar tempo are grouped together in order to preserve a steady pulse when transitioning between tracks.
- Beat Location/Beat Matching – Overlapping incoming and outgoing tracks must have their beats aligned to preserve rhythmic similarity between tracks.
- Local Context – Songs transitions match the local context of the outgoing and incoming tracks to preserve the number of beats in a bar. For example, the last four beats of the outgoing song must match the first four beats of the incoming song, 1-to-1, 2-to-2, 3-to-3, 4-to-4.
- Song Structure – Song transitions typically happen at the end or beginning of sections to preserve one song's structural identity into the next.

The Analysis API returns specific values for a song's tempo, so playlist items of similar speed can be easily grouped together after the program's initialization. Beat

location and local context attributes are also pre-calculated by the Analysis API, but specifying a song structure’s significance to the DJ mixing process is more a complicated task.

To determine how transition points are chosen in the context of song structure, I made a twenty song DJ mix of my own to train an algorithm that automatically chooses transition points. I selected contemporary tracks with strong, high-energy dance rhythms to further define the program’s input and mitigate the amount of variability between songs of different genre. I recorded the times where each individual song fades in and fades out of the mix in the following table. Each row is a separate audio file that partially comprises the entire mix, and the times recorded are specific to the audio file, not the duration of the entire twenty-song mix.

Song Name	Start Time	End Time
Spectrum	2:27.876	5:25.127
Somebody Told Me	0:21.150	3:13.356
Bangarang	0:26.516	2:11.312
When Love Takes Over	0:37.282	2:05.899
Believer	0:00.000	4:32.304
Feel So Close	0:37.785	3:15.357
I Remember	0:15.230	3:15.357
We Are The People	1:22.312	2:55.997
Call On Me	0:30.887	2:02.294
Pins	0:14.775	3:12.008
Downforce	3:33.437	4:58.800
Dreamcatcher	3:58.297	4:54.371
Hearts On Fire	0:15.370	0:46.003
Move Your Feet	0:04.746	2:52.450
Le7els	1:07.721	2:37.716
Til Death	0:37.539	2:52.498
Lights	0:41.044	3:21.012
What You Know	0:41.625	2:18.350
Cry	2:02.042	3:33.469
Sandstorm	0:29.553	1:26.022

Table 1: Training Set of manually selected demarcations

Using the data above, I graphed each song to look for relationships between its musical attributes and my transition location choices, and found that loudness data sufficiently indicates a relationship between song dynamics and my transition points. I low-pass filtered the loudness with a moving average to suppress any distracting outlier values and found the relationships shown in following graph. The x-axis is a sequence of the audio file's individual analysis segments, and the y-axis corresponds to each segment's maximum loudness attribute. The red vertical lines mark where I made my manual selections.

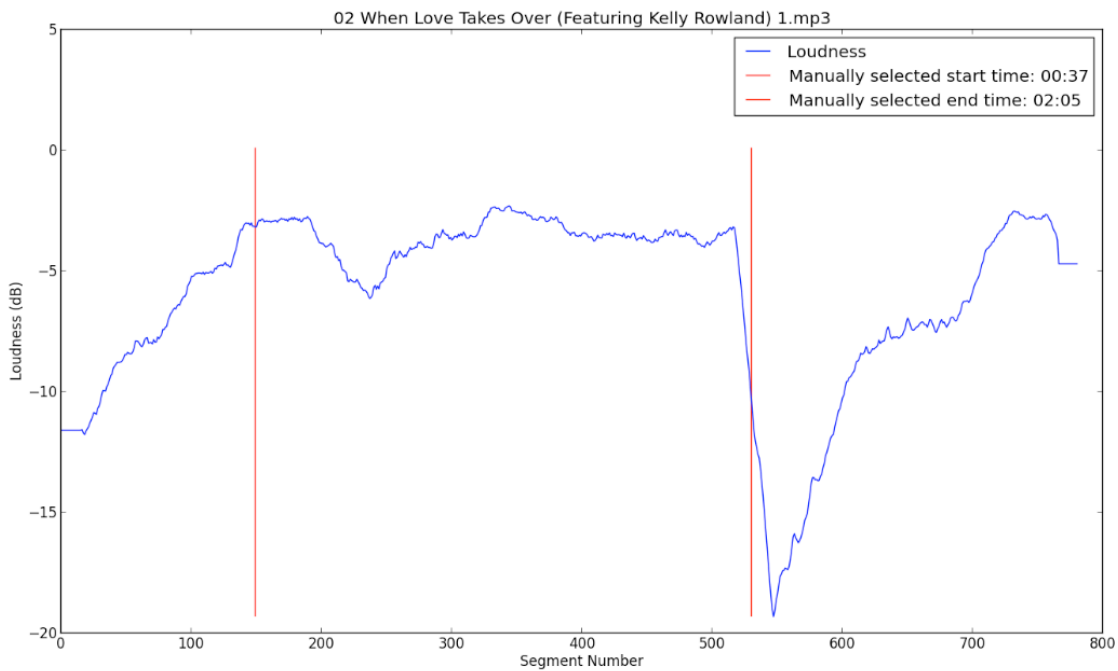


Figure 5: Loudness data for *When Love Takes Over* by David Guetta

The graphs show how most songs have a unique average loudness for different structural sections. A song chorus is typically louder than the verse, and looks like a plateau in the image above. I typically made my selection where there was a “beat-drop,” or the sudden

increase in dynamics before a chorus or other section of high energy. My selections were motivated by the goal of preserving a high-energy dance atmosphere, and this goal is indicated in the graphs as plateaus, or regions of consistent loudness.

The code responsible for choosing transition points analyzes a song's loudness data, and then demarcates it similar to the graphs above. I compared the computer-generated and human-generated demarcations graphically on the training songs, and changed the following four key system parameters to match the algorithm's output with my own selections as closely as possible.

- Window length – This determined how much smoothing was done to the loudness data. A small window length does less smoothing and makes consistently loud regions less apparent, but consequently also preserves possibly indicative changes in dynamics. A large window length does more smoothing and does the opposite, making song sections more distinct, but stifling possibly important sudden changes in dynamics. I set this value to a 16 beat long approximation of segments by default, as this length was the best balance the two conflicting interests.
- Loudness Ceiling – This metric defined when the loudness data was “loud enough” to be considered the beginning of a chorus. It is determined as a multiple of the mean loudness of the entire song (a negative number of decibels). I set this parameter to .8 by default, because songs typically rose to this loudness level where I made my demarcation selections.
- Loudness Floor – This metric defined when the loudness data was “quiet enough” to indicate that the consistently loud region of the song was over. It is also determined as a multiple of the mean loudness of the entire song. I set this value to 1.2 by default, because my manually chosen song regions were typically cut off once they fell below this amount of loudness.
- Minimum Duration – This parameter defined the minimum duration for the chosen region. A typical mix plays each song for at least one minute, so this parameter was assigned a value of 60.0 seconds.

If the algorithm is unable to find a region that fits these parameters, then the Loudness Ceiling and Loudness Floor values are slightly increased by .05 to include more song segments in the chosen regions until the Minimum Duration parameter is satisfied.

The graphs in Appendix C compare the program's selections with my own for every item in the training set. The picture below is one example, where my manual selections in red compare to the program's selections in green.



Figure 6: Manual and automatic demarcations based on loudness data

Next, I ran the choosing algorithm on a validation set of ten more songs and modified the system parameters as needed to increase the accuracy of the automatically chosen demarcations.

In addition to selecting the most consistently high-energy region of the song, the software includes functionality for automatically transitioning between the songs. The Remix API has helpful methods for analyzing a track's beat information, and then cross fading from one song to the next. I converted the chosen analysis segments to their corresponding sequence of bars within the song, to ensure that each song in the playlist starts and ends on beat 1 to satisfy the Local Context consideration from above. Finally, I transitioned between the beat lists using the Crossmatch method included in the `echonest.action` module. Crossmatch lengthens or shortens the ends of audio files of different tempo so that their overlapping beats are synchronized. I used Crossmatch to overlap the last four beats of the outgoing song with the first four beats of the incoming song, and simultaneously cross fade between the two songs.

After completing the entire system's code, I completed some functional testing on a test set of 20 new songs with similar characteristics to those selected for the training and validation sets. The testing showed that there were common attributes of songs for which the program performed well, and of songs for which the program performed poorly. As expected, high-energy dance songs with sections characterized by significant changes in average loudness performed best. The graph below is an example of such a song, and the "plateaus" in its graph represent the ideal case of when song sections clearly differ in their average loudness.

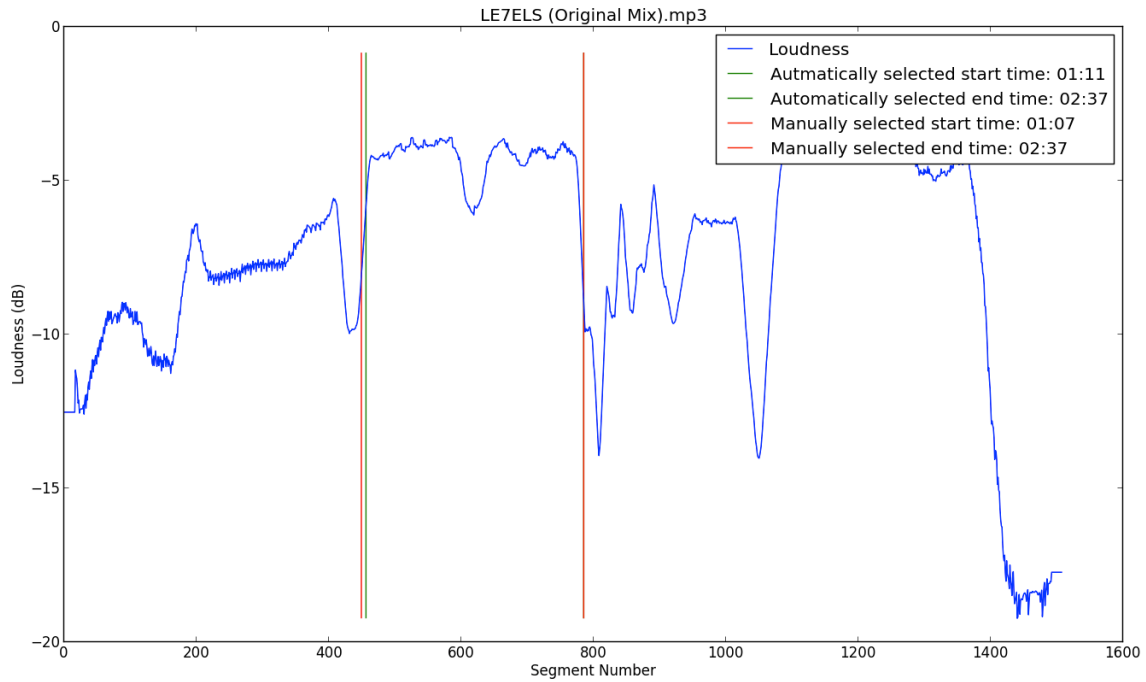


Figure 7: Loudness data for *LE7ELS* by Avicii

These are the kinds of songs that trained the section-choosing algorithm, so it makes sense that this genre of random input generated the best output. Alternatively, the algorithm performed less reliably on input songs with loudness values that varied more during song sections, or varied very little throughout the entire song. Loudness values for Michael Jackson’s “Don’t Stop ‘Til You Get Enough” do not reveal any distinct sections, so the choosing algorithm did not demarcate this song as accurately.

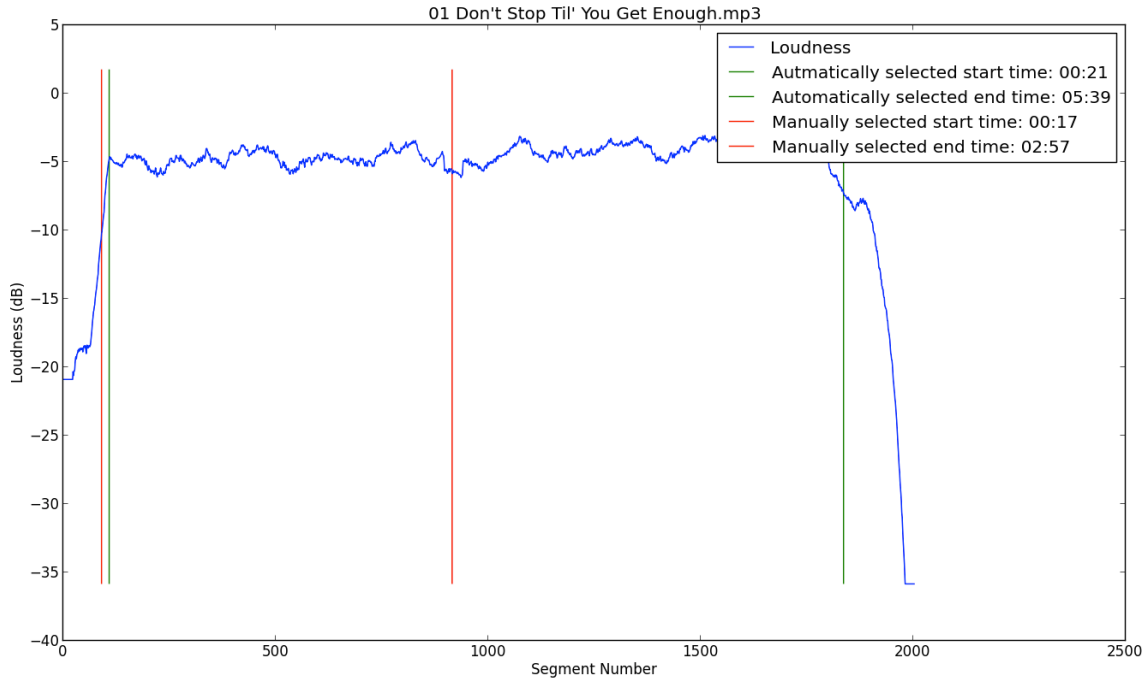


Figure 8: Loudness data for *Don't Stop Til' You Get Enough* by Michael Jackson

In addition to the choosing algorithm's imperfections, inaccuracy in The Echo Nest's analysis data was another occasional source of error. Loudness data was consistently accurate, but local context and beat detection information was less reliable, and resulted in less seamless transitions between songs. The code responsible for calling Crossmatch assumes that the two overlapping lists of four beats both have a local context of $\{1,2,3,4\}$. If the local context data returned by the Remix API is inaccurate, then two lists with local context $\{1,2,3,4\}$ and $\{4,1,2,3\}$ may be mixed together, and the listener hears an arrhythmic sequence such as $\{1,2,3,4,4,1,2,3,\dots\}$ instead of the correct sequence of $\{1,2,3,4,1,2,3,4,\dots\}$. A song's tempo estimation was almost always very accurate, but was sometimes incorrectly estimated at half-time or double-time. This issue sometimes resulted in awkward Crossmatches, because actually only 2 beats of one song would overlap with 4 of the other. This issue often resulted, however, in an interesting transition

between two songs with tempos that proved to still sound compatible. Using the results of the functional testing and the criteria for good input cases, I selected a playlist of songs for formal testing on human subjects.

3.2 Testing

Human listeners ultimately decide a DJ's performance, so this test procedure gathers feedback from individuals and quantifies the program's ability to produce output similar to that of a human DJ. Twenty volunteers listened to ten audio clips and completed the questionnaire in Appendix D. Each clip is approximately forty seconds in length, and consists of a transition from one song to the next. The questionnaire asks subjects to guess if individual transitions were generated automatically by a computer or manually by a human. Listeners also rated on a 1-10 scale the compatibility, or seamlessness, of each transition and how well the transitions were located within the song's structure.

To generate the ten audio clips, I used the ideal input characteristics defined after functional testing to choose the following eleven input songs.

Name	Artist
Payphone	Maroon 5 Feat. Wiz Khalifa
There She Goes	Taio Cruz
Greyhound	Swedish House Mafia
Euphoria	Loreen
30 Days	The Saturdays
Chasing The Sun	The Wanted
LaserLight	Jessie J Featuring David Guetta
Turn Up The Music	Chris Brown
Scream	Usher
Let's Go	Calvin Harris Feat. Ne-Yo
Feel The Love	Rudimental Feat. John Newman

Table 2: Automatic DJ test input

After running the program and generating the mixed output playlist, I manually created the ten audio clips to better feature the transitions between the eleven input songs. Each clip includes about ten to twenty seconds of audio before and after the transition point to give listeners context about where the transitions occurs within the two songs' structures. I then used a random number generator to pick four of the ten transitions, and replaced them with my own manually generated transitions. These test cases comprised the "Playlist 1" referenced on the survey sheet, and are included in the following table.

Name	Duration (secs)	Generated By...
Feel The Love to Payphone	47	Human
Payphone to Greyhound	50	Computer
Greyhound to Laserlight	46	Human
Laserlight to Scream	37	Computer
Scream to Let's Go	44	Human
Let's Go to There She Goes	45	Computer
There She Goes to Chasing The Sun	47	Human
Chasing the Sun to Turn Up The Music	48	Computer
Turn Up The Music to 30 Days	47	Computer
30 Days to Euphoria	44	Computer

Table 3: Audio clips of transitions used for human testing

The full results from the study are included in Appendix E, and briefly summarized in the table below.

Origin	Compatibility	Location	% Marked Human
Computer	5.92	6.03	34.16
Human	6.94	7.38	50
Total	6.33	6.57	40.5

Table 4: Average test results for Automatic DJ

The table shows average ratings for compatibility and location, and the percentage of responses that believed a transition type to be human. The twenty responses are broken down into two categories based on whether the transition type was truly human-generated, or actually computer-generated. The table shows that human-generated transitions were on average rated higher than computer-generated transitions in both the compatibility and location categories. Also presented above is that human-generated transitions were typically perceived to be human-generated more often than computer-generated transitions.

3.2.1 Statistical Results

The following ANOVA tests use a significance threshold of .05 to determine which variables impacted the listeners' responses. Each test answers the following questions for different combinations of Factor 1, Factor 2, and Result variables:

1. Does the independent variable Factor 1 have a significant impact on the dependent Response variable?
2. Does the independent variable Factor 2 have a significant impact on the dependent Response variable?
3. Is there an interaction between independent variables Factor 1 and Factor 2 that significantly impacts the dependent Response variable?

Each unique combination of factors is placed into its own test group, and each test group is comprised of three tests against three different response variables. The variables used in the table are defined as follows:

- Listener #: Listener number is the unique integer used to identify one of the twenty listeners.
- Song #: Song number is the unique integer used to identify one of the ten audio clips used in each of the listening tests.
- isHumanTruth: A Boolean value used to declare whether an audio clip is human-generated (isHumanTruth = 1) or computer-generated (isHumanTruth = 0).
- isHumanChoice: A Boolean value used to declare whether an audio clip was perceived to be human-generated (isHumanChoice = 1) or computer-generated (isHumanChoice = 0).

Group	Test	Factor 1	Factor 2	Response
A	1	Listener #	isHumanTruth	isHumanChoice
	2	Listener #	isHumanTruth	compatibility
	3	Listener #	isHumanTruth	location
	4	Listener #	isHumanChoice	isHumanTruth
B	5	Listener #	isHumanChoice	compatibility
	6	Listener #	isHumanChoice	location
	10	Song #	isHumanChoice	isHumanTruth
C	11	Song #	isHumanChoice	compatibility
	12	Song #	isHumanChoice	location

Table 5: Automatic DJ ANOVA tests

The hypotheses and results for each ground of tests are summarized below. For each test, the two factors and their interaction are each evaluated as sources of variance, and considered to have a significant impact on the response if the significance value (p) is less than the significance threshold of .05. Conclusions for each group summarize the software’s ability to produce human-quality output.

Group A

Hypotheses:

1. Listener number has no significant impact on listeners’ perceptions of which transitions are human or computer-generated, or how the listeners rate compatibility or location.
2. The factor of whether a transition was generated by a human or by a computer has no significant impact on listeners’ perceptions of which transitions were human or computer-generated, or how the listeners rate compatibility or location.
3. The interaction between listener number and whether a transition was generated by a human or by a computer has no significant impact on listeners’ perceptions

of which transitions were human or computer-generated, or how the listeners rate compatibility or location.

Significance Values (p):

Source of Variation	isHumanChoice	Compatibility	Location
Listener #	0.999	0.024	0.310
isHumanTruth	0.028	0.045	0.003
Listener #*isHumanTruth	0.113	0.099	0.154

Conclusions:

1. Listener number has no significant impact on the listeners' perceptions of which transitions were human or computer-generated, or how the listeners rate location. Listener number does, however, have a significant impact on how listeners rate compatibility.
2. The factor of whether a transition was generated by a human or by a computer has a significant impact on listeners' perceptions of which transitions were human or computer-generated, and how the listeners rate compatibility and location.
3. The interaction between listener number and whether a transition was generated by a human or by a computer has no significant impact on listeners' perceptions of which transitions were human or computer-generated, or how the listeners rate compatibility or location.

Group B

Hypotheses:

1. Listener number has no significant impact on whether transitions were human or computer-generated, or how the listeners rate compatibility or location.

2. Listeners' perceptions of whether a transition was generated by a human or by a computer have no significant impact on which transitions were human or computer-generated, but do have a significant impact on how the listeners rate compatibility and location.
3. The interaction between listener number and listeners' perceptions of whether a transition was generated by a human or by a computer has no significant impact on which transitions were human or computer-generated, or how the listeners rate compatibility or location.

Significance Values (p):

Source of Variation	isHumanTruth	Compatibility	Location
Listener #	1.000	0.009	0.338
isHumanChoice	0.030	0.000	0.002
Listener #*isHumanChoice	0.126	0.745	0.422

Conclusions:

1. Listener number has no significant impact on whether transitions were human or computer-generated, or how the listeners rate location. Listener number does, however, have a significant impact on how listeners rate compatibility.
2. Listeners' perceptions of whether a transition was generated by a human or by a computer have a significant impact on which transitions were human or computer-generated, and how the listeners rate compatibility and location.
3. The interaction between listener number and listeners' perceptions of whether a transition was generated by a human or by a computer has no significant impact on which transitions were human or computer-generated, or how the listeners rate compatibility or location.

Group C

Hypotheses:

1. Song number has no significant impact on how listeners rate compatibility or location, and is completely related to whether a transition was generated by a human or by a computer.
2. Listeners' perceptions of whether a transition was generated by a human or by a computer have no significant impact on which transitions were human or computer-generated, but do have a significant impact on how the listeners rate compatibility and location.
3. The interaction between song number and listeners' perceptions of whether a transition was generated by a human or by a computer has no significant impact on which transitions were human or computer-generated, or how the listeners rate compatibility or location.

Significance Values (p):

Source of Variation	isHumanTruth	Compatibility	Location
Song #	-	0.009	0.015
isHumanChoice	.041	0.000	0.000
Song #*isHumanChoice	.529	0.176	0.141

Conclusions:

1. Song number has a significant impact on how the listeners rate compatibility or location.
2. Listeners' perceptions of whether a transition was generated by a human or by a computer have a significant impact on which transitions were human or computer-generated, and how the listeners rate compatibility or location.

3. The interaction between song number and listeners' perceptions of whether a transition was generated by a human or by a computer has no significant impact on which transitions were human or computer-generated, or how the listeners rate compatibility or location.

The following table summarizes key conclusions drawn from the test procedures.

Source of Variation	Variables affected
Listener #	compatibility
Song #	compatibility, location
isHumanTruth	isHumanChoice, compatibility, location
isHumanChoice	isHumanTruth ¹⁴ , compatibility, location

Table 6: Automatic DJ ANOVA key conclusions

The general hypothesis that precedes all of the ANOVA tests asserts that human listeners respond to transitions generated by the Automatic DJ similarly to those generated by a human DJ. The test conclusions, however, prove that listeners' reactions (as measured by the compatibility, location, and isHumanChoice quantifiers) are significantly impacted by whether or not the transition is generated by a human or in computer software. Because listeners react to the two transition types so differently, it can be concluded that the current version of the Automatic DJ program cannot accurately imitate human-quality DJ mixing.

3.2.2 General Listener Feedback

This section summarizes common feedback collected from the free-response questions included in the questionnaire.

¹⁴ Although this effect is statistically significant, it is incorrect to conclude that listeners' perceptions actually affect whether or not a transition was generated by a human.

1. *In your own words, how well do the computer-generated transitions imitate those of a real DJ?* Responses to this question indicated the importance of context when judging the effectiveness of automatically mixed DJ playlists. Listeners noted that they scrutinize DJ performances less when they are outside of a testing environment. A more thorough listening test process would vary the amount of distraction during the testing process to see if setting and context affect the listeners' reactions to the song transitions. Although as one survey responder pointed out, however, perhaps the most prized feature of the Automatic DJ program is that

2. *Imagine hearing these computer generated transitions in a mix at a venue or club. Could they pass for DJ quality, or is it apparent that they were generated by a computer? Does anything specific compromise their credibility?* Listeners cited the repetition of the same transition technique as the strongest indicator of a computer-generated transition. Feedback for this question also argued that one awkward transition amongst ten smooth transitions is enough to draw attention and compromise the quality of a mix.

3. *Do you have any suggestions to improve this program?* The most popular response to this question recommended increased precision when choosing transition points. Similar to answers for the previous question, some responses also called for increased variety in how the program transitions from one song to the next.

The statistical test results and listener feedback contributed to the following list of enhancements that may be used to improve upon the existing Automatic DJ code.

3.3 Enhancements

This section identifies specific program enhancements that still need to be addressed, and offers possible solutions.

1. Variety in Transition Techniques – The program currently uses a 4 beat beat-matched crossfade to transition between songs, whereas a professional DJ uses a variety of techniques to produce multiple creative transitions in a mix. I first recommend replacing the hardcoded 4 beat default for Crossmatching with a randomly selected number from the set $\{2,4,8,12\}$. This simple change will allow for quick transitions that occur over a short amount of time, as well as more gradual transitions that allow for more overlap between songs.
2. More Intelligent Choosing Algorithm – Testing has revealed that the section choosing algorithm is effective at identifying significant changes in song dynamics, but less robust at noticing gradual increases or decreases in loudness common to pre-chorus build-ups or fade-outs. I recommend using multiple filters on the loudness data to not only accentuate areas of consistent loudness, but to also accentuate areas of consistent change in loudness. Whereas a moving average was already used in the former case, I believe some form of a moving derivative would be a sufficient way to highlight these build-up and fade-out regions in the loudness data.
3. Using Sections in Choosing Algorithm – The sections analysis data returned by the Analysis API may provide additional insight when further developing the choosing algorithm. One could try comparing demarcations produced by the

choosing algorithm with those of the sections list, to either verify or discredit the performance of the algorithm. During the development stage of the Automatic MashUp program, the average loudness of a section seemed to be very relevant to how the sections were demarcated. I touch on how section loudness is used to understand compatibility between different song structures in the following section.

Chapter 4: Automatic MashUp

4.1 Development

The Automatic MashUp program builds on the Automatic DJ by investigating a more complicated music production task of layering multiple tracks. I begin this section by first defining the criteria for a successful mashup, and then detail my software development methodology. I finish the section with a summary of the feedback from human listening tests, and then recommend feasible enhancements to improve the program's performance. The complete Python script of this application is included in Appendix F and a visual summary of how the program operates is included in Appendix G.

A professional mashup producer not only adheres to basic requirements rooted in music theory, but as Hoffner (2013) explains, also makes deliberate stylistic choices to create successful mashups. The former group of basic considerations is common to all mashups, and is described in detail below.

- Key – Every song in the mashup must have the same chromatic scale, or else dissonance will occur.
- Tempo – Every song in the mashup must have a compatible tempo, or else the tracks will become out of sync and sound arrhythmic. The tempo values can be identical, or they can be multiples of each other and still sound compatible.
- Track Separation – Fully mixed tracks cannot be overlaid to create a mashup, because the combination of their instrumental content is almost always too great, and will result in distortion or artifacts. The combination of multiple vocal parts is

usually permissible, but the combination of two entire instrumentals is often inconsistent. Separate acapella and instrumental “component” tracks are instead mixed together to preserve the number of instrumental parts in a song.

Since the Analysis API returns estimates for song key and tempo, I write code that uses the *modify* module to modify the audio files as necessary to ensure that the first two items are upheld. The algorithm calculates the average key and tempo for the two songs, and then calls *modify* functions that shift the pitch and speed of the audio files to their new key and tempo values. The third item cannot be feasibly automated, because track isolation is a very complicated procedure that produces only imperfect results when done manually. I decided to leave the responsibility of collecting component tracks to the user, since they cannot be automatically extracted, but can be found online from websites like YouTube and MP3PoolOnline.

Matching key and tempo is a common sense first step to automatic mashup creation, but determining how to automatically overlay the songs requires more investigation. To gain a better understanding of how component tracks are aligned with each other, I investigated the mashup creation process by manually producing my own set of mashups. I first collected a large number of acapella and instrumental component tracks, and determined the key and tempo for their corresponding fully mixed tracks in my library. I then grouped together songs of similar tempo and key, as shown in the table below.

Mashup Group	Name	Artist	Key	Tempo
1	Just Dance	Lady Gaga	E	119
	In My Head	Jason Derulo	Eb	110
2	Paradise	Coldplay	F	70
	Wide Awake	Katy Perry	F	80
	Day n Nite	Kid Cudi	D	69
3	Good Feeling	Flo Rida	E	128
	Dynamite	Taio Cruz	E	120
	Just Dance	Lady Gaga	E	119
4	Titanium	David Guetta Feat. Sia	Eb	126
	Who's That Chick?	Rihanna	D	128
	Disturbia	Rihanna	D	125
	Parallel Universe	The Red Hot Chili Peppers	Eb	124
5	Don't Stop The Music	Rihanna	A	123
	Every Teardrop Is A Waterfall	Coldplay	A	118
	What I've Done	Linkin Park	Bb	120
6	Firework	Katy Perry	Ab	124
	Paparazzi	Lady Gaga	Ab	115
	Feel So Close	Calvin Harris	G	128
7	When You Were Young	The Killers	B	130
	When Love Takes Over	David Guetta Feat. Kelly Rowland	B	130
	We Run The Night	Havana Brown	C	127
	Lights	Ellie Goulding	B	120
8	Feel So Close	Calvin Harris	G	128
	Last Friday Night	Katy Perry	F#	126
	Please Don't Go	Mike Posner	G	120
9	Burn It Down	Linkin Park	C	110
	Pumped Up Kicks	Foster The People	Eb	128
	I Miss You	Blink-182	B	110

Table 7: Groups of component tracks used for mashup creation

I used the modify module and my written python code to match the tempo and key of the grouped component tracks, and then imported them into GarageBand to be aligned and manipulated manually to gain insight into mashup production. I found that I typically overlaid two songs where their chord progressions were most similar. I subsequently aligned the tracks to match their 4 beat local contexts, and found consistent structural commonality between chord progressions, rhythmic information, and loudness

levels. I decided to include three data sources in my analysis for compatibility in the software.

- Chromatic Content – The chromatic content of a song reveals chord progression patterns, and governs harmonic compatibility between songs.
- Timbre Content – The timbre content of a song best describes its rhythmic patterns, and signifies rhythmic compatibility between songs.
- Loudness Content – The loudness content of a song, as shown in the Automatic DJ program, reveals information about song structure and how energy changes in a song.

These attributes can be more easily visualized as a two-dimensional image. The picture below is as many pixels tall as the number of beats in the song, and thirty pixels wide for each beat. The thirty pixels are comprised of twelve chroma values, twelve timbre values, and six identical loudness values. The loudness data on the far right are black, because they are mainly negative values, but they become more relevant when processing the image later.

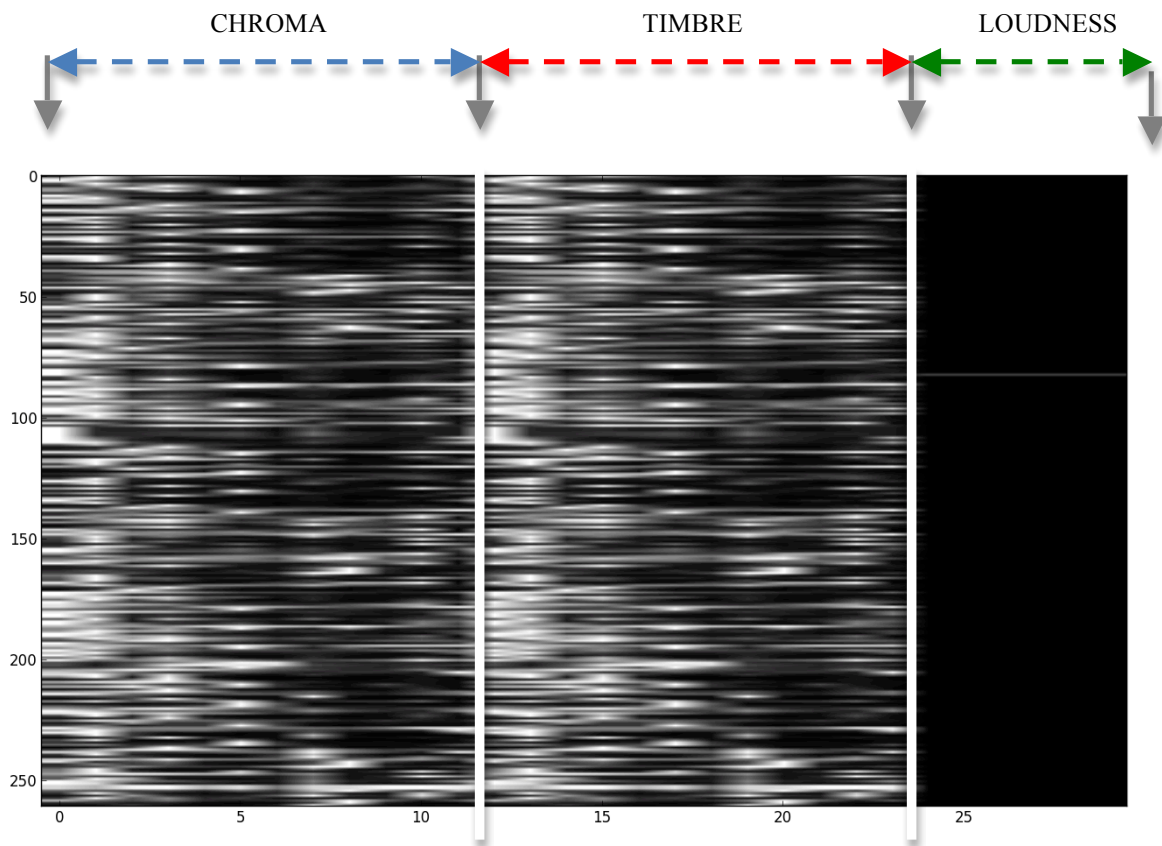


Figure 9: Chroma, timbre, and loudness per beat

I used these matrices to calculate compatibility between two songs. Since the goal of a mashup is to combine compatible sections from different tracks, I used template matching to find the best match for each section, or template, in one song within the entirety of the other song. The following diagram shows how four different analysis sections of one song are compared to the entirety of the other song in four separate template matching procedures.

Template Matching

Slide each section of Song 2 through all of Song 1 to find the best match...

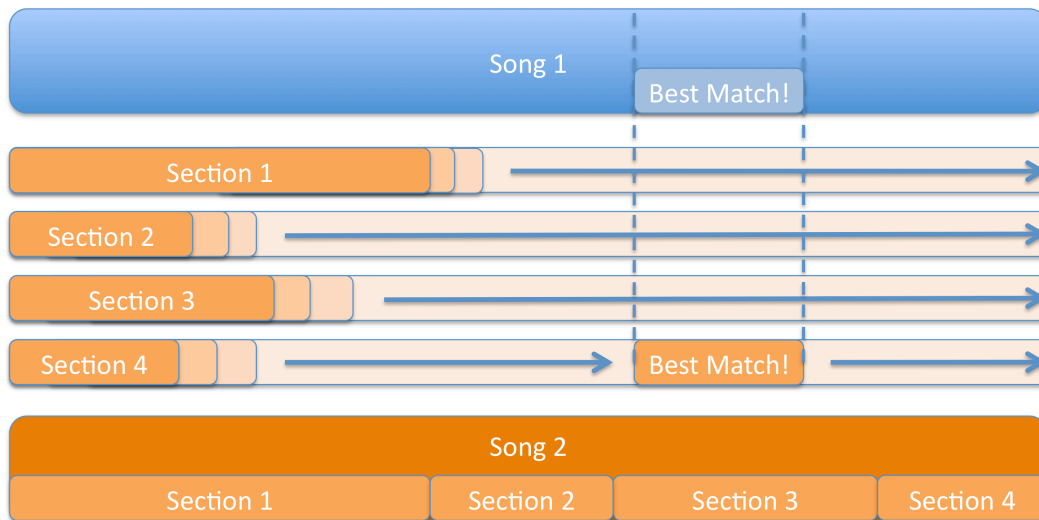


Figure 10: High-level visual depiction of template matching.

I placed the result of each template matching procedure in the same image, and then designated the pixel with the highest correlation as the center of the best-matched template. The image below shows the actual results for all four of the template matching processes depicted in the above diagram.

Template Matching Result for 4 Sections

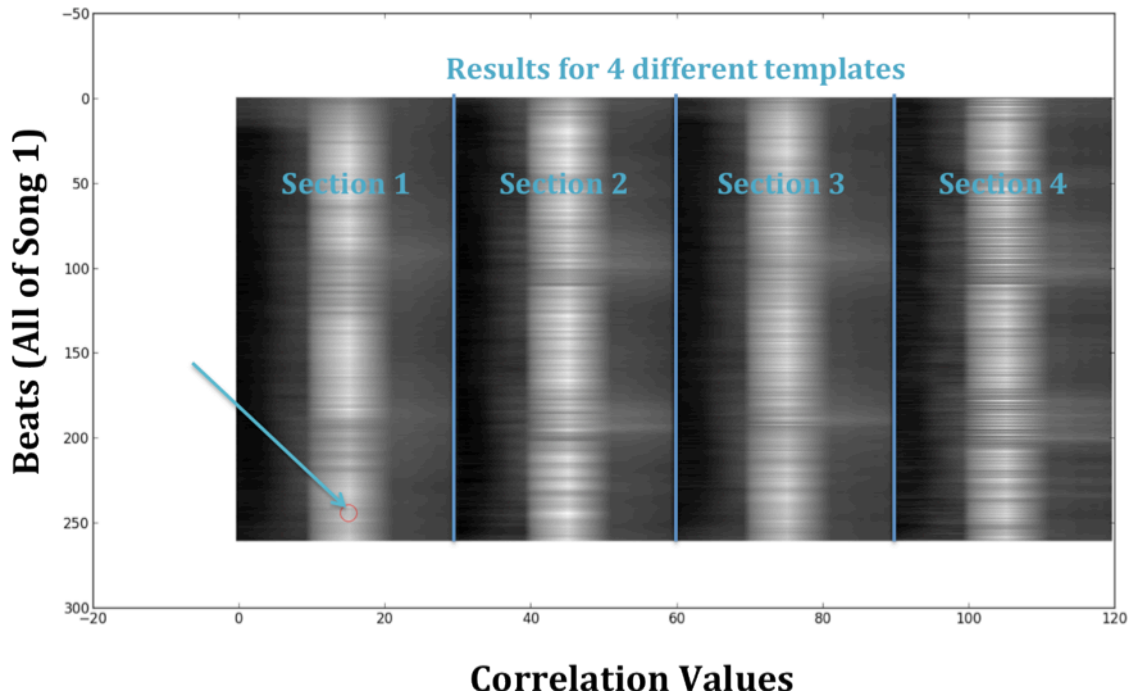


Figure 11: Four analysis section templates matched with another song

Each pixel is a correlation value that signifies how well matched the section is with a specific position in the other song. The sections are of different length, but are all 30 values wide and therefore separated by 30 on the x-axis. The red circle designates the best correlation value, or the location of the best-matched template, and its pixel location derives the section number and corresponding best-matched beats.

After determining the best matching sections, their comprising beat lists were further aligned so that their local context values were in the same sequence. This measure ensured that beat 1 of one song matched with beat 1 of the other song. I then mixed together the corresponding sections of the two tracks' respective vocal and/or instrumental tracks. The vocal version of the new song (labeled "Song 2" in the figure

below) is combined with the instrumental version of the original song (labeled “Song 1” in the figure below) for the entirety of their best-matched sections. The combination is then given context by a few seconds of the original vocal track before and after fading to the new vocal.

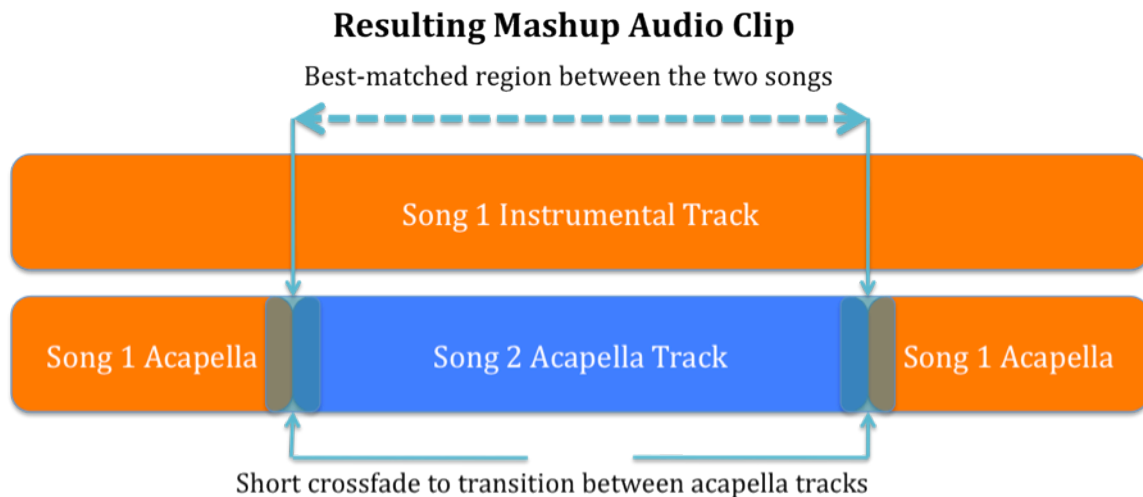


Figure 12: Automatic MashUp output audio clip.

The result of the procedure is a short audio clip used to demonstrate the software’s ability to complete its original program requirements.

Functional testing produced some successful mashup pairs, but because the program relies on accurate analysis data from the Remix API, it suffered from frequent inaccuracies in this data. Estimations for the song key were sometimes incorrect, which stopped the program from correctly matching the key of the component songs. Analysis data for beat location and local context was also unreliable, so song sections could not always be aligned perfectly. The two images below show four beats of two different tracks being aligned according to beat number. The program’s attempt to properly align the two tracks (left) is compromised by incorrectly assigned beat numbers for Song 2, yielding an awkwardly aligned combination of the two different songs.

To eliminate these sources of error, input tracks were verified for correctness in their beat and key analysis data before being used as program input. To pick test cases for human listening tests, I manually analyzed 72 possible input tracks, and found that only five had fully reliable analysis information.

Name	Artist
Fallin	K. Michelle
Louboutins	Jennifer Lopez
Please Don't Go	Michael Posner
Titanium	David Guetta Feat. Sia
Good Girls Like Bad	Jadyn Maria ft. Flo
Boys	Rida

Table 8: Test cases with fully verified beat and key information

I used the five tracks to automatically generate ten mashups for human listening tests.

4.2 Testing

Similar to the Automatic DJ testing procedure, human test subjects rated a ten-item playlist of computer and human-generated audio clips. The survey is included in Appendix D, and the table below shows the mashups used in the study. The notation “Song1 X Song2” means that Song1 served as the instrumental track upon which the acapella track of Song2 was overlaid.

Name	Duration (secs)	Generated By...
Fallin X Louboutins	31	Human
Fallin X Please Don't Go	30	Computer
Titanium X Good Girls Like Bad Boys	22	Computer
Titanium X Louboutins	34	Computer
Please Don't Go X Titanium	24	Computer
Please Don't Go X Fallin	16	Human
Louboutins X Fallin	26	Human
Louboutins X Good Girls Like Bad Boys	30	Computer
Good Girls Like Bad Boys X Please Don't Go	47	Human
Good Girls Like Bad Boys X Titanium	25	Computer

Table 9: Mashup audio clips used in human testing

The full test results are included in Appendix H, and briefly summarized in the table below. Listeners guessed whether a mashup was created by a human or by software, and rated how well the rhythm, harmony, dynamics, and structure of the two songs combined on a scale from 1 to 10.

Origin	Rhythm	Harmony	Dynamics	Structure	% Marked Human
Computer	6.10	6.19	6.49	5.68	36.14
Human	8.00	7.78	7.95	7.51	68.75
Total	6.87	6.83	7.08	6.415	47.08

Table 10: Average test results for Automatic MashUp

The table shows average ratings and the percentage of responses that believed a mashup type to be human. The twenty responses are broken down into two categories based on whether the mashup was truly human-generated, or actually computer-generated. The

table shows that human-generated mashups were on average rated higher than computer-generated mashups in all four categories categories. Also presented above is that human-generated mashups were typically perceived to be human-generated more often than computer-generated transitions.

Testing reveals a clearly more positive response for human-generated mashups, as well as interesting relationships between variables that significantly affect results. Statistical testing procedures used in section 3.2.1 are applied to draw conclusions from the survey responses.

4.2.1 Statistical Results

The following ANOVA tests use a significance threshold of .05 to determine which variables impacted the listeners' responses. Each test answers the following questions for different combinations of Factor 1, Factor 2, and Result variables:

1. Does the independent variable Factor 1 have a significant impact on the dependent Response variable?
2. Does the independent variable Factor 2 have a significant impact on the dependent Response variable?
3. Is there an interaction between independent variables Factor 1 and Factor 2 that significantly impacts the dependent Response variable?

Each unique combination of factors is placed into its own test group, and each test group is comprised of three tests against three different response variables. The variables used in the table are defined as follows:

- Listener #: Listener number is the unique integer used to identify one of the twenty listeners.
- Song #: Song number is the unique integer used to identify one of the ten audio clips used in each of the listening tests.
- isHumanTruth: A Boolean value used to declare whether an audio clip is human-generated (isHumanTruth = 1) or computer-generated (isHumanTruth = 0).
- isHumanChoice: A Boolean value used to declare whether an audio clip was perceived to be human-generated (isHumanChoice = 1) or computer-generated (isHumanChoice = 0).
- Rhythmic compatibility (Rcomp): A measure of how well the rhythmic attributes of two different tracks combine to make a mashup.
- Harmonic compatibility (Hcomp): A measure of how well the harmonic or chromatic attributes of two different tracks combine to make a mashup.
- Dynamic compatibility (Dcomp): A measure of how well the dynamic characteristics of two different tracks combine to make a mashup.
- Structural compatibility (Scomp): A measure of how well the structures of two different tracks combine to make a mashup.

Group	Test	Factor 1	Factor 2	Response
A	1	Listener #	isHumanTruth	isHumanChoice
	2	Listener #	isHumanTruth	Rcomp
	3	Listener #	isHumanTruth	Hcomp
	4	Listener #	isHumanTruth	Dcomp
	5	Listener #	isHumanTruth	Scomp
	6	Listener #	isHumanChoice	isHumanTruth
B	7	Listener #	isHumanChoice	Rcomp
	8	Listener #	isHumanChoice	Hcomp
	9	Listener #	isHumanChoice	Dcomp
	10	Listener #	isHumanChoice	Scomp
C	11	Song #	isHumanChoice	isHumanTruth
	12	Song #	isHumanChoice	Rcomp
	13	Song #	isHumanChoice	Hcomp
	14	Song #	isHumanChoice	Dcomp
	15	Song #	isHumanChoice	Scomp

Figure 13: Automatic MashUp ANOVA tests.

The hypotheses and results for each group of tests are summarized below. For each test, the two factors and their interaction are each evaluated as sources of variance, and considered to have a significant impact on the response if the significance value (p) is less than the significance threshold of .05. Conclusions for each group summarize the software’s ability to produce human-quality output.

Group A

Hypotheses:

1. Listener number has no significant impact on listeners’ perceptions of which mashups are human or computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.
2. The factor of whether a mashup was generated by a human or by a computer has no significant impact on listeners’ perceptions of which mashups were human or

computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.

3. The interaction between listener number and whether a mashup was generated by a human or by a computer has no significant impact on listeners' perceptions of which mashups were human or computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.

Significance Values (p):

Source of Variation	isHumanChoice	Rcomp	Hcomp	Dcomp	Scomp
Listener #	0.818	0.243	0.000	0.000	0.000
isHumanTruth	0.000	0.000	0.000	0.000	0.000
Listener #*isHumanTruth	0.117	0.018	0.030	0.204	0.004

Conclusions:

1. Listener number has no significant impact on the listeners' perceptions of which mashups were human or computer-generated, or how the listeners rate rhythmic compatibility. Listener number does, however, have a significant impact on how listeners rate harmonic, dynamic, and structural compatibility.
2. The factor of whether a mashup was generated by a human or by a computer has a significant impact on listeners' perceptions of which mashups were human or computer-generated, and how the listeners rate rhythmic, harmonic, dynamic, and structural compatibility.
3. The interaction between listener number and whether a mashup was generated by a human or by a computer has no significant impact on listeners' perceptions of which mashups were human or computer-generated, or how the listeners rate dynamic compatibility. The interaction does, however, impact how listeners rate rhythmic, harmonic, and structural compatibility.

Group B

Hypotheses:

1. Listener number has no significant impact on whether mashups were human or computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.
2. Listeners' perceptions of whether a mashup was generated by a human or by a computer have no significant impact on which mashups were human or computer-generated, but do have a significant impact on how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.
3. The interaction between listener number and listeners' perceptions of whether a mashup was generated by a human or by a computer has no significant impact on which mashups were human or computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.

Significance Values (p):

Source of Variation	isHumanTruth	Rcomp	Hcomp	Dcomp	Scomp
Listener #	1.000	0.094	0.000	0.000	0.000
isHumanChoice	0.000	0.000	0.000	0.000	0.000
Listener #*isHumanChoice	0.075	0.000	0.007	0.020	0.001

Conclusions:

1. Listener number has no significant impact on whether mashups were human or computer-generated, or how the listeners rate rhythmic compatibility. Listener number does, however, have a significant impact on how listeners rate harmonic, dynamic, and structural compatibility.

2. Listeners' perceptions of whether a mashup was generated by a human or by a computer have a significant impact on which mashups were human or computer-generated, and how the listeners rate rhythmic, harmonic, dynamic, and structural compatibility.
3. The interaction between listener number and listeners' perceptions of whether a mashup was generated by a human or by a computer has no significant impact on which mashups were human or computer-generated, but does have a significant impact on how listeners rate rhythmic, harmonic, dynamic, and structural compatibility.

Group C

Hypotheses:

1. Song number has no significant impact on how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility, and is completely related to whether or not a mashup is computer or human generated.
2. Listeners' perceptions of whether a mashup was generated by a human or by a computer have no significant impact on which mashups were human or computer-generated, but do have a significant impact on how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.
3. The interaction between song number and listeners' perceptions of whether a mashup was generated by a human or by a computer has no significant impact on which mashups were human or computer-generated, or how the listeners rate rhythmic, harmonic, dynamic, or structural compatibility.

Significance Values (p):

Source of Variation	isHumanTruth	Rcomp	Hcomp	Dcomp	Scomp
Song #	-	0.000	0.000	0.003	0.000
isHumanChoice	0.000	0.000	0.000	0.000	0.000
Song #*isHumanChoice	0.081	0.000	0.000	0.009	0.000

Conclusions:

1. Song number has a significant impact on how the listeners rate rhythmic, harmonic, dynamic, and structural compatibility.
2. Listeners' perceptions of whether a mashup was generated by a human or by a computer have a significant impact on which mashups were human or computer-generated, and how the listeners rate rhythmic, harmonic, dynamic, and structural compatibility.
3. The interaction between song number and listeners' perceptions of whether a mashup was generated by a human or by a computer has no significant impact on which mashups were human or computer-generated, but does have a significant impact on how listeners rate rhythmic, harmonic, dynamic, and structural compatibility.

The following table summarizes key conclusions drawn from the test procedures.

Source of Variation	Variables affected
Listener #	Hcomp, Dcomp, Scomp
Song #	Rcomp, Hcomp, Dcomp, Scomp
isHumanTruth	isHumanChoice, Rcomp, Hcomp, Dcomp, Scomp
isHumanChoice	isHumanTruth ¹⁵ , Rcomp, Hcomp, Dcomp, Scomp
Listener #*isHumanTruth	Rcomp, Hcomp, Scomp
Listener #*isHumanChoice	Rcomp, Hcomp, Dcomp, Scomp
Song #*isHumanTruth	Rcomp, Hcomp, Dcomp, Scomp

Table 11: Automatic MashUp ANOVA key conclusions

The general hypothesis that precedes all of the ANOVA tests asserts that human listeners respond to mashups generated by the Automatic MashUp similarly to those generated by a human mashup artist. The test conclusions, however, prove that listeners' reactions (as measured by the Rcomp, Hcomp, Dcomp, Scomp, and isHumanChoice quantifiers) are significantly affected by whether or not the transition is generated by a human or by computer software. Because listeners react to the two mashup types so differently, it can be concluded that the current version of the Automatic MashUp program cannot accurately imitate human-quality DJ mixing.

The test results also reveal that variation between listeners and interactions between factors affected all five of the response variables. The human listeners were therefore less reliable judges of mashups than they were of transitions in the previous chapter. Future testing may require human listeners with a stronger understanding of music production.

¹⁵ Although this effect is statistically significant, it is incorrect to conclude that listeners' perceptions actually affect whether or not a mashup was generated by a human.

4.2.2 General Listener Feedback

This section summarizes common feedback collected from the free-response questions included in the questionnaire.

1. *How well do the computer-generated mashups replicate the quality of a human music artist?* The collective answer to this question indicated that the software is incapable of imitating human output, but responses differed in their reasoning. Some listeners divulged that they tended to focus on just one or two of the four musical attributes defined in the study (rhythm, harmony, dynamics, and structure) to judge the overall quality of a mashup. The responses suggest that a mashup with well-rounded compatibility scores may not be as effective as a mashup that has one strong compatibility score tailored to a subset of listeners with similar taste.

2. *Can you identify anything specific about the computer-generated mashups that compromise their credibility?* Listeners answered that human-generated mashups seemed to have more deliberate section combinations, especially in terms of rhythmic and structural compatibility.

3. *Do you have any suggestions to improve this program?* The most popular response to this question recommended more precise track alignment. Many responses also called for better compatibility in one of the four attributes.

4.3 Enhancements

This section identifies specific program enhancements that still need to be addressed, and offers possible solutions.

1. Local Context Alignment – Because the software aligns songs based on unreliable local context analysis data, the output will sometimes sound euphonious, but often sound awkwardly arrhythmic. Although I am not confident such a simple change would totally ameliorate the issue, I would first recommend testing the program without any local context alignment, and only rely on the template matching procedure to align the data correctly.
2. More Precise Track Mixing – The instrumental and acapella tracks are currently mixed together using a 50/50 loudness ratio. Some tracks include artifacts from loudness values going above the audio codec’s maximum loudness threshold. I recommend comparing instrumental and acapella tracks to their full studio mix to better identify the correct mixing ratio for both tracks, and also include a measure to normalize audio data before encoding.
3. Key and Tempo Matching – The software calculates average key and tempo values to shift the pitch and speed of the mashup component songs. Although the current method achieves its intended purpose of harmonic and temporal compatibility, testing revealed that shifting both the instrumental and acapella tracks by an equal amount is overly simplistic. Acapella tracks were more robust to pitch and tempo shifting procedures, whereas instrumental tracks easily degraded in sound quality. Instead of averaging the songs’ key and tempo attributes, I recommend shifting the acapella track to match the instrumental track’s key and tempo.
4. User-defined Matching Parameters – Song sections are currently compared and matched using a fixed weighting of timbre, chroma, and loudness attributes. I

recommend further defining the relationship between how different weightings of these metrics in the software affect how listeners rate the rhythmic, harmonic, dynamic, and structural compatibility of the mashups. I also recommend considering how weighting differs between listeners to allow users to customize software parameters according to their specific listening tastes.

Chapter 5: Conclusion

The results of this project yield some successful application of prior MIR research in both software programs, and some examples of where imperfect data metrics limit the performance of the applications. Because the Automatic DJ program relied mainly on consistently accurate loudness data, its performance was more reliable, and showed promise in its ability to model the output of a human DJ. Even when a song's beat information was inaccurate, the Crossmatch method used to transition between songs was often sufficient in covering up the unaligned beats. The Automatic MashUp program produced some examples of success, but ultimately fell short in its ability to produce human-quality mashups. The software's success relied heavily on accurate analysis data, and its development was thwarted by the challenge of finding songs with correct key and beat predictions. If an application such as this is to be developed in the future, it must be preceded by techniques for more robust key estimation and beat detection. Ultimately, an inaccurate library of data limited the success of the The Automatic MashUp program.

The progress made with this software also shows that music production algorithms can be identified and automated, but that the variability in the output of the imitated work must be matched by the algorithm's complexity. For both programs, a few basic considerations were identified as common to the entire production process, and played an important role in guiding both development phases. These limited sets of variables were sufficient for making basic programs, but music production will always contain an infinite amount of things that can be unexpectedly altered, so there will always be a need to make these automatic algorithms even more complex.

The criticisms for the programs that were raised during human listening tests all call for an increase in variability. The programs may need to be more robust in understanding input songs of greater variability, as demonstrated by the Automatic DJ's ability to identify sudden changes in loudness, and its inability to identify gradual build-ups in loudness. The programs may also require greater variability in how they generate output, like the enhancement that calls for random transition durations between songs in the Automatic DJ.

Finally, the greatest enhancement for these programs would be to combine their functionality and build upon their ability to produce music. The tasks of mixing and matching are often used by DJs simultaneously in a performance, so a combination of both programs could serve as a useful application for professional disk jockeys. One could even try using studio samples from multiple different instrument and vocal tracks as input to the Automatic MashUp program to fabricate "multi-track mashups," or ultimately, original music.

The results of this project will hopefully provide new insight to the millions of active musicians in the digital age. Written music has taken on a new, digital format, and has gained the powerful capability of computational analysis. By furthering MIR research and applying its findings, music composers will be able to expand their creativity, and music admirers will acquire greater understanding.

References

- Bartsch, M. A., & Wakefield, G. H. (2001). To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the* (pp. 15-18). IEEE.
- Bertin-Mahieux, T., Weiss, R. J., & Ellis, D. P. (2010). Clustering beat-chroma patterns in a large music database. In *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference, August 9-13, 2010, Utrecht, Netherlands* (pp. 111-116). International Society for Music Information Retrieval.
- Tsai, W. H., Yu, H. M., & Wang, H. M. (2005). A query-by-example technique for retrieving cover versions of popular songs with similar melodies. In *Int. Symp. on Music Information Retrieval (ISMIR)* (pp. 183-190).
- Ellis, D. P., & Poliner, G. E. (2007, April). Identifying cover songs' with chroma features and dynamic programming beat tracking. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on* (Vol. 4, pp. IV-1429). IEEE.
- Randel, D. M. (2003). *The Harvard dictionary of music*. Belknap Press.
- Lewis, J. P. (1995, May). Fast normalized cross-correlation. In *Vision interface* (Vol. 10, No. 1, pp. 120-123).
- Smidansky, M. (2012, October 20). Personal Interview.
- Hoffner, K. (2013, January 3). Personal Interview.
- Template image of a coin. Digital image. *Scikit-Image: Image Processing in Python*. N.p., n.d. Web. 27 Mar. 2013. <http://scikit-image.org/docs/dev/auto_examples/plot_template.html>.
- Large image of many coins. Digital image. *Scikit-Image: Image Processing in Python*. N.p., n.d. Web. 27 Mar. 2013. <http://scikit-image.org/docs/dev/auto_examples/plot_template.html>.
- Match_template result. Digital image. *Scikit-Image: Image Processing in Python*. N.p., n.d. Web. 27 Mar. 2013. <http://scikit-image.org/docs/dev/auto_examples/plot_template.html>.

Appendix A: Automatic DJ Python Script

```
#!/usr/bin/env python
# encoding: utf=8
'''
Created on Oct 15, 2012
@author: jordanhawkins
'''

import echonest.audio as audio
import echonest.action as action
import echonest.selection as selection
import os
import plistlib
import shutil
import urllib
import numpy.matlib as matlib
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as pyplot

workingDirectory = '/Users/jordanhawkins/Documents/workspace/Automatic DJ/src/root/nested'
programFiles = ['__init__.py', 'Main.py', 'AutoMashUp.py']
lib = plistlib.readPlist('/Users/jordanhawkins/Music/iTunes/iTunes Music Library.xml')
LOUDNESS_THRESH = -8 # per capsule_support module
MIX_LENGTH = 4 # defines the length, in beats, of fades between songs
'''

Remove any old audio files from the project directory.
'''

def flushDirectory():
    for filename in os.listdir(workingDirectory):
        if programFiles.count(filename) == 0:
            os.remove(filename)

'''

Copy audio files listed in the "Automatic DJ Input" iTunes playlist
into the project directory.
'''

def getAudioFiles():
    for count in range(len(lib['Playlists'])):
        if lib['Playlists'][count]['Name'] == 'Automatic DJ Input':
            playlistItems = lib['Playlists'][count]['Playlist Items']
            trackIDs = [i['Track ID'] for i in playlistItems]
            for i in range(len(trackIDs)):
                location = lib['Tracks'][str(trackIDs[i])]['Location']
                location = urllib.unquote(location)
                try:
                    shutil.copy(location[16:], workingDirectory)
                except:
                    print "exception in getAudioFiles"
            break

'''

Find the longest consistently loud region of the song.
window is the approximate number of segments in a 16 beat span.
'''
```

lpf is a 16-beat-long rectangular window *convolved* with the loudness values to smoothen them the while loop tries to find a loud region of the song that's at least 60 seconds long. If such a region cannot be found the first time, the `LOUDNESS_FLOOR` value is increased to tolerate slightly softer loud regions for the sake of a longer song duration.

**To better understand the mathematics involved, note that loudness is measured in negative decibels, so a small negative number is louder than a large negative number.*

```

"""
def findLoudestRegion(segments,tempo):
    segmentMarkers = []
    for segs,tempo in zip(segments,tempo):
        LOUDNESS_CEILING = .8
        LOUDNESS_FLOOR = 1.2
        window = int((16.0/tempo)*60.0/(matlib.mean(matlib.array(segs.durations))))
        lpf = np.convolve(segs.loudness_max,np.ones(window)/window)[window/2:-window/2]
        lpf[0:window/2] = lpf[window/2]
        lpf[-window/2:] = lpf[-window/2]
        mean = matlib.mean(matlib.array(lpf))
        marker1 = 0
        finalMarkers = (0,0)
        foundFirstMarker = 0
        while((sum([s.duration for s in segs[finalMarkers[0]:finalMarkers[1]]]) < 60.0)
            and LOUDNESS_FLOOR < 2.0):
            for i,l in enumerate(lpf):
                if(foundFirstMarker):
                    if l < mean*LOUDNESS_FLOOR or i == len(lpf)-1:
                        foundFirstMarker = 0
                        if((i-marker1) > (finalMarkers[1]-finalMarkers[0])):
                            finalMarkers = (marker1,i)
                    elif l > mean*LOUDNESS_CEILING:
                        foundFirstMarker = 1
                        marker1 = i
                # adjust thresholds to allow for longer region to be chosen, if necessary
                LOUDNESS_FLOOR = LOUDNESS_FLOOR + .05
                LOUDNESS_CEILING = LOUDNESS_CEILING + .05
            segmentMarkers.append(finalMarkers)
    return segmentMarkers

```

"""

Collects the names of the audio files in the project directory, then creates a list of `LocalAudioFile` objects, which gets returned along with key analysis objects.

"""

```

def getInput():
    filenames = []
    for filename in os.listdir(workingDirectory):
        if programFiles.count(filename) == 0:
            filenames.append(filename)
    inputList = []
    for filename in filenames:
        try:
            inputList.append((audio.LocalAudioFile(filename).analysis.tempo['value'],
                audio.LocalAudioFile(filename),filename))

```



```

    except: print "Exception in getInput for filename: ", filename
    inputList.sort()
    localAudioFiles = [t[1] for t in inputList]
    return [f.analysis.segments for f in localAudioFiles], [t[2] for t in inputList],[f.analysis.tempo['value'] for
f in localAudioFiles], localAudioFiles, [f.analysis.beats for f in localAudioFiles]
"""

```

I copied this method from capsule_support. It equalizes the volume of the input tracks.

```

def equalize_tracks(tracks):
    def db_2_volume(loudness):
        return (1.0 - LOUDNESS_THRESH * (LOUDNESS_THRESH - loudness) / 100.0)
    for track in tracks:
        loudness = track.analysis.loudness
        track.gain = db_2_volume(loudness)
"""

```

This method deletes the original input songs from the directory so they don't get copied into iTunes.

```

def deleteOldSongs(filenamees):
    for filename in os.listdir(workingDirectory):
        if filename in filenamees:
            os.remove(filename)
"""

```

This method generates 4-beat Crossmatch objects, then renders them attached to the end of Playback objects.

```

def generateCrossmatch(localAudioFiles, beatMarkers, filenames, beats):
    actions = []
    for i in range(len(beatMarkers)-1):
        cm = action.Crossmatch((localAudioFiles[i], localAudioFiles[i+1]),
            ((b.start, b.duration) for b in beats[i][beatMarkers[i][1] -
            MIX_LENGTH:beatMarkers[i][1]]),(b.start, b.duration)
            for b in beats[i+1][beatMarkers[i+1][0]:beatMarkers[i+1][0]+
            MIX_LENGTH]))
        actions.append(cm)
    for i in range(len(beatMarkers)):
        startBeat = beats[i][beatMarkers[i][0]+MIX_LENGTH]
        endBeat = beats[i][beatMarkers[i][1]-MIX_LENGTH]
        actions.insert(2*i, action.Playback(localAudioFiles[i],
            startBeat.start, endBeat.start-startBeat.start))
    action.render([action.Fadein(localAudioFiles[0],beats[0]
        [beatMarkers[0][0]].start,beats[0][beatMarkers[0][0]+MIX_LENGTH].
        start-beats[0][beatMarkers[0][0]].start),"000 fade in"])
    for i in range(len(actions)/2):
        index = str(i+1)
        while(len(index) < 3): index = "0" + index
        try:
            action.render([actions[2*i],actions[2*i+1]], index + " " + filenames[i])
        except: print filenames[i]
    index = str(len(filenames))
    while(len(index) < 3): index = "0" + index
    action.render([actions[-1]], index + " " + filenames[-1])
    action.render([action.Fadeout(localAudioFiles[-1],beats[-1][beatMarkers[-1][1]-
        MIX_LENGTH].start,beats[-1][beatMarkers[-1][1]].start-beats[-1]

```

```

        [beatMarkers[-1][1]-MIX_LENGTH].start)], "999 fade out")
"""
This method finds the closest beat to my designated segments for Crossmatching.
I had some trouble with this method, which is why it is so fragmented. I may resolve
this later.
"""
def getBeatMarkers(loudnessMarkers,segments,beats):
    return [(b.index(b.that(selection.overlap(segments[i][loudnessMarkers[i][0]]))[0]),
            b.index(b.that(selection.overlap(segments[i][loudnessMarkers[i][1]]))[0]))
            for i,b in enumerate(beats)]
"""
This method probably won't be used as much as Crossmatch.
It immediately transitions between songs, without any kind of mix.
"""
def generateFaderSlam(localAudioFiles, beatMarkers, filenames):
    actions = [action.Playback(laf, b[0].start, (b[1].start-b[0].start)) for laf,b in
zip(localAudioFiles,beatMarkers)]
    for i in range(len(actions)):
        action.render([actions[i]],str(i) + " " + filenames[i])

def main():
    flushDirectory()
    getAudioFiles()
    segments, filenames, tempos, localAudioFiles, beats = getInput()
    equalize_tracks(localAudioFiles)
    loudnessMarkers = findLoudestRegion(segments,tempos)
    beatMarkers = getBeatMarkers(loudnessMarkers,segments,beats)
    generateCrossmatch(localAudioFiles,beatMarkers,filenames,beats)
    deleteOldSongs(filenames)
    os.system('automator /Users/jordanhawkins/Documents/workspace/'
        + 'Automatic\ DJ/import.workflow')

if __name__ == '__main__':
    main()

```