Auto DJ Mixing

Vivek Jayaram, Saroj Kandel and Samarth Singal
Harvard College, CS182 - Artificial Intelligence

1) Using Fourier Transform for Mashability

We consider the aural similarity of all pairs of songs to compute a metric for mashability. Each song is marked with one mix-in point and one mix-out point. For each pair of these songs, 32 beat long samples are considered. Each song is in a particular key and particular key combinations sound well together. This is because most sounds in the song are in similar frequency bands called semitones.

We use a library - Librosa - that samples the mix-in/ mix-out sections at 22 kHz and produces a power array representation of the music. We then use a stepwise Fast-Fourier Transform (FFT) to bin the power series into the 12 semitones of music.

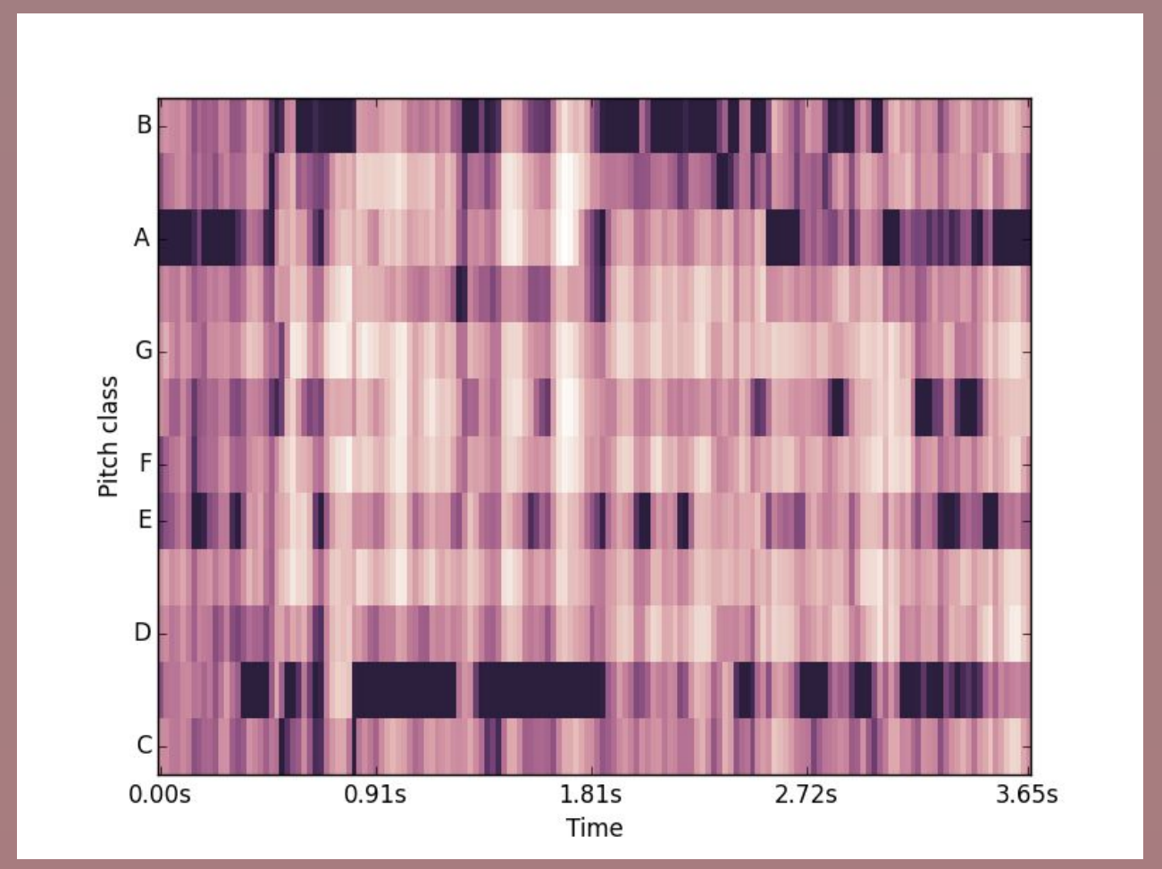


Fig 1. Song 1 chromagram

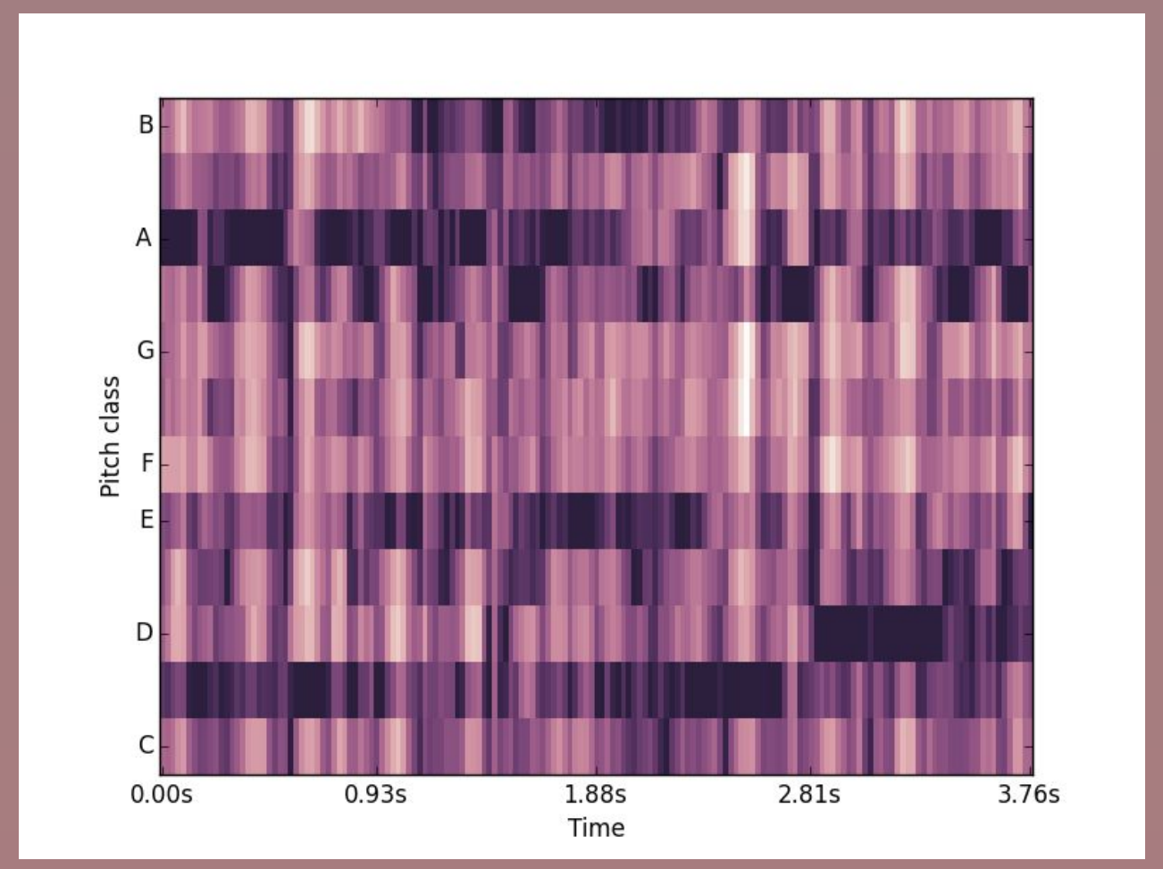


Fig2. Song 2 chromagram

We thus decompose each song beat into a 12-dimensional chroma vector corresponding to semitones. If this matrix of [12 * length of song] representation for both song samples is similar, then they should mix well. We use the cosine distance between corresponding chroma vectors in the two songs and return the mean cosine distance as the mashability score.

2) Backtracking With Forward Searching

We cast the problem of generating the mix as a Constraint Satisfaction Problem. There are m variables, each corresponding to a slot in the mix, and |m-1| factors each connecting successive variables. A factor f1 between x1 and x2 constraints (x1, x2) to values where the mashability is higher than some value, epsilon.

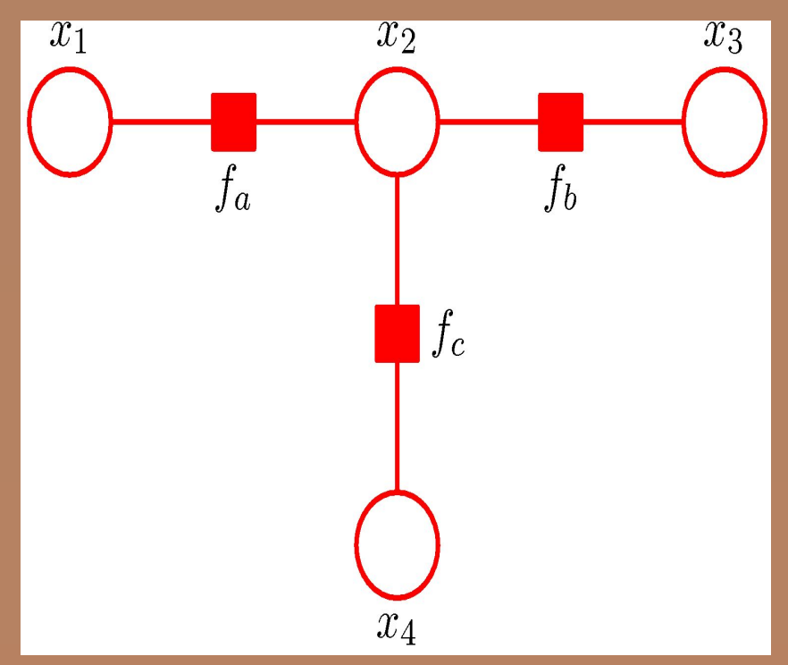


Fig 1: Factor Graph

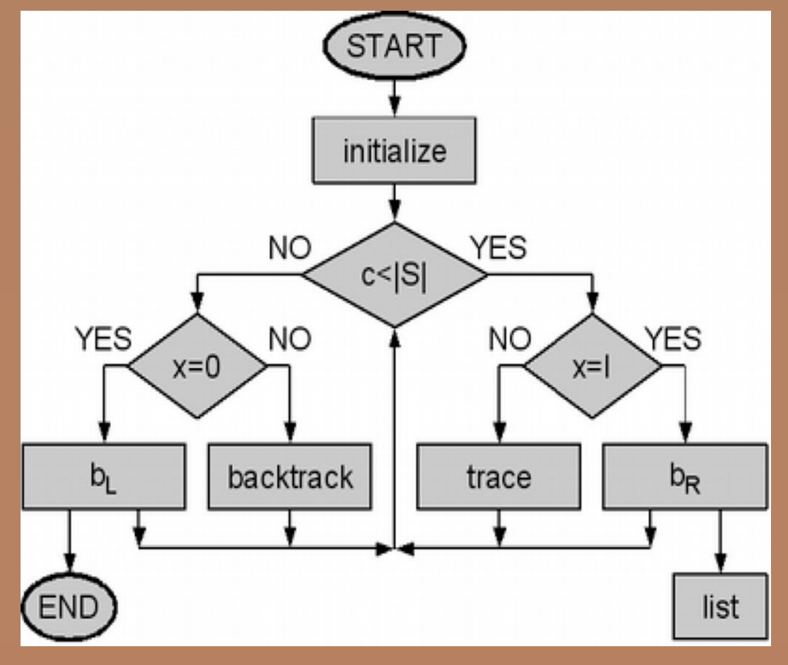


Fig 2: Backtracking

For the implementation of mix generation, we used backtracking algorithm with forward searching. The function starts by picking the first uninitialized variable in the slot, and then assigns the next best song in its domain by checking all the constraints. The forward checking happens when the next best song returns -1, in which case we prune the domain and recurse on updated domain.

Introduction

The goal of this project was to build a system to create DJ style mixes of songs. The ability to combine songs in a pleasing way is highly desirable by listeners and is a feature of live parties, radio shows, and online recordings. While being a good DJ takes a certain element of human creativity, certain aspects of a mix follow clear rules and could be automated.

Approach

Our approach to this problem was divided into three main sections

1) Determine how well songs transition using signal processing

2) Use CSPs to generate a mix where all transitions meet a minimum threshold

3) Generate the mix and output it as an mp3

Results

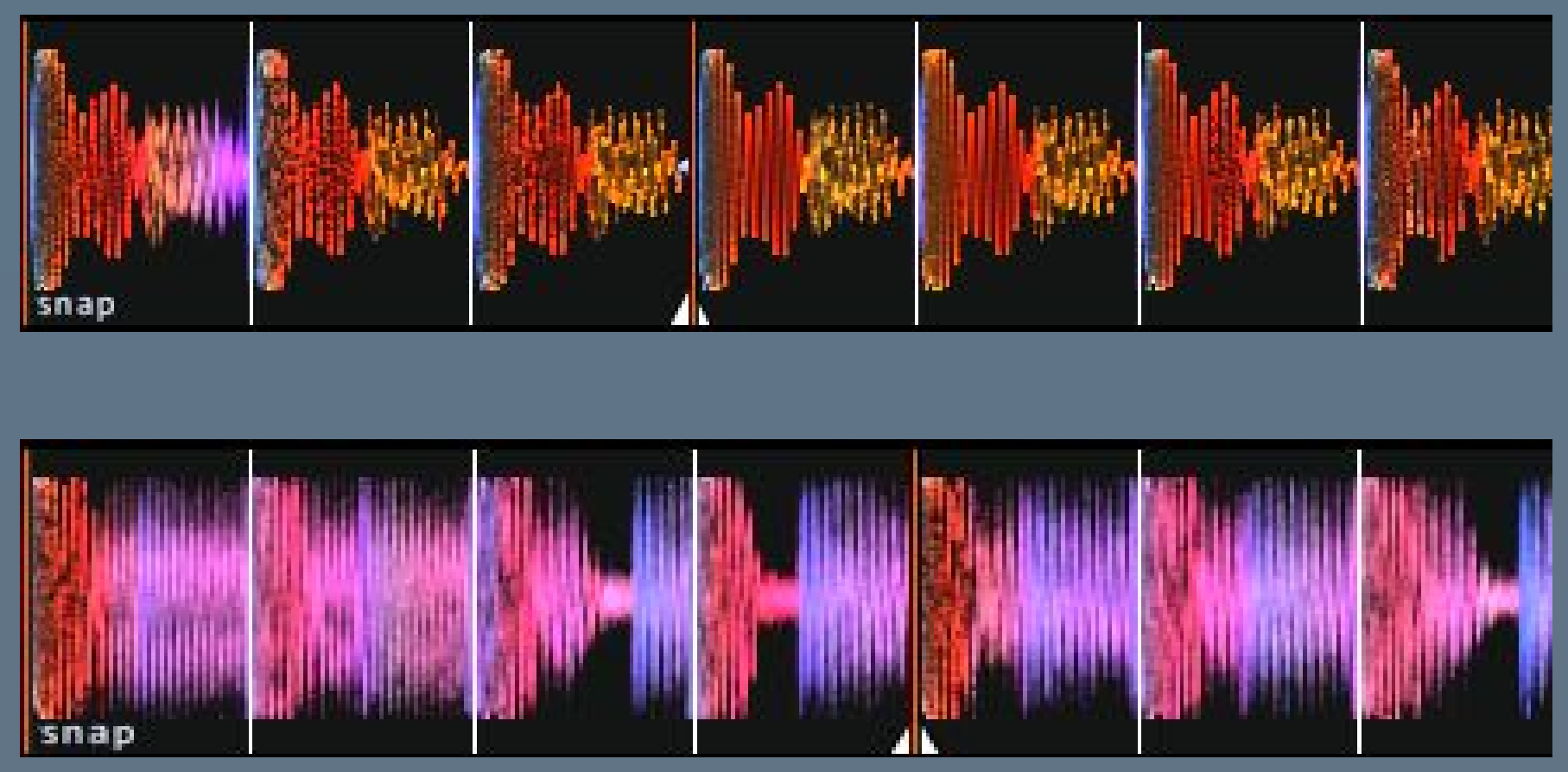
We found that the semitone chromograms did a good job of figuring out which songs went well together. The CSP algorithm was quick in finding the solution, and the output was easily rendered using EchoNest's python API. There were many powerful libraries used that greatly reduced the amount of code we had to write, particularly when it came to signal processing.

Discussion and Further Work

We choose to manually pick mix-in/ mix-out points for each song. Ideally, the AutoDJ can figure out the most optimal transition points for any given pair of songs. In practice, this increases the complexity of mashability by a quadratic factor. Further work could be done to determine optimal alignment instead of trying every combination of transition points.

Beat Matching and Outputting

Certain aspects of making a DJ mix were handled by the EchoNest Remix library. One such aspect was beatmatching. The library detects the downbeats based on volume and matches up two songs so they play synchronously. The API returns a list of audio objects where each item contains the audio for one beat. This makes it easy to manipulate the audio by beats rather than by time and select a desired region.



The waveform of two songs with the downbeats shown

Another aspect handled by the EchoNest API was crossfading the song. Both linear and exponential crossfades were used, and exponential gave better results since the original song could be audible for a longer period of time.

References

Davies, M. et al.. AutoMashupper: An Automatic Multi-Song Mashup System. 2013. http://telecom.inesctec.pt/~mdavies/pdfs/DaviesEtAl13-ismir.pdf

Muller, Meinard. Signal Processing for Music Analysis. 2011. https://www.ee.columbia.edu/~dpwe/pubs/MuEKR11-spmus.pdf

Acknowledgements

We would like to acknowledge the various libraries that handled the majority of low level signal processing

- EchoNest Remix https://echonest.github.io/remix/
- Librosa https://github.com/bmcfee/librosa
- Dirac http://www.musicdsp.org/showone.php?id=201