TAMPERE UNIVERSITY OF TECHNOLOGY

BISHWA PRASAD SUBEDI

AUDIO-BASED RETRIEVAL OF MUSICAL SCORE DATA

Master of Science Thesis

Supervisor: Associate Professor Anssi Klapuri
Examiners:
Associate Professor Anssi Klapuri
Adjunct Professor Tuomas Virtanen
Examiner and topic approved by the faculty counsil of the Faculty of Computing and Electrical Engineering on 05.03.2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SUBEDI, BISHWA PRASAD : Audio-Based Retrieval on Musical Score Data
Master of Science Thesis, 72 pages
July 2014
Major: Multimedia
Examiners: Associate Professor Anssi Klapuri and Adjunct Professor Tuomas Virtanen, Department of Signal Processing, Tampere University of Technology.
Keywords: Content-based Music Information Retrieval, Music, Locality Sensitive Hashing, Dynamic Time Warping, MIDI, Audio, Transcription.

Given an audio query, such as polyphonic musical piece, this thesis address the problem of retrieving a matching (similar) musical score data from a collection of musical scores. There are different techniques for measuring similarity between any musical piece such as metadata based similarity measure, collaborative filtering and content-based similarity measure. In this thesis, we use the information in the digital music itself for similarity measures and this technique is known as content-based similarity measure.

First we extract chroma features to represents musical segments. Chroma feature captures both melodic information and harmonic information and is robust to timbre variation. Tempo variation in the performance of a same song may cause dissimilarity between them. In order to address this issue we extract beat sequences and combine them with chroma features to obtain beat synchronous chroma features.

Next, we use Dynamic Time Warping (DTW) algorithm. This algorithm first computes the DTW matrix between two feature sequences and calculates the cost of traversing from starting point to end point of the matrix. Minimum the cost value, more similar the musical segments are. The performance of DTW is improved by choosing suitable path constraints and path weight. Then, we implement LSH algorithm, which first indexes the data and then searches for a similar item. Processing time of LSH is shorter than that of DTW. For a smaller fragment of query audio, say 30 seconds, LSH outperformed DTW. Performance of LSH depends on the number of hash tables, number of projections per table and width of the projection. Both algorithms were applied in two types of data sets, RWC (where audio and midi are from the same source) and TUT (where audio and midi are from different sources).

The contribution of this thesis is twofold. First we proposed a suitable feature representation of a musical segment for melodic similarity. And then we apply two different similarity measure algorithms and enhance their performances.

This thesis work also includes development of mobile application capable of recording audio from surroundings and displaying its acoustic features in real time.

PREFACE

This thesis work has been performed at the Institute of Signal Processing of Tampere University of Technology. This work has been funded as an ongoing research project work at the department of signal processing.

I am so thankful to almighty god for his blessings, guidance and courage to complete this task. In particular, I would like to pay my heartiest gratitude to my supervisor Associate Professor Anssi Klapuri for his guidance throughout my work. He had given me an opportunity to work in the Audio Research Group and answered all my questions. His guidance and support helped me to stand-in my work in the correct direction.

I would like to thank to all people who had helped me to complete my thesis. I would like to thank all the member of Audio Research Group for their support. The time I spent in the Audio Research Group of TUT has been very memorable.

Last, but not the least, I would like to thank my parents, grandparents for supporting me throughout my academic studies. It is my pleasure to thank my friends for their support and motivation to chase this degree.

Tampere, July 2014

Bishwa Prasad Subedi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACF	Automated Collaborative Filtering
BSC	Beat-Synchronous Chroma
CBSM	Content-Based Similarity Measure
CD	Compact Disc
CENS	Chroma Energy Normalized Statistics
CP	Chroma-Pitch
DCT	Discrete Cosine Transform
IDCT	Inverse Discrete Cosine Transform
DTW	Dynamic Time Warping
F0	Fundamental Frequency
E^2LSH	Exact Euclidian LSH
IOI	Inter Onset Interval.
LSH	Locality Sensitive Hashing
MIR	Music Information Retrieval
MIDI	Musical Instrument Digital Interface
MBSM	Metadata-Based Similarity Measure
NN	Nearest Neighbor
OMR	Optical Music Recognition
RWC	Real World Computing
SMF	Standard MIDI File
WWW	World-Wide Web

1. INTRODUCTION

Nowadays music is easily available and accessible in digital forms. Amount of digital music contents are also increasing far more quickly than our ability to process them. Release of a new song, remix of old popular songs, cover songs from different artists, songs performed by other artists are main reasons for the increase in musical contents. Often same songs are stored in different format as well like audio, score and compressed. Every day, numerous music files are uploaded and downloaded from different media sharing websites. There are many music downloading and streaming service operating on World-Wide Web. Because of this it is very easy to collect music. With the increase of those multimedia data, there arises a problem of managing those data such as searching and retrieving music content. Assume a situation where you want to play one of your favorite songs from a collection of a few thousand songs. If you know the artist name or title of a song then you could directly filter it from your local collection or you could search online by artist/title. Suppose you know few words from the lyrics of the song then you could find the name of the song/artist by using Google. Once you have the song title and/or artist you can find the song easily. But imagine a situation where you only have the melody of a song, or a classical music theme then how would you search for that particular piece of a music. Or you may have an audio song (or a small piece of audio song) and want to retrieve its musical score.

It is not only about listeners, due to growth of consumers in digital music, there is an opportunity for researchers to discover trends and patterns in digital music world. For example, researchers may be interested in discovering the trends on online music sales [64], music recommendations, evaluation of music, finding repeated section in music [44]. At present common and effective approach for recommending similar music, for example, is based on usage. Last.fm builds a database of millions of users and songs. It then recommends you song based on "users who liked artist/songs A and B that you are currently listening also liked songs C and D". Figure 1.1 shows the music recommended by last.fm, while I was listening to the song by Michelle Branch. Music tastes of individuals are so different that you cannot produce precise information for an individual just by averaging the opinions.

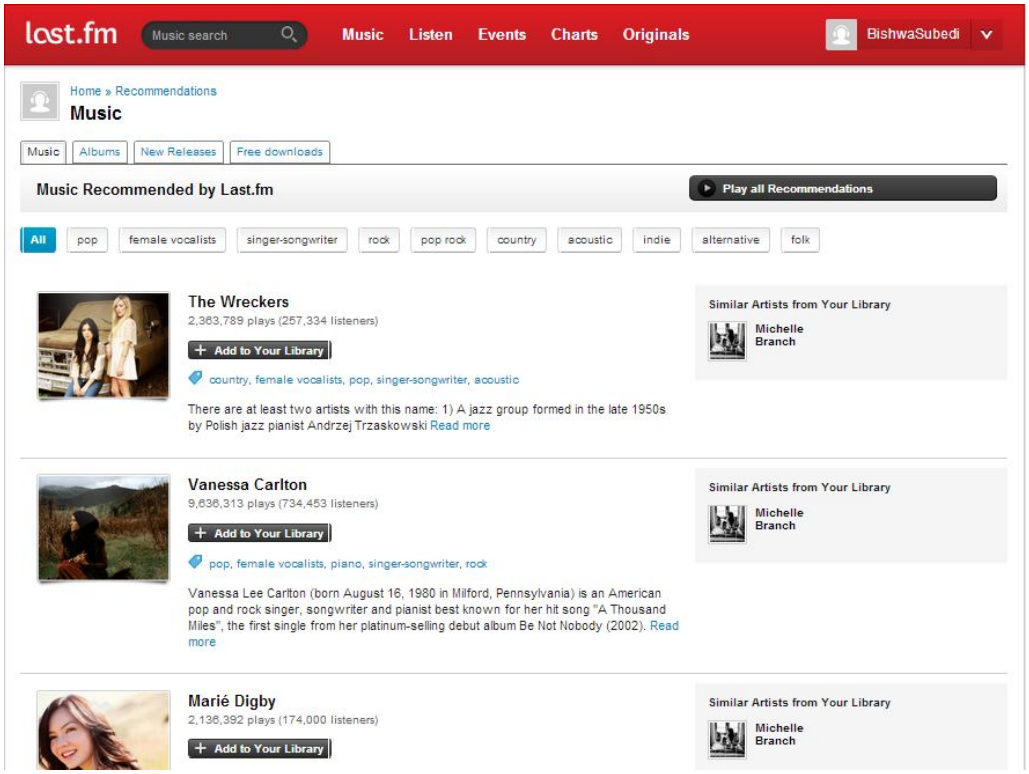


Figure 1.1: Music recommended by last.fm.

1.1 Scope and Purpose of thesis

The objective of this thesis is to investigate content-based similarity measures for retrieving musical score data based on an audio query. We need to develop strategies that enable us to access those music collections for search and browse functionality. This strategy is commonly known as Music Information Retrieval (MIR) and is a hot topic among researchers. There are many aspects of MIR, in this thesis work we focus on content-based retrieval technique. This thesis proposed a system capable of retrieving similar musical score data based on audio query. You may use full audio file or a small segment of an audio piece; the proposed system retrieves one or more similar musical score file. In this thesis, we use feature vectors to represent any musical segment, and those feature sequences are used to find similarity between songs. Although an audio can be represented by several features, this thesis is limited to chroma features and beat sequences. In this thesis, experiments were performed using only MIDI and audio file format. In order to retrieve similar item from the database, this thesis use Dynamic Time Warping and Locality Sensitive hashing algorithm.

The search engine capable of retrieving musical scores similar to the query can be helpful for musicologists to find out how their work are related to the work of other composer or their own earlier works as well. Although this task can be done

by musicologists manually, it would be interesting if computer would perform this task reasonably well with much less effort. This system can be used to solve or avoid copyright infringement. Composers can find out if someone is plagiarizing them or if their new work exposes them to the risk of plagiarism.

This system can also be used in musical stores, suppose customer may only know the tune from a record but not any other information as artist name, song title. It would be interesting if computers do the task of identifying melodies and suggesting the records.

1.2 Literature Review

Music item is multi-faceted object. When we talk about music, it has its own aspects, for example title, artist, genre, mood, melody, rhythm etc. In digital form, music is multimedia document. Some researchers believe that music similarity measure is one of the fundamental concepts in the field of MIR [7]. Music similarity can be used for:

- Search: Based on similarity measure, we can find nearest music item to the query item.
- Recommendation: We can list similar musical items to the one that user is currently listening to, which means recommending similar music to user.
- Browsing: Based on similarity measure, one can organize large music collection into cluster or into hierarchies so that user can easily navigate within those music collections.

There are different ways to find out similarity between music [44; 8]

- Metadata-Based Similarity Measure (MBSM).
- Usage-Based Approach.
- Content-Based Similarity Measure (CBSM).

Metadata-Based Similarity Measure

First approach to find similarity between music is through textual metadata, known as Metadata-Based Similarity Measure (MBSM). In this technique, we match the textual information of a query and database items to find similarity between them. Sometimes textual information associated with the item is sufficient for similarity measure. Currently, many music service providers are based on this technique and achieve success to some degree. This method is very fast. Some commercially successful MBSM driven music system ask user to provide the name of their favorite

artist or song title and then it will create a playlist based on artist similarity [44]. With the help of provided information, those system use metadata to find an artist similarity or a genre similarity and then retrieves few tracks that user might be interested in hearing. In MBSM, query is simple and easy.

But sometimes it is very difficult to manage a meaningful set of metadata for huge database because many people are employed to create description and inconsistency in the description may impact the search performance. Moreover, such textual information assigned to music recordings provides very limited information about the content of that particular song. The time and human resources taken to prepare such database is enormous. And it is also difficult to describe the melody of music in human language. Suppose a polyphonic music is played, and you were asked to describe the melody of that music, it won't be easy for you to describe the melody in your own language.

Usage-Based Similarity Measure

Second and most common approach in similarity measure is Usage-based approach, which is also known as collaborative filtering. Collaborative filtering has roots from something we people have been doing from centuries - sharing opinions. For example, if Bishwa's colleagues say that they liked a latest music released by Beyonce, then he might consider buying and/or listening it. If majority of them did not like the song at all, he might decide to spend his money somewhere else. Bishwa might observe that Kevin recommend the type of music that he likes. Lee has a history of recommending song that he hates. Clark might recommend everything. Over a time, Bishwa learns whose opinion he should listen and how can he apply those opinion to help him finding a music item of his choice.

Collaborative filtering works by constructing a database of preference for music by users. Based on listening history, any new user Bishwa is matched against the database to find out neighbor, who are other users having similar taste to Bishwa. The music items liked by the neighbor are recommended to Bishwa, and he will probably like those items as well. This method has been successful in both research and practice. Figure 1.1 and 1.2 shows the item recommended by last.fm and youtube respectively.

One of the earliest implementations of collaborative filtering based recommendation system is Tapestry [13], which rely on opinions of people from a close-knit community like office. Earlier automated collaborative filtering (ACF) systems include GrounpLens research system [25; 53], Ringo [61], Video Recommender [62]. All of these systems first collect ratings from users and then compute the correlations between pairs of user in order to identify user's neighbor and make a recommendation based on the ratings from those neighbors. Other examples of collaborative filtering

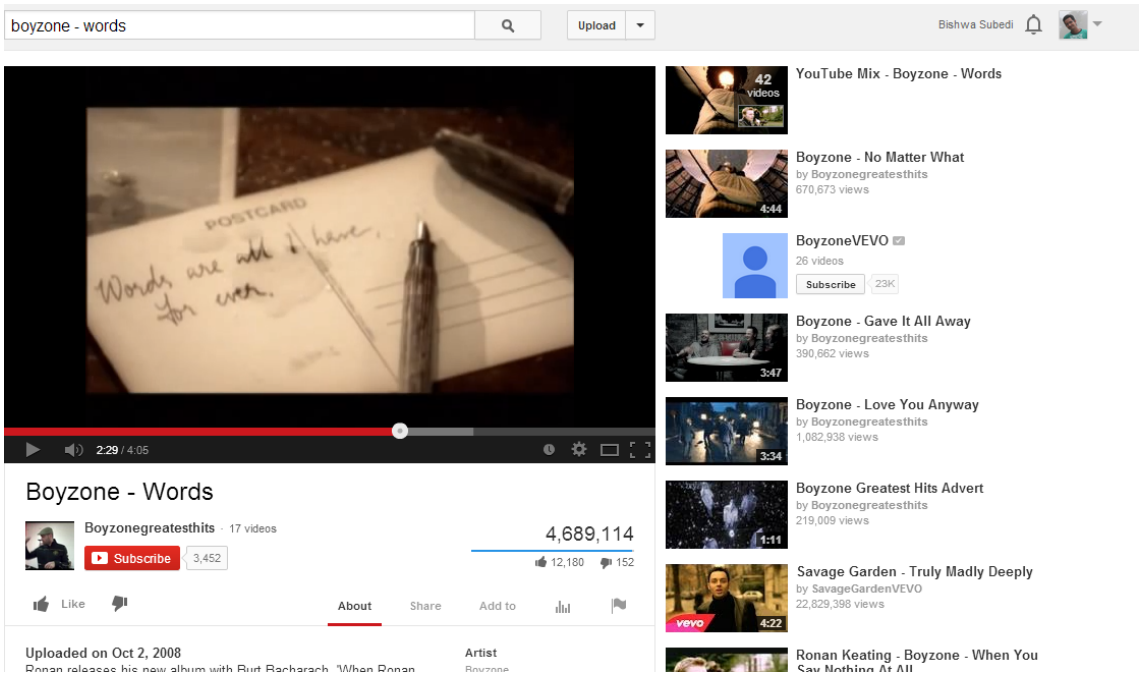


Figure 1.2: Recommendation from Youtube on playing BoyZone's song.

based recommender system include the book recommendation system from Amazon.com, the PHOAKS system to find relevant information in WWW [34], Jester system which recommend jokes [31]. Collaborative filtering based recommender system has also been applied to other technologies including Bayesian networks [27], Clustering [35] and Horting [9].

Bayesian Network model that can be built offline using training sets, creates a decision tree at each node and edge represents user information. This model is small, fast and as accurate as nearest neighbor method [27]. In clustering technique, clusters are created based on a user having similar preferences. Prediction for a user can be made by averaging the opinions of other users in that cluster. Horting is a graph based technique where node represents the users and edge between nodes represent the degree of similarity between two users. Prediction for a user is done by walking the graph to nearby nodes and combining the opinion of nearby users.

One challenge for collaborative filtering based recommendation system is to improve the quality of the recommendation for the user. User needs trustable recommendation systems which can help them find the item/music they are looking for. Next challenge is to improve the scalability of the system.

Content-Based Similarity Measure

Another approach in music similarity measure is based on the audio content of a music piece known as Content Based Similarity Measure (CBSM). This approach

has its root in information retrieval and information filtering research. Content-based method use the information in the digital audio. Audio features contain the information about musical work and music performance. This method identifies what user is looking for, even though user doesn't know exactly what he/she is looking for. For example, if user has a small segment of music clip, but don't know about artist or songs, this method is helpful. In this technique, we match the audio content of query music to the database music. Some people may focus on the specific aspect of the content, for example, tempo, melody etc. Some researchers do not care about specific aspect, and they derive some features to describe the whole contents of an audio and use these features to match between the query audio and audio in the database.

Earlier work on content based music retrieval includes [4], where author use music segment to retrieve music from the database using beat and pitch information. "Query-by-humming" system takes user-hummed tone as an input and matches it against the database of MIDI files [5]. [63] propose efficient algorithm to retrieve similar audio, based on spectral similarity, where each audio file are processed to find local peaks in signal power. Near each peak, spectral vector is extracted to represents the music. Specially-defined distance vector is used to match two music pieces. [52] proposed a system that align audio and MIDI and search through the database of MIDI file using query audio using chroma feature and Dynamic Time Warping (DTW) algorithm. The example of commercial content-based system is shazam.com described in [6]. It can identify music recording taken by mobile device at restaurants, dance club, pub, car, home and returns the artist, album and track title with a nearby location where user can buy, or link to purchase online and download. There are also systems where user can provide melody as a query and the system identifies the correct match. Soundhound.com identifies the song you sing or hum. Some systems where user can query by humming or singing are midomi.com, musicline.de and musipedia.org.

Whenever performing similarity measure, it is not advisable to compare two songs directly. While processing full songs we need to deal with huge amount of data, which leads to huge memory consumption and high computational load. Instead it is good idea to extract features that best describe the music content. Such collection of feature known as feature vectors and are used to compare between music signals. One should be careful choosing the relevant features because similarity is not well defined. We can define similarity between any music piece based on instrumentation, harmonic content and rhythm. If one defines similarity based on instrumentation then one should use Mel-frequency cepstral coefficients (MFCCs) explained in [3]. If one needs to retrieve similar music based on harmonic content, then he/she will prefer chroma features and rhythmogram is suitable for rhythmic similarity [32].

This thesis work deals with chroma feature, beat sequences of any musical file and combination of both chroma and beat.

After feature extraction of music segment, next step is to determine similarity between two music segments. Dynamic Time Warping (DTW) is well known algorithm to find similarity between two sequences of feature vectors. For example, one can use DTW to find out similarity in walking pattern, or even it can detect if one person is walking slower or faster as compared to other. DTW is applied in many applications to measure similarity between audio, music, video etc. Initially, Dynamic time warping was used to compare speech patterns in automatic speech recognition. In the field of information retrieval and data mining, DTW is successful in coping with time-deformation and different speeds related with time-dependent data.

1.3 Structure of a Thesis

The structure of this thesis follows: Section I gives a brief introduction on Music Information Retrieval (MIR). It provides insight about scope and purpose of the thesis. This Section also provides a literature review on MIR. Section II is focused on background knowledge required to understand this thesis. It provides insight about music terminologies, different types of music representation. In section III, this thesis gives an overview of the proposed system, its components and working principle of the system. In section IV, thesis discuss about chroma features, its variants and extraction process. In this section we also discuss about estimating beat sequences and combining beat and chroma to obtain beat synchronous chroma feature. Section V, explain about time alignment, dynamic programming, time normalization constraints, dynamic time warping algorithm, its implementation with results. In section VI, we provide a theory behind locality sensitive hashing, its implementation and results. And finally in section VII, we end this thesis with some concluding remarks.

2. BACKGROUND

This section reviews some methods and tools used in music information retrieval and music content analysis. Music information retrieval (MIR) is a field of science which deals with retrieving information from a music file, searching a database of music file. It is growing field of research with many real-world applications. Some of the application of MIR technique includes music recommender systems, sound source separation and recognition systems, automatic music transcription, music genre classification. This section deals with the important properties of music that are mostly used in information retrieval.

2.1 Subjective Attributes of Musical Sound

Before going into further details of MIR, it is good to introduce some basic terminology related to sound and music.

Waveform is representation of an audio signal that shows the change in air pressure. If change in air pressure follows some regular pattern, then it is termed as periodic waveform and the distance between two successive high or low pressure levels is known as the period of the waveform. The reciprocal of the period is known as fundamental frequency (F0). Periodic sound may contain several frequency components that are multiples of the F0. The air pressure at highest pressure point is known as amplitude of the waveform.

A harmonic of a sound wave is a frequency component of the signal that is an integral multiple of the fundamental frequency. If fundamental frequency (first harmonic) is 20 Hz, then frequencies of next harmonics are 40 Hz for second harmonic, 60 Hz for third harmonic and so on. Harmonics are periodic at the fundamental frequency, so sum of harmonics is also periodic at that frequency. Waveform of periodic sound is shown in Figure 2.1.

Pitch is one of the important perceptual attributes of music signal and is defined as an auditory sensation that allows the ordering of sounds on frequency-related scale ranging from low (slower oscillation) to high (rapid oscillation). More precisely, pitch can be defined as the frequency of a sinusoidal wave that is matched to the target sound by listener [1]. Pitch has a close relation with the fundamental frequency (F0) but they are not the same, Pitch is a subjective attribute whereas fundamental frequency is an objective. Fundamental Frequency (F0) is the inverse of the period.

In musical term, note means pitched sound itself which represents relative duration and pitch of a sound. Two notes are perceived similar to each other if the ratio of fundamental frequencies of those two notes is in the power of two. All notes satisfying this property are grouped as pitch class.

Loudness is the subjective attribute of a sound (music) that describes the amplitude or power of the sound perceived by human ear. It is related with the amplitude (physical strength) of a sound wave and can be ordered on a scale from quiet to loud.

Timbre, also known as tone colour, is a sound quality which helps human ear to distinguish between the sounds produced by two different sources with the same pitch, duration and loudness. With this attribute we can distinguish between the sound produced by guitar and piano even if they are played at same note with same loudness. For a normal listener, timbre is how an instrument sounds like. It is a complex concept and cannot be explained by simple auditory property, it depends on the coarse spectral energy distribution of a sound. In analysis of music signal, timbre is multidimensional concept and is represented by a feature vector, as opposed to pitch, loudness and duration that can be represented by single scalar value.

2.2 Music Terminology

Melody of a song is a series of pitched sounds with musically meaningful pitch and interonset interval (IOI) relationship. IOI means time interval between the beginnings of two sound events. In written music, melody refers to sequences of single notes. Combination of two or more simultaneous notes form chord and it can be harmonic or dissonant. Music theory that studies the formation and relationships of chords is known as harmony.

Musical meter means the regular pattern of strong and weak beats in a musical piece which consists of detecting the moments of musical emphasis in an acoustic signal and filtering them to discover their underline periodicity. Those periodicities at different time interval together form the meter. Perceptually the most salient metrical level is tactus, also known as beat or foot-tapping rate. Tactus can also be considered as the temporal backbone of any musical piece; hence beat tracking plays an important part in music transcription.

2.3 Music Representation

In modern world, digital music contains textual, visual and audio data. Music information is represented in a different format depending upon the particular application, and such representations differ in structure and content. In this section, we focus on three formats of music representation that are widely used. They are Score representation, Audio format and MIDI format.

2.3.1 Score Representation

Score representation gives the symbolic description of a music piece as shown in Figure 2.2. This representation can also be termed as sheet music. The score encodes the musical work and depicts it in a graphical-textual form. Sheet music can be used by a musician as a guide to perform a piece of music. In order to create a performance based on such representation one should have prior knowledge about music notation. In this type of representation, note objects are used to represent music. Notes are given in terms of attributes such as musical onset time, pitch, dynamics, and duration. In order to represent a piece of music by score, one should have good experience on it.

Scores come in various formats:

- Full Score: Full score represents music for all instruments and voices. It is intended for conductor to read while directing rehearsals and performances.
- Piano Score: Piano score, often called as reduction score, is a transcription for piano.
- Lead Sheet: It specifies lyrics, melody and harmony. It can be used to capture important elements of a popular song.

There are different ways to represent digital score. One way is to input sheet music manually in MusicXML format. Similar to all other XML-based formats, it is intended to be easy to parse and manipulate music. This approach is tiresome and error prone. One can also use Music Notation Software (scorewriters) to write and edit digital sheet music, where we can easily input or modify note objects by computer keyboard, a mouse, or an electric piano. Another method is to scan the sheet music and then convert the score into the digital image. For computers, this image is just a set of pixels and cannot create any semantics from it. The next step is to translate digital image into standard encoding scheme like MusicXML to show the semantics of the score symbols. This process is known as Optical Music Recognition (OMR). See [46] for more detail. Current OMR software produce good accuracy rates for sheet music when using a clean scan or good-quality printed score. There are still challenges in OMR [14] which includes small artifacts occurred during a scan. Such artifacts may lead to wrong interpretation problems and to incorrect recognition result.

2.3.2 Audio Representation

Audio signal or sound wave is generated by a change in air pressure due to some vibrating object like vocal cord of a human, string of a guitar etc. When there is

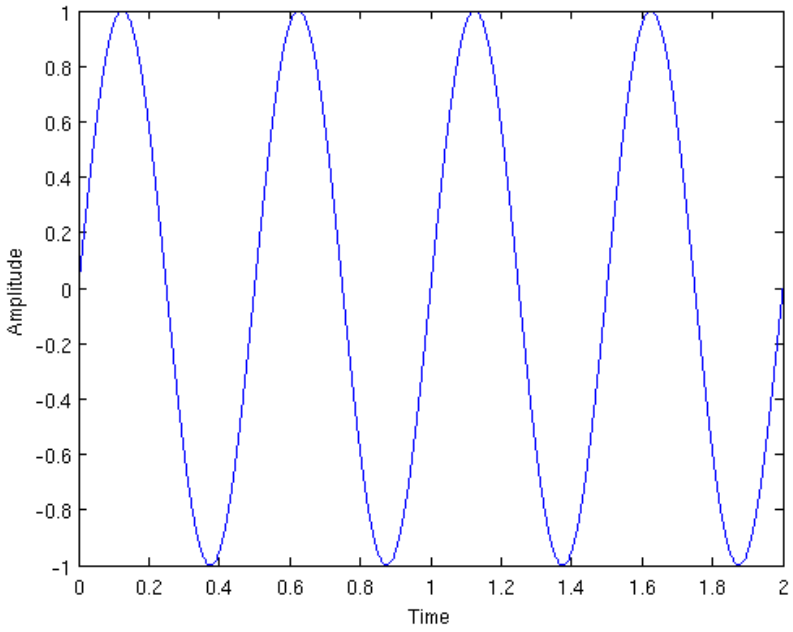


Figure 2.1: Waveform of a periodic sound.

vibration, it causes the displacement and oscillation of air pressure that ultimately cause compression and rarefaction in air. This change in air pressure travels as a wave, from source to the receiver, which can be perceived as sound by human ear. In the case of receiving electronic device (microphone), such air pressure is converted to an electrical signal. Such variation in air pressure is represented as a waveform (sound pressure level as a function of time).

The Audio (waveform) representation would encode all important information (temporal, dynamic, and tonal micro deviations) needed to replicate the acoustic realization of a musical signal. However, the note parameters like onset time, note duration, pitches are not given explicitly in the waveform. Sometimes even single note of sound, played in instrument producing multiple harmonics, becomes complex with noise components and vibrations. As a result, it is difficult to compare and analyze on the basis of waveform representation. For the case of polyphonic music, the complexity of waveform representation increases considerably, because there occurs interference between the components of musical tones, and those mixed components are hard to recover. This makes it difficult to extract note parameters from polyphonic waveform.

The waveform we discussed is the analog waveform with an infinite number of values. Since computer can handle a finite number of values, the analog waveform has to be converted to discrete (digital) form and the process is known as digitization. Digitization of a signal is achieved by sampling and quantization. First waveform

is sampled at uniform interval and then values of the waveform at each sample are quantized to discrete set of values. In the case of compact disc (CD), the waveform is sampled at 44100 Hz (44100 samples per second) and then quantized by using 65536 possible values and finally encode by 16-bit coding scheme. Since digitization is lossy transformation, it is generally not possible to reconstruct the original waveform from the digital representation. Such error is commonly known as quantization errors and aliasing which may sometimes introduce audible sound artifacts. For the case of digital representation used in CD, the sampling rate and quantization values are chosen in such a way that any degradation caused cannot be noticed by human ear. For further detail see [30].

2.3.3 MIDI Representation

MIDI stands for Musical Instrument Digital Interface. Originally it was developed as a standard protocol for controlling and synchronizing digital electronic musical instrument from different manufacturers. MIDI allows musicians to remotely access the electronic instrument or a digital synthesizer. For example, you push down a key of digital piano to start a sound, velocity of keystroke controls the intensity of sound and sound stops as you release the key. Instead of physically doing these activities you may send a suitable MIDI message to trigger the instrument to produce the same sound. MIDI provides the information about how an instrument is played and how to produce music.

MIDI standard was modified in order to include file format specification Standard MIDI File (SMF). SMF tells how to store midi data in the computer and allows exchange of MIDI data between users. SMF also called as MIDI file, contains a collection of message with a time stamp to find out timing of message. Information in MIDI files known as metamessages are important to software that process MIDI files.

The most important MIDI messages are note-on and note-off. Note-on corresponds to the start of a note whereas note-off corresponds to the end of the note. Along with note-on and note-off, MIDI file has time stamp, key velocity, MIDI note number as well as channel specification. MIDI note number encodes the pitch of a note and contains an integer value between 0 and 127. MIDI pitches are based on equal-tempered scale. The key velocity controls the intensity of sound and contains an integer value between 0 and 127. It determines the volume during note-on event and decay during note-off event. However, exact interpretation depends upon respective instruments. MIDI channel is represented by an integer between 0 and 15, which helps the synthesizer to use an instrument that was earlier assigned to that particular channel number. Each channel supports polyphony that means several simultaneous notes. Finally, the timestamp is also an integer that represents the

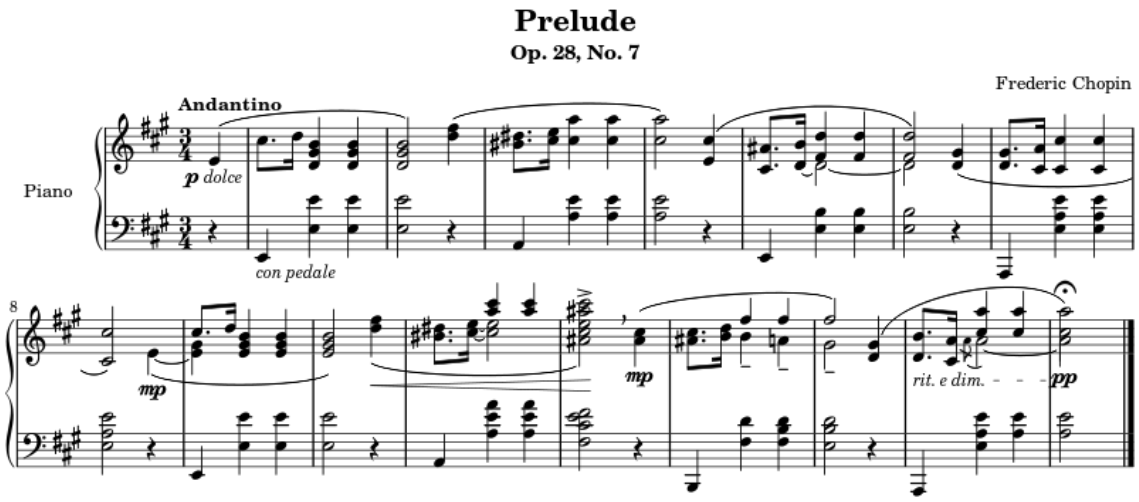


Figure 2.2: An example of modern music notation.

clock pulses to wait before particular note-on command.

An important aspect of MIDI file is that it can handle musical along with physical on-set times and note duration. Like score representation, MIDI also uses timing information in terms of musical entities instead of expressing it in absolute unit of time like microseconds. A quarter note is subdivided into basic time unit known as ticks or clock pulse. Number of pulses per quarter note is mentioned in the header of the MIDI file. Timestamp is used to tell how many ticks to wait before executing MIDI message relative to previous MIDI message. MIDI also allows us to encode and store absolute timing information at much finer resolution level and in more flexible way. MIDI can include the tempo message to specify microseconds per quarter note, and we can use tempo information to compute absolute timing of a tick.

2.4 Music Transcription

In music, Transcription means to write down the aurally perceived music using musical notation, that is, to extract the human readable information from a music recording. It also includes writing down the pitch, beats sequences, onset time, offset time, duration of each individual sound in the music signal. Traditionally music is written by using note symbols that describe pitch, rhythm, beats. The Figure 2.2 shows the notation of music by using note symbol taken from Wikipedia [65]

The main convention in such notation is that time flows from left to right, and the pitch of a note is represented by the vertical position on a staff (fundamental latticework of music notation, upon which symbol are placed) line. In the case of drum, it represents instruments and a stroke type.

In the past, music transcription was done by human musicians by listening the music recording, and the ability to transcribe the music is directly related to ex-

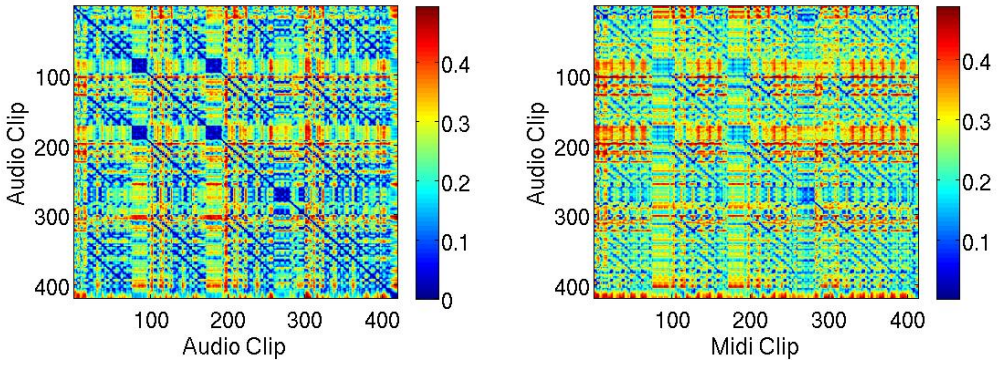
perience and depth of knowledge in music. Average listener may perceive some information from a complex polyphony music recording. One can recognize the musical instrument played, one may be able to tap their foot on beat sequences, one may be able to detect structural parts as well. But untrained listener may not be able to detect the sub melodies; he may only perceive the dominant information but unable to perceive the detailed information. So training is required to analyse the music while listening. Listener should have adequate knowledge of the instruments involved and their musical pattern and playing technique.

In mid 1970s, Moorer proposed a system for transcribing music recording by computer [21]. This was the first step towards automatic music transcription. His work was followed by other researchers in 1980s. All of those early works use only two concurrent sounds and the pitch relationships between those sounds were restricted in various ways. Transcriptions of the percussive instruments were first addressed by Schloss, in which different types of conga strikes was classified in the continuous recordings. Since 1990, the interest in music transcription took the pace. The most common and successful approach was to use statistical methods. And another trend was to utilize computational model of the human auditory system. Anssi Klapuri and Manuel Davy have summarized different signal processing algorithms dedicated to the various aspect of automatic music transcription [1].

Skilled human musicians are far more accurate and flexible than automatic music transcription methods. This means till date we do not have any reliable polyphony music transcription systems. However in the case of limited complexity in the polyphony music, some degree of success has been achieved. For a transcription of pitched instrument, there is a restriction in the number of concurrent sound sources, only the specific instrument is considered, interference between percussive sound and drums is forbidden. Ryynänen and Klapuri demonstrate promising results in transcribing real world music from CD recordings [42]. For the advancement in music information retrieval research, one should examine challenge, results and goals of Music Information Retrieval Evaluation eXchange MIREX [28].

2.5 Similarity matrix

Music is generally self-similar; repetition of certain section is a common phenomenon in almost all music. Some part of a music segment often resemble with another part within the same (or different) music piece, for example, second chorus sounds similar as the first chorus. Similarity matrix provides the graphical representation of the similar segments within the data. Similarity matrix can be of same song or between different songs. If the similarity matrix is calculated for a same song then it is termed as self-similarity matrix. Let us suppose we have a sequences feature vector of one music clip $X = (x_1, x_2, \ldots, x_N)$, where x_i represents a feature vector at given



(a) Similarity matrix of same song.

(b) Similarity matrix for midi and audio of same song.

Figure 2.3: Similarity matrix

interval i, and sequences feature vector of another music clip $Y = (y_1, y_2, \ldots, y_M)$, where y_k represents a feature vector in time frame k. Then Similarity matrix can be defined as $D(i,k) = d(x_i, y_k)$, for $i \in 1, 2, \ldots, N$,$k \in 1, 2, \ldots, M$, where $d(x_i, y_k)$ is a function measuring the similarity between two vectors. Most common similarity measure is cosine distance defined as

$$dc(x_i, y_k) = 0.5 \left(1 - \frac{\langle x_i, y_k \rangle}{\|x_i\| \|y_k\|}\right) \tag{2.1}$$

where $\|.\|$ denotes the vector norm and $\langle ., . \rangle$ dot product. The matrix $D(i,k)$ obtained by taking cosine distance is known as distance matrix. Similarly we can calculate self-similarity matrix where feature vector X and Y represents the same song.

The concept of self-similarity matrix was first used in music by Foote [22] to visualize the time structure of given music recording. The property of self-distance matrix is determined by the feature representation and distance measure. Usually distance measure is used to compare single frames. In order to enhance the structural property of self-distance matrix, it is good to add local temporal evolution of the feature. Foote [22] proposed an idea to calculate a distance value by taking the average distance from a number of successive frames. This results in the smoothing effect of the self-distance matrix. Later Müller and Kurth [47] suggest contextual distance measure to handle local tempo variation in audio recording. As an alternative of using sliding window, compute the average distance of a feature vector of non-overlapping musically meaningful segments such as music measure [23]. Another approach suggested by Jehan [60] was to compute self-distance matrix at multiple levels starting from individual frame to musical pattern.

Self-distance matrix is visualized in two-dimensional space as shown in Figure

2.3. Figure 2.3(a) is visualization of the distance matrix of same audio file. Here time runs from left to right and top to bottom, so top-left corner represents the start of the feature vector, and bottom-right corner indicates the end. At any given instance the colour at point (x, y) represents the similarity between the features at instance x and y. Dark blue at point (x, y) means there is less distance (more similarity) between features at instance x and y where as red colour means there is more distance (dissimilarity) between features at x and y. If feature describes some musical properties and remain constant over a certain duration, then block of low distance is formed, and size of the block tells the duration of the constant feature. Instead of remaining constant, if feature describes some sequential properties, then diagonal stripes of low distance are formed.

There is always a diagonal blue line passing from top-left corner to bottom-right corner, because the feature vector is always same to itself at particular instance of time. Repeating pattern in the sequences of feature vector $(x_1, x_2, x_3, \ldots, x_N)$ of a musical segment are often visible in the distance matrix. If some part of the feature is repeated, we see stripes in the self-distance matrix that runs parallel to main diagonal line, and the separation of those stripes from main diagonal blue line represents the time difference in the repetitions.

Feature only is not responsible for the formation of block or stripe, temporal parameter for feature extraction also plays a vital role [18]. The longer the temporal window, most likely it is that blocks are formed in self-distance matrix. Paulus and Klapuri [24] mentioned working with low resolution benefits for computation along with structural reasons.

Often there is repetition of musical parts in another key. By circularly shifting the chroma feature, Goto [50] simulates transpositions. Later on Müller and Clausen [48] adopt this idea to bring in the concept of transposition-invariant self-distance matrix to show the repetitive structure in the presence of key transposition.

Similarly, we can also calculate cross-similarity matrix where feature vector X and Y in the equation 2.1 represents different song. Figure 2.3(b) is visualization of the distance matrix between midi and audio format of the same song. Figure 2.4 shows the visualization of two completely different songs. We can see from Figure 2.3(a), diagonal is darker as compared to Figure 2.3(b) because distance between same feature sequence is more similar. But in the case of Figure 2.4, we do not see any low cost diagonal line because there is not any similarity between two songs.

One can show repetitive information by transforming self-distance matrix to time-lag format. Figure 2.5 is the time-lag representation of the audio clip that is used in similarity matrix of Figure 2.3. In case of distance matrix D both axis represents the time whereas in case of time-lag matrix R, one of the axis represents the difference

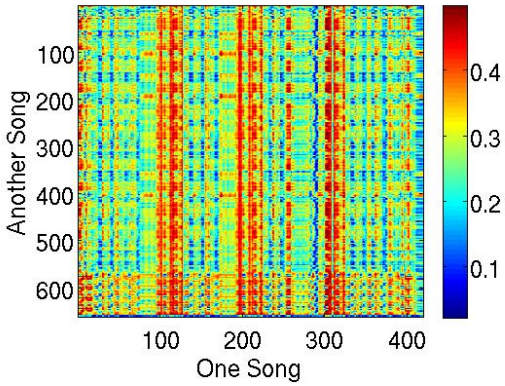


Figure 2.4: Distance matrix for two entirely different songs.

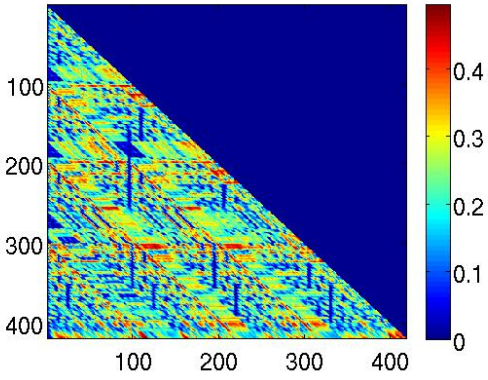


Figure 2.5: Time lag for the audio song used in Figure 2.3

in time (lag).

$$R(i, i-j) = D(i, j) \tag{2.2}$$

for $i - j > 0$.

This transformation throws-out the duplicate information from the symmetric distance matrix. The diagonal stripes in distance matrix representing repeated sequences, appears as vertical lines in time-lag representation. In this representation, stripe information is transformed into easily interpretable form, whereas block information may be difficult to extract as they are transformed into parallelograms. Moreover, this representation works only in the case where repeating parts occur in the same tempo.

3. IMPLEMENTATION

In previous section, we have gained some terminology related to music and some signal processing algorithms that are used in music information retrieval. In order to realize music information retrieval process, we have created a simple system. In this section we describe the overview of the system, its different components and its working procedures.

3.1 System Overview

We would like to construct a system that is capable of matching query feature sequence of an audio file against database feature sequences of MIDI files. The general idea of our system is presented in Figure 3.1. The system architecture consists of four main blocks. Synthesizer block is responsible for synthesizing a MIDI file to an audio signal. Feature Extraction Block is responsible for extracting features from the music piece. Database stores all the extracted features along with other relevant information of music. The matching block performs comparison between features and returns the result. In an indirect way, this system can be used to transcribe the given piece of music from audio format to score/MIDI format. Having a segment of query audio, transcription process is accomplished by finding the corresponding MIDI from the database. The functionality of each module is presented below.

3.1.1 Synthesizer

As we mentioned earlier, MIDI is not actual recording of music, it is rather a spreadsheet like set of instruction used to create music and it uses about thousand times less space than the corresponding CD quality audio recording. So it is preferable to use MIDI format for the database items. In order to extract features, for comparisons, we first convert MIDI to audio format. This block is mainly responsible for converting higher level music representation such as MIDI files to low-level music representation such as audio file. This block uses 'Timidity++' software to synthesize MIDI file to uncompressed audio format .wav file. Timidity++ is a software synthesizer, capable of playing MIDI files without any hardware synthesizer. It is a free software released under GNU General Public License. It is a cross platform system written in C language. The audio signal generated from this block acts as

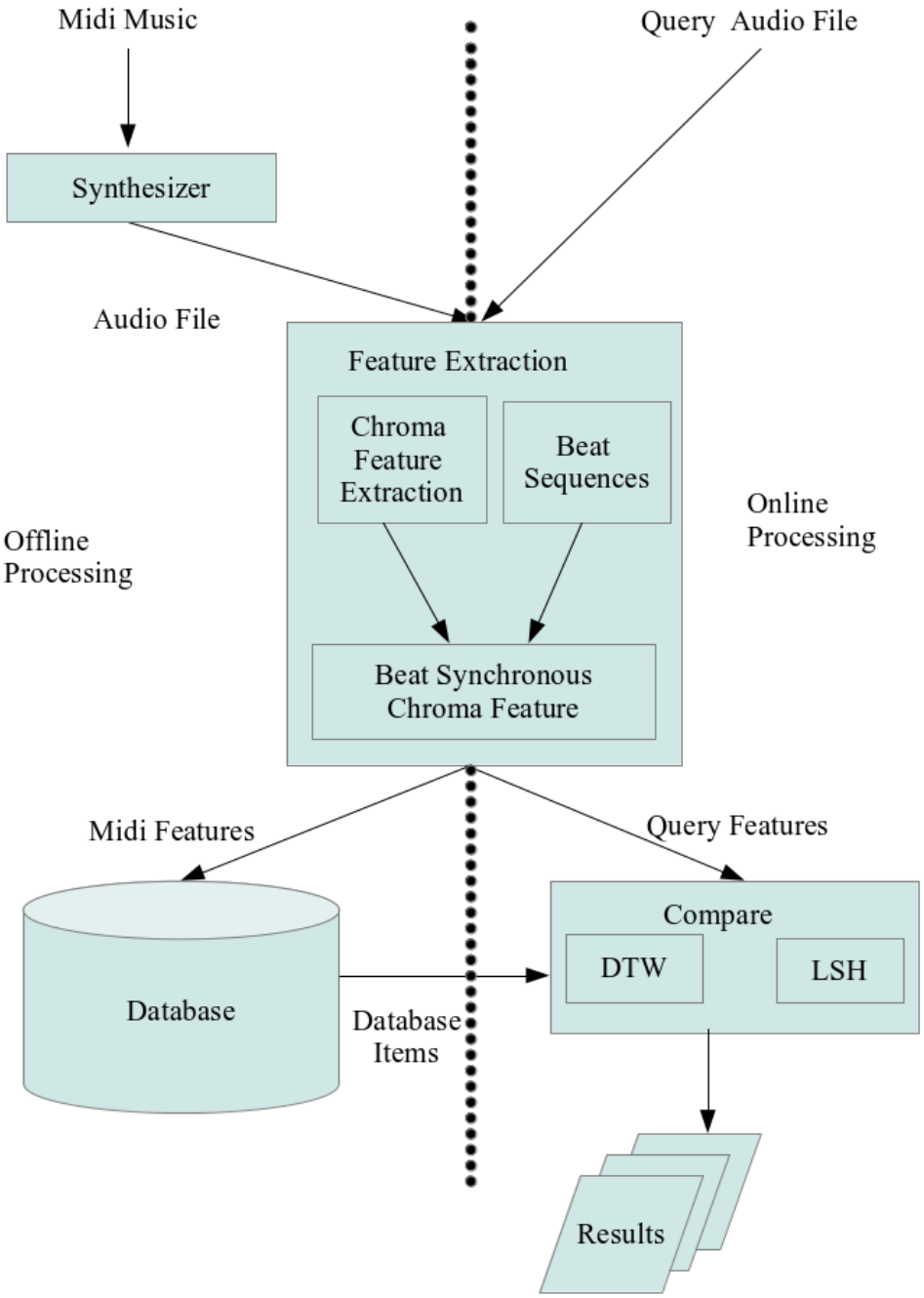


Figure 3.1: System Overview.

an input to the feature extraction block. As seen in Figure 3.1, only database items (MIDI files) are passed through this block because we are using audio files to query, hence no need of synthesizer for query files.

Table 3.1: Important MATLAB functions in chroma toolbox.

Function Name	**Description**
wav_to_audio	Takes wave file and convert it to audio format.
estimateTuning	Takes audio data and estimate shift parameter.
audio_to_pitch_via_Fb	Extract pitch feature from audio data.
pitch_to_chroma	Derive chroma feature from pitch feature.
pitch_to_CENS	Derive CENS feature from pitch feature.
smoothDownsampleFeature	Smoothing and downsampling of chroma feature.
normalizeFeature	Normalization of chroma feature.

3.1.2 Feature Extraction

Once we have an audio signal for a database or a query item, next step is to extract features describing musical content of the signal. We have already mentioned importance of feature extraction for matching purpose. We are mainly interested in chroma features and beat synchronous chroma features. Chroma features capture both melodic information and harmonic information and variation in tempo is normalized by beat synchronous feature extraction [12]. Chroma feature of an audio file is extracted by using chroma toolbox released under GNU-GPL license, see [49].

For beat synchronous chroma features, we first need to extract chroma feature in short frames as mentioned earlier. Table 3.1 gives the overview of important MATLAB functions used from chroma toolbox. Extraction of chroma features starts by calling function *wav_to_audio*, which converts input WAV file to audio data with a sampling rate of 22050 Hz. In the next step audio data is passed to a function *estimatetuning*, which estimates an appropriate filter bank shift for the music segment. In next step, pitch features are computed by calling *audio_to_pitch_via_FB* function, setting window length to 4410 samples and window overlap of 50% of window length. Here using window of 4410 samples with a sampling rate 22050 Hz, gives the window of 200 ms of audio. Now next step is to compute CP (Chroma-Pitch) and CENS (Chroma Energy Normalized Statistics) features, variants of chroma features described in chapter 4, by passing pitch features to functions *pitch_to_chroma* and *pitch_to_CENS* respectively. Both of those functions call function *smoothDownsampleFeature* for smoothing and downsampling and function *normalizeFeature* for normalization of the features. Chroma feature extraction is described in more detail in section 4.1.

After successful extraction of chroma features, next step is to extract the beat sequence of a musical file using the system explained in [2]. In this project, beat tracking is done applying two-pass beat tracking method, which means first beat tracking was performed with default parameters. After than compute the tempo as T = median(diff(beatTimes)) which is the median of inter-beat-intervals from

the first pass (typically around 0.5 seconds). Then the beat tracker parameters are changed by setting myBeat = T (tempo computed above) and sigmaBeat = 0.05 (allowing less variance around the median tempo). And finally run beat tracker again for same input audio file but now with updated parameters. This process is done independently for each file. This mechanism makes sure there is no tempo fluctuation that would break down the synchronization process.

Once we have chroma and beat sequence, we combine them to get beat synchronous chroma feature. The goal is to have one feature vector per beat. It is done by averaging the chroma features between two beat times. Such a representation helps to normalize the tempo variation. Derivation of beat synchronous chroma Feature starts with calling function $PerformBeatSyncChroma$, which takes chroma and beat features as input and returns the beat synchronous chroma feature. All chroma features between the i^{th} and $(i+1)^{th}$ beat are averaged.

Sometime there occur a situation when beat tracker do not perform well, meaning producing wrong beat sequences. Sometime beat tracker may miss the beat times and or sometime they may introduce extra beat times. In order to encounter such situation we derived different variety of beat features.

- Upsampled: In this variant, we upsample the original beat sequences by 2, meaning one extra beat time is inserted between each original beat sequences. After Upsampling beat sequences we then calculate beat synchronous chroma feature. The length of feature vector is this variant is double as compared to original beat synchronous chroma feature.

- Downsample: In this variant, we down sample the original beat sequences by 2, meaning we keep every odd beat time. After down sampling beat sequences we then calculate beat synchronous chroma feature. Here we halved the feature vector as compared to original beat synchronous chroma feature.

- Threshold based: In this variant, we first compute tempo of the music and compare this tempo value with given threshold (110 in our case). If tempo is less than given threshold value we upsample the beat sequence else we downsample the beat sequence and then compute beat synchronous chroma feature.

3.1.3 Database

Once features have been extracted from a database item (MIDI file), they are stored in a database because they are used in future during matching against query item. Since we are interested with chroma features and beat synchronous chroma features, we need to store them in the database. So we create a struct which contains chroma

Table 3.2: Important MATLAB functions for DTW matching.

Function Name	**Description**
PerformMatching	Main function, for matching query against database.
GetDistance	Compute cosine distance between two feature vectors.
DynamicTimeWarping	Performs Dynamic Time Warping.
StaticTimeWarping	Strictly follow diagonal path.

features, beat features and beat synchronous chroma features along with other information related to features and the musical files. For this purpose we create a function *addToDatabase*, which takes the input parameters as struct of features along with other relevant information and then store them in the database.

Although it is not necessary to store the feature of query items (.wav), we have also saved the feature of a query item also. Because we can directly use the stored information for further processing this in turn saves time of feature extraction. The format and procedures for storing query information is similar to that of database items as mentioned above.

3.1.4 Matching

This block is responsible for performing matching between query feature and database feature. Once the features of a query file are extracted, then they are matched against the database items for similarity. This block uses either of two algorithms for similarity measure, Dynamic Time Warping or Locality Sensitive Hashing. Once the matching is done using either of algorithms the N nearest matches to the query file are retrieved as a result.

DTW

For matching using Dynamic Time Warping, Table 3.2 gives the overview of functions required. We first need to call function *PerformMatching* with parameter like QueryFeature, DatabaseFeatures and Weight vector for DTW paths. For all items in the database, this function first calls *GetDistance* functions and then call *DynamicTimeWarping* function. *GetDistance* takes two arguments QueryFeature and i^{th} database feature and then returns the cosine distance matrix between them. This matrix is passed to *DynamicTimeWarping* along with Weight vector for path, which returns the cost of moving from starting position to the end position. Such cost is calculated against every database item and the database item with minimum cost is considered as the closest match to the query item. Not always the item suggested by this algorithm is a perfect match to the query. So, this function also calculates the rank of the query item based on the cost value we obtained above. If

Table 3.3: Important MATLAB functions for LSH matching.

Function Name	**Description**
LSHMatching	Main function, for matching query against database.
lsh	Initialize and index LSH data structure.
lshlookup	Perform matching and return n nearest neighbor.

rank is 1 then there is a perfect match between query item and the nearest item suggested by the algorithm. If rank is 2, then the correct match to the query item have second minimum cost value and so on. We can find the efficiency of the algorithm as the ratio of correct match to the total item queried.

In order to favor horizontal, vertical and diagonal direction during the alignment process, we introduce a cost for each path W_{hor}, W_{ver}, W_{dia} for horizontal, vertical and diagonal paths respectively. For example, if you wish to favor diagonal path than the cost of horizontal path W_{hor} and vertical path W_{ver} should be greater than the cost of diagonal path W_{dia}. Since during alignment process, we want DTW path not to deviate more from diagonal path so generally value of W_{dia} is less as compared to the value of W_{hor} and W_{ver}. But too much variation on those costs may also lead us to unfavorable results. One should be sensible while choosing cost for each path.

If we want to find the cost of moving from starting point to end point strictly following the diagonal path then we can use *StaticTimeWarping* function or just use weight that give infinite cost to paths other than horizontal path.

LSH

The main idea of LSH is to project the high-dimensional data into low-dimensional space, where each point is mapped to a k-bit vector, called as hash key. Similar input items are mapped to the same bucket with high probability. Table 3.3 gives the overview of LSH based matching. LSH based matching starts by calling function LSHMatching with parameters like QueryFeature, Database. This function calls another function lsh which initialize the LSH data structure. This function will just index the data; it will not store it. You need to have original data so that you can go back to actual data from indices. Once the indexing of database is performed, this function then calls another function lshlookup, which performs the actual comparison between query item and then database items and returns the n nearest neighbor. In the case of DTW, we can directly use the feature vectors extracted from feature extraction block, but in case of LSH, one should first concatenate N 12-long feature vectors into one 12N-long feature vector. LSH toolbox used in this thesis is available at [67]. This toolbox is maintained by Assistant Professor Greg Shakhnarovich at Toyota Technological Institute - Chicago. LSH is described in more detail in chapter 6.

3.2 DataSet

In order to perform content based information retrieval experiments, one needs to have a set of data. Researchers can apply different algorithms and methodology to the same set of data and compare the results between them. For this thesis work we perform an experiment on two types of database, RWC (Real World Computing) database and TUT database, collected and maintained by audio research group of Tampere University of Technology.

3.2.1 Real World Computing (RWC)

RWC music database is music database that is available for researcher as a common foundation. This is considered as the world's first large scale copyright-cleared music database. This database was created by RWC Music Database Sub working Group of the Real World Computing Partnership (RWCP) of Japan. Japan's National Institute of Advance Industrial Science and Technology (AIST) holds all the copyright and legal issues related to this database and is distributed among researchers at minimum cost which covers the shipping, duplication and handling chargers. The entire music file in the database has its corresponding MIDI files and text files for lyrics. This database contains Popular Music Database (100 pieces), Jazz Music Database (50 pieces), Classical Music Database (50 Pieces), Royalty-free Database (15 Pieces), see [39; 66] for further information.

In this thesis work, we are concerned about 100 pieces of popular database among which 20 songs were performed with English lyrics whereas 80 songs were performed with Japanese lyrics. 148 people were involved in recording those songs including 30 lyrics writer, 23 arrangers, 25 composers and 34 singers. In order to maintain balance between male-female singers, 15 male singers sung 50 songs, 13 female singers sung 44 songs and 6 vocal group sung 6 songs.

3.2.2 TUT Dataset

Apart from RWC database, we have also created our own database named as TUT database. In RWC database, the audio (query item) and its corresponding MIDI version (database item) are structurally same. But in real world situation, that is often not the case. In real world situation we have a different version of the same song like cover songs, remix songs and many more with structural differences. Often we come across a situation where we need to compare two structurally different songs, for example, we need to query with the remix version of the songs and database contain its corresponding original songs.

To handle such a situation we decided to collect MIDI and audio of popular songs from a different source and this dataset is termed as TUT database. It contains 290

popular audio songs and a MIDI version of those songs. This database was collected and maintained by research assistants Tuomo Tuunanen and Jani Mäkinen. MIDI format was obtained from midimusic whereas audio was obtained from itunes.

4. FEATURE EXTRACTION

Automatic music processing faces lots of difficulties because music data are complex and diverse. For musical data we need to take care of various aspects such as instruments (drums, guitar, voice, piano), data format (score, audio, MIDI) and other parameters like dynamics, tempo, etc. In order to analyse and access the musical data, one should extract suitable features that capture the important aspects of data leaving out irrelevant details.

In recognition and retrieval tasks, feature extraction is a special form of reducing data dimensionality. When we have an input item that is too large to be processed, then such item will be transformed into a reduced set of feature (also known as feature vectors) which are relevant to our task, leaving out irrelevant information (noise). Such process of transforming large input item into the set of features is known as feature extraction. Feature extraction process reduces the amount of resource required to describe the item (music). Analysis of a music signal with extracted feature will allow more reliable modelling / learning / matching, etc. as irrelevant (noisy) information is suppressed.

One should be careful while choosing which feature to extract. Not all features are important, so one should analyse the relevant feature for the task. Extraction of irrelevant features will unnecessarily increase the processing time and may result into wrong result. For this thesis work, we concentrate only on chroma features and beat sequences. And later we combine both chroma feature and beat sequences to get beat synchronous chroma features.

4.1 Chroma Feature Extraction

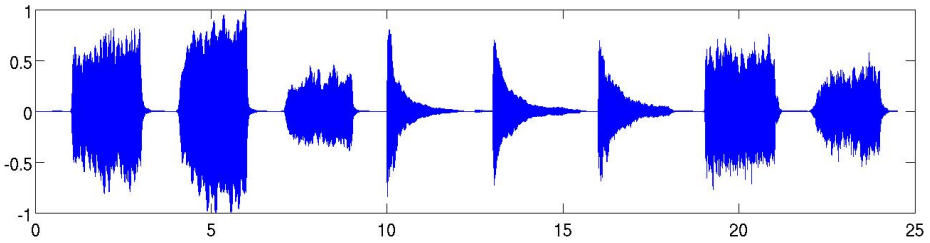
Chroma based audio features are well known in the field of musical signal processing and analysis. It is well known that human perception of pitch is periodic in a way that two pitches differing by an octave are perceived as harmonically similar. Because of this property of pitch, we can separate pitch into tone height and chroma. Music audio can be represented by using chroma features in which the entire spectrum is divided into 12 bins each bin representing the distinct chroma a musical octave. Thus, we can represent a chroma feature of a music by a 12 dimensional vector $x = (x(1), x(2), x(3), \ldots, x(12))^T$, where $x(1)$ corresponds to chroma C, $x(2)$ to chroma C#, $x(3)$ to chroma D, and so on. Each sequence of chroma features

expresses the short time energy distribution over twelve chroma bands. Chroma features are robust to variation in timbre and are closely related to harmony. As a result, chroma-based audio feature also known as pitch class profile are used in music processing and analysis. Chroma-based features are used in chord recognition, music synchronization and alignment as well as structure analysis of audio signals. In the case of content based music/audio retrieval such as audio matching and cover song identification, chroma features are considered as powerful mid-level feature representation. There are different ways of computing chroma feature of an audio. It can be either computed by using short-time Fourier transform (STFT) with binning strategies [40] or by using multirate filter banks [46]. We can also introduce some pre-processing and post-processing steps in order to alter spectral, temporal and dynamic aspect which leads to numbers of different variant of chroma features.

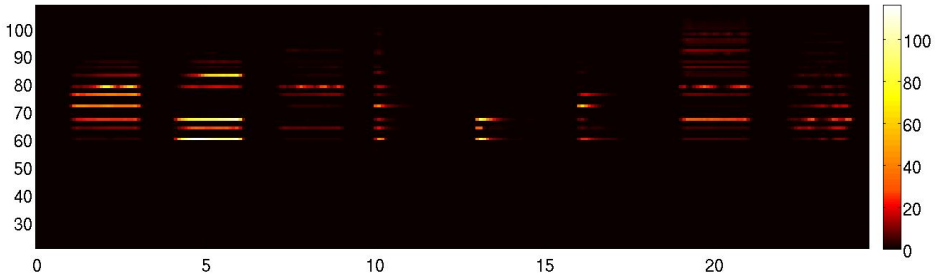
In order to extract chroma feature of a music piece, this thesis uses the chroma toolbox released under GNU-GPL license, see [49]. This toolbox contains the matlab implementation and introduces pre-processing and post-processing steps to modify spectral, temporal and dynamic aspect which gives us the different variants of chroma feature named as Pitch, CP, CLP, CENS and CRP. This thesis work concentrates on CP and CENS features. During feature extraction process, an audio signal is decomposed into 88 frequency bands corresponding to pitches $A0$ to $C8$. For sufficient spectral resolution for lower frequencies, authors use constant Q multirate filter bank using a sampling rate of 882 Hz for low pitches, 4410 Hz for medium pitches and 22050 Hz for high pitches see [46] for details. And then using a fixed length window with 50% overlap, short-time mean-square power is computed for each 88 pitch subbands. By using a window of length 200 milliseconds will give the features at the rate of 10 Hz (10 features per second). The obtained feature measures the short-time energy content of an audio signal within each subbands, and this feature is termed as Pitch.

In order to account for global tuning of an audio segment, the center frequencies of the sub band-filters of the multirate filter bank need to be shifted properly. At this point, average spectrogram vector is computed and estimate for filterbank shift is estimated using weighted binning techniques similar to [15]. From six different pre-computed multirate filter banks (available in toolbox) corresponding to a shift of $\sigma \in \{0, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}\}$ semitones, the most suitable one is chosen according to the estimated tuning deviation.

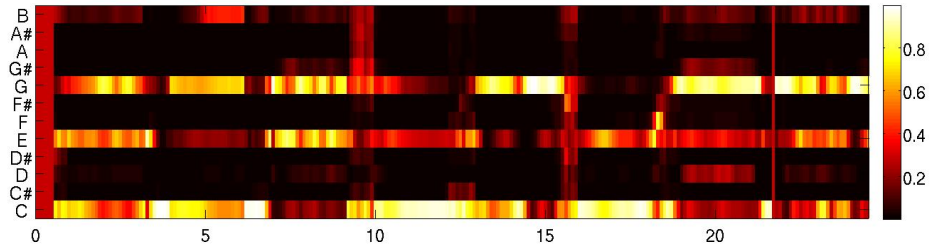
Once we have the pitch features, we can compute chroma features by adding the corresponding values that belong to same chroma.The resulting 12 dimensional vector $x = (x(1), x(2), x(3), \ldots, x(12))^T$, is known as Chroma-Pitch and is denoted by CP, where $x(1)$ represents chroma C, $x(2)$ represents chroma C#, $x(3)$ represents



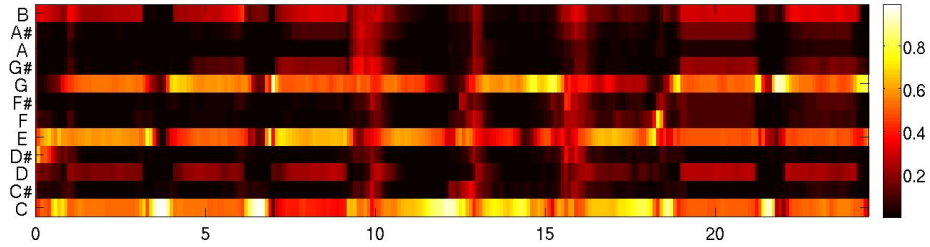
(a) Waveform



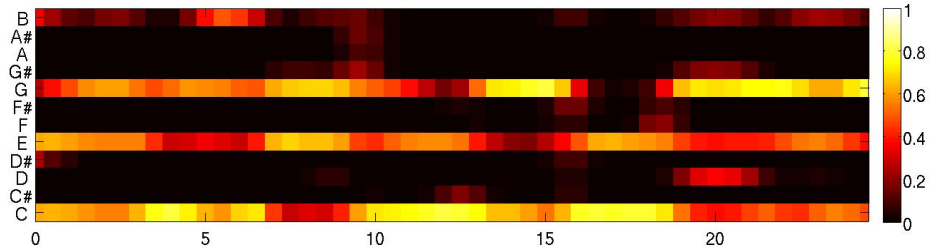
(b) Pitch



(c) CP Normalized



(d) CLP



(e) CENS

Figure 4.1: Various feature representation of an audio.

chroma D, and so on.

In order to achieve invariance, chroma feature can be normalized to suitable ℓ^p-norm defined by $\|x\|_p = (\sum_{i=1}^{12} |x(i)|^p)^{1/p}$, for some natural number $p \in \mathbb{N}$ and for a given chroma vector $x = (x(1), x(2), x(3), \ldots, x(12))^T$. In this thesis work, chroma feature is normalized with respect to Euclidean norm i.e., $p = 2$.

Sometimes it is important to represent the sound intensity on a logarithmic scale. For this purpose, we can apply the logarithmic compression to the energy while generating audio pitch features. If e be the energy value of a pitch representation and η be the suitable positive constant known as compression parameter, logarithmic compression can be achieved by replacing the value of e with $log(\eta.e + 1)$. Then chroma features are computed as explained above. The computed feature is known as Chroma-Log-Pitch and is represented by CLP$[\eta]$.

Another variant of chroma feature known as Chroma Energy Normalized Statistics (CENS) contains robust and scalable audio features. This feature is widely used in audio matching and retrieval systems see [16; 45]. In order to compute CENS feature, all chroma vector is normalized to ℓ^1-norm and quantized to chosen threshold. In the next step, features are smoothed by using window of suitable length $w \in \mathbb{N}$ and then features are downsampled by factor d. Finally the downsampled features are again normalized to ℓ^2-norm.

Another variant of chroma feature provided by chroma toolbox is known as Chroma DCT-Reduced log Pitch (CRP), which is invariant to timbre. Given a pitch vector, we first perform logarithmic compression and then apply DCT to the compressed pitch vector and keep only the higher coefficient of the resulting spectrum. In the next steps we apply IDCT and finally convert the resulting pitch vector into 12 dimensional chroma features which is then normalized by Euclidean norm.

Among those chroma variants, this thesis uses the CP and CENS feature, which captures harmonic and melodic information. Both variants are robust against the variations of musical properties like timbre, dynamics, articulation, execution of note groups, and temporal micro-deviations [51]. CENS features, which was introduced first in [45], are strongly correlated to short-time harmonic content of the audio signal while absorbing a variation in other parameters. CENS feature can be processed efficiently because of their low temporal resolutions, see [45; 46] for details.

4.2 Beat Feature Extraction

When you listen to your favorite songs, you are usually able to tap your foot in time to the music. It is quite automatic and unconscious human response to the music. Unfortunately, it is not easy for computers. The computational equivalent to this behavior is termed as a beat tracking. Beat tracking involves estimating the beat location (i.e., where you would tap your foot). In engineering terms, it

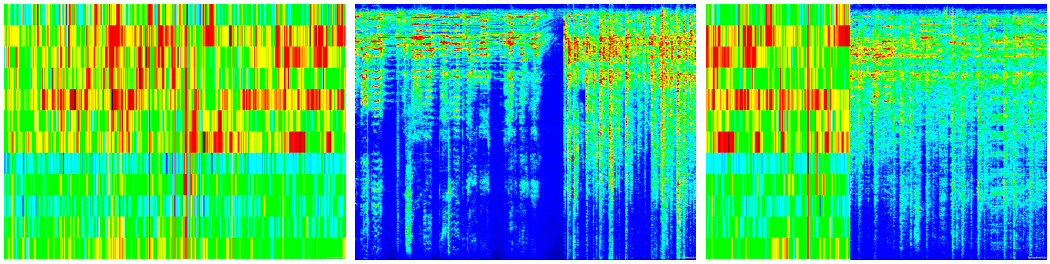
is the frequency and phase of a beat signal, phase of which is zero at the location of each beats. The main purpose of beat tracking system is to compute the set of beat times that would exactly match with the foot taps of trained human musicians. The tempo of a musical piece can be defined as the rate of the musical beat and is measured in beats per minute. In case of music information retrieval based research, beat sequences can give you the musically significant temporal segments for additional analysis; such as rhythmic genre classification [58], long-term structural segmentation of audio [40; 41], chord estimation for harmonic description [26].

Beat tracking with the help of computers is an active area of research. Most of the earlier algorithms for beat tracking used symbolic data or MIDI rather than audio signals. It might be due to inadequate computational resources or may be due to difficulty in note onset detection for early computers. However, recent progress on computational power, audio feature extraction and onset detection has enabled the use of audio signals. There are many beat tracking algorithms proposed by the researchers. For example, Goto [37; 38] proposed a real-time beat tracking system for music.

Dixon beat tracking system [57; 59], has two major components, that performs tempo induction and beat tracking respectively. First digital audio input is preprocessed by an onset detection stage. The list of onset times given by this stage is fed into the tempo induction module. This component examines the time between pairs of note onset, and then performs clustering to find significant cluster of inter-onset intervals. Each cluster representing a hypothetical tempo. These clusters are ranked such that most salient cluster is ranked most highly. The output of tempo induction module, which is ranked list of a tempo hypothesis, together with event times are input to the beat tracking module. Those hypotheses are tested by agent architecture about the rate and timing of beats. And then give the output as beat times found by highest ranked agent.

Ellis describe a beat tracking system [11], which at first estimate a global tempo. This global tempo is used to construct a transition cost function. And then dynamic programming is used to find best-scoring set of beat times that reflect the tempo.

There are many beat sequence extraction algorithms proposed by different researchers. Some state-of-art algorithms were compared by [17], and it was shown that the approach [2] was most accurate. This thesis uses the methods suggested by [2]. In his approach, there are mainly three important entities: time-frequency analysis, comb filter resonators and probabilistic model for pulse periods. In time-frequency analysis part, his technique measures the degree of accentuation in a music signal. This technique is robust to diverse acoustic (music) materials. A registral accents is used as an input in order to generate time instants of meter (tactus, measures, tatums). Periodic analysis of registral accents is performed by using bank of



(a) Chroma Feature. (b) Spectrogram. (c) Real time switching between chroma and spectrogram.

Figure 4.2: Mobile application displaying acoustic features.

comb-filter resonators. Comb filter resonator is used to extract feature for estimating pulse period and phase. Here filter bank estimates the periods (time duration between two successive beats) and the phase (time distance from the start of the performance) at mentioned three time scales. Lastly probabilistic model is used to estimate tatum, period-length of tactus, and pulse and temporal continuity. For each time instance, first estimation of period of pulse is performed which is used as input to the phase model. The prior musical knowledge is encoded by probabilistic model that lead to reliable and temporally stable meter/beat tracking. Meter estimation can be done in both causal and non-causal way. Causal implementation requires more time to become steady. In this thesis work, we employed non-causal implementation. For more detail, see [2].

4.3 Beat Synchronous Chroma Features

This representation has two features, beat sequence and chroma feature. The goal of beat synchronous chroma features is to have one feature vector per beat. By using this representation, one can normalize the variation on tempo. It is done by averaging all the chroma features between any two beats. It can also be viewed as average power between two beats. Beat synchronous chroma feature was used by Ellis [10] for cover song identification. In this thesis we use CP variant of chroma feature along with beat sequences to compute beat synchronous chroma (BSC) feature.

4.4 Visualizing Acoustic Features in Real Time On a Mobile Device

For this thesis work, we decided to develop a mobile application that can record the audio from its surroundings and display the acoustic features in real time. For this purpose, we choose Unity3D platform to develop our application and deploy it into mobile device. Our application is capable of recording audio through the

built-in microphone of mobile device at sampling frequency of 44100 and displays its spectrum on mobile device in real time. In order to calculate spectrum data, we use inbuilt function GetSpectrumData, which takes two important parameters like numSamples (number of samples) and FFTWindow. GetSpectrumData returns a block of currently playing source's spectrum data. By default, our application uses rectangular window with 1024 samples. User interface allows you to change some parameters. You can easily switch between chroma feature and spectrogram. Number of samples must be a power of 2 with a minimum value of 64 and maximum value of 8192. We use window to reduce the leakage between frequency bands. The more complex window type, better the quality but at the cost of processing speed. Figure 4.2 shows the screenshot of mobile device displaying acoustic features.

5. DYNAMIC TIME WARPING

In this chapter, we will first go through the basic concepts related to dynamic time warping and then we will highlight the principle and concept of DTW. Before stepping into DTW, we first talk about time alignment and normalization, dynamic programming and time-normalization constraints. In this section, we discuss DTW algorithm and its implementation.

5.1 Time Alignment and normalization

In speech recognition, variation in the speaking rate and duration contribute to the dissimilarity of the same utterance. Such variation in the rate should not contribute in dissimilarity of the same utterance. Thus, there is a need of normalization in the utterance before making any meaningful comparison. Same is the case with music as well, often same song is sung by different artist at different speed and tempo. Such variation in the singing rate and tempo should not contribute in dissimilarity of the same piece of music.

One of the best solutions to such problem is to find the best alignment between a pair of music sequences. Here finding the best alignment means to find the best path through the distance matrix that maps the feature sequences from one song pattern to the feature sequences from another song pattern. In order to find the best path, one should solve a minimization problem to calculate the dissimilarity between two feature patterns. Simple solution to time alignment is linear time alignment technique.

Let us suppose $X = (x_1, x_2, x_3, \ldots, x_{T_x})$ and $Y = (y_1, y_2, y_3, \ldots, y_{T_y})$ be the feature sequences of two songs respectively. Let the time indices of X and Y be denoted by i_x and i_y respectively and Tx and Ty need not to be identical. The dissimilarity between X and Y is denoted by $d(i_x, i_y)$, where $i_x = 1, 2, 3, \ldots, T_x$ and $i_y = 1, 2, 3, \ldots, T_y$.

In linear time normalization, the dissimilarity between X and Y is defined by

$$d(x, y) = \sum_{i_y=1}^{T_y} d(i_x, i_y) \tag{5.1}$$

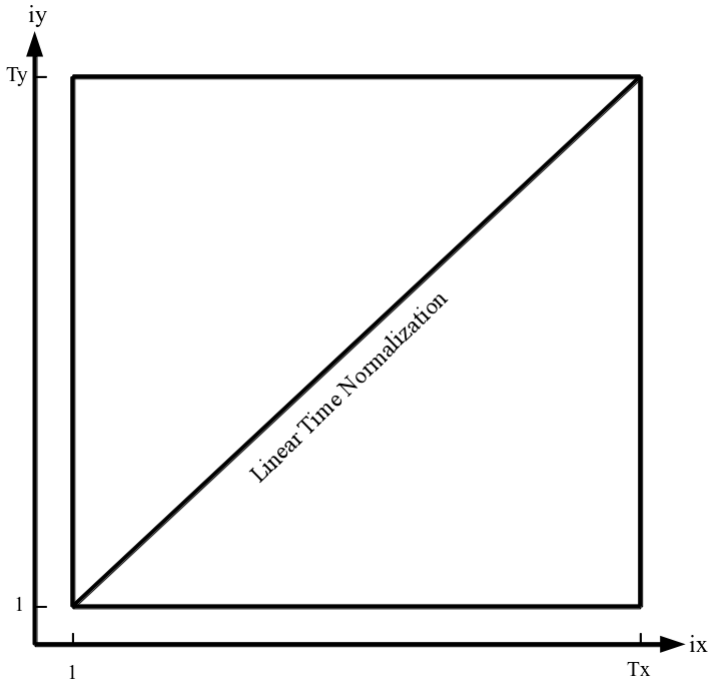


Figure 5.1: An Example of Linear time alignment for two sequence of different time duration.

where

$$i_y = \frac{T_y}{T_x} i_x \tag{5.2}$$

Here, round-off rule is applied in equation 5.2 because i_x and i_y both are integer. Here according to the direction of time normalization the summation in equation 5.1 can be done i_x ranging from 1 to T_x.

Linear time alignment assumes that the variation in tempo of the song is proportional to the musical sound and is independent to music being produced. Thus, one can assume that measurement of distortion takes place along the straight diagonal line of a rectangle in a (i_x, i_y) plane as shown in the Figure 5.1. The points (i_x, i_y) along the diagonal line represents the dissimilarity (distance) between the feature vector X and Y at index i_x and i_y. Clearly this does not effectively model the simulation for real music.

Another general approach for time alignment and normalization is to use two wrapping functions ϕ_x and ϕ_y. Those two wrapping function maps the indices of two music features i_x, i_y to the common time axis k.

$$i_x = \phi_x(k), k = 1, 2, 3, \ldots, T \tag{5.3}$$

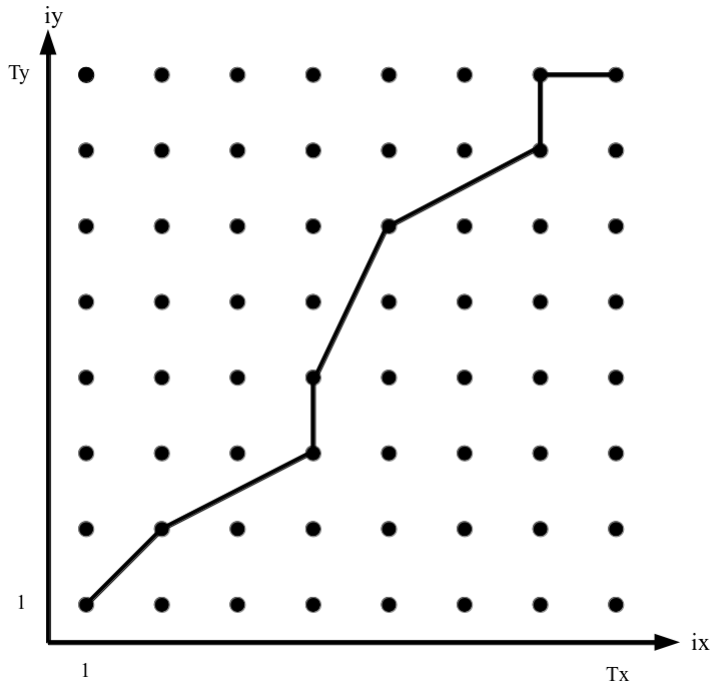


Figure 5.2: An example of time normalization of two sequence where individual time index i_x and i_y are mapped to common time index k.

and

$$i_y = \phi_y(k), k = 1, 2, 3, \ldots, T \tag{5.4}$$

Those two wrapping function pair $\phi = (\phi_x, \phi_y)$ are used to define the global pattern dissimilarity measure $d_\phi(X, Y)$ as the sum of distortion over whole utterance as

$$d_\phi(X, Y) = \sum_{k=1}^{T} d\left(\phi_x(k), \phi_y(k)\right) m(k)/M_\phi \tag{5.5}$$

where $d(\phi_x(k), \phi_y(k))$ is again dissimilarity defined for $x_{\phi_x(k)}$ and $y_{\phi_y(k)}$, $m(k)$ is a non-negative path weighing coefficient and M_ϕ is a path normalization factor. Figure 5.2 shows the example to general time normalization scheme. The solid line in the grid is the path along which the $d_\phi(X, Y)$ is evaluated.

Now we need to specify the path $\phi = (\phi_x, \phi_y)$ as indicated by above equation. The main issue is to choose the appropriate path from an extremely large number of possible pairs of wrapping functions, in order to measure overall path dissimilarity with consistency. Sakoe and Chiba [19] suggest to choose the minimum of $d_\phi(X, Y)$, over all possible path to define dissimilarity $d(X, Y)$ such that

$$d(X, Y) \triangleq \min_{\phi} d_\phi(X, Y) \tag{5.6}$$

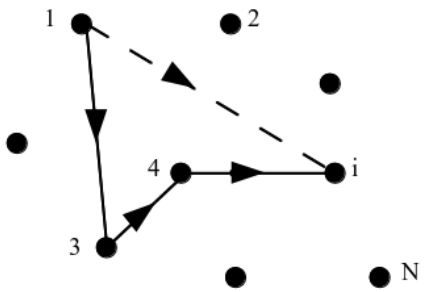


Figure 5.3: An example of optimal path problem, finding minimum cost for moving from point 1 to i in as many moves as required.

For performance of the same piece of music, the above equation measures the dissimilarity based on the best probable alignment in order to compensate for the tempo difference. In order to find the best path, one can use dynamic programming.

Now let us analyze the time complexity. Let M and N be the length two sequences. As a result, our dissimilarity matrix will be of size MxN. Filling the each cell of the matrix with dissimilarity value is constant time. Thus, filling entire matrix is $O(MN)$. Calculating path parameters in all MxN points is also $O(MN)$. Hence overall time complexity is $O(MN)$. Its space complexity is also $O(MN)$.

5.2 Dynamic Programming

Dynamic programming is widely used method for solving sequential decision problem. To show its applicability in music matching, let us discuss two problems (particularly related with time alignment and normalization) that are solved by dynamic programming.

First problem is optimal path problem. Let us suppose we have a set of N points. Let $\zeta(i, j)$ be the non-negative cost of moving from i^{th} point to j^{th} point in a single step. Now problem is to find the minimum cost for moving from any point, say 1, to another point, say i, using as many steps as required. This problem is shown in Figure 5.3, where cost of moving from point 1 to i is minimum if we go through intermediate points 3 and 4, $\zeta(1, i) > \zeta(1, 3) + \zeta(3, 4) + \zeta(4, i)$. Using conventional terms, let 'policy' be the term used to determine the cost and the sequence of points to traverse from start point, say 1, to end point, say i. Now main issue is to determine the policy with minimum cost for moving from point 1 to point i. Let this cost is denoted by $\varphi(1, i)$.

Let us first consider $\varphi(1, j)$ be the cost of moving from initial point 1 to optimal intermediate point j using as many steps as required and then $\zeta(j, i)$ be cost of

moving from optimal intermediate point j to point i in a single step, then following equation holds

$$\varphi(1,i) = \min_{j}[\varphi(1,j), \zeta(j,i)] \tag{5.7}$$

Then optimal sequence of moves with minimum cost from any point i to any other point j with as many steps as necessary with intermediate point ℓ can be written as

$$\varphi(i,j) = \min_{\ell}[\varphi(i,\ell), \varphi(\ell,j)] \tag{5.8}$$

This equation tells that all the consecutive sequences of movement from point i to j must also be optimal, and the intermediate point ℓ must also be an optimal point linking the optimal partial sequences of move before and after that point.

The second problem is synchronous sequential decision problem. Now next step is to find the optimal number of moves, say M, starting from point i to point j with minimum cost $\varphi_M(i,j)$. Starting from point i, after m^{th} step, $(m < M)$, path can reach any point ℓ, $\ell = 1, 2, 3, \ldots, N$, with minimum cost of $\varphi_m(i,\ell)$. Assume in $(m+1)^{th}$ move, we reach point n, then

$$\varphi_{m+1}(i,n) = \min_{\ell}[\varphi_m(i,\ell) + \zeta(\ell,n)] \tag{5.9}$$

Equation (5.9) allows us to search optimal path incrementally, in a progressive manner and only best moves are considered.

Now let us analyze the time complexity. Let M and N be the length of two sequences. As a result, our dissimilarity matrix will be of size MxN. The initialization step is performed only in the first row and column of a matrix; hence time complexity for initialization step is $O(M+N)$. Filling the each cell of the matrix with cost of reaching that cell from the initial point involves evaluating its neighbor cells and updating cell with optimal value, which is constant time. Thus, filling up entire matrix is $O(MN)$. Finally during backtracking we move maximum of $M+N$ steps, $O(M+N)$. Hence overall time complexity is $O(M+N) + O(MN) + O(M+N) = O(MN)$.

5.3 Time-Normalization Constraints

For meaningful alignment process in terms of time normalization, we require some constraints on warping functions. Unconstraint minimization in equation (5.6) can result in a close match between two different utterances resulting in meaningless comparison. Some necessary and reasonable warping constraints for time alignment include:

- endpoint constraint

- monotonicity conditions
- local continuity constraints
- path constraints
- slope weighting

5.3.1 Endpoint Constraints

Usually endpoints mark beginning and ending points of the music signal. Endpoints of a music signal are considered to be known prior, and all the temporal variation occurs in the range of known endpoints. In case of time normalization, end points provides the limits to the music signal, giving a set of constraints for warping functions

$$\phi_x(1) = 1, \phi_y(1) = 1 \tag{5.10}$$

$$\phi_x(T) = T_x, \phi_y(T) = T_y. \tag{5.11}$$

5.3.2 Monotonicity Conditions

The temporal order of the spectral sequence in a music signal is important. In order to maintain the order for time normalization, monotonicity constraints is imposed in a form

$$\phi_x(k+1) \geq \phi_x(k) \tag{5.12}$$

$$\phi_y(k+1) \geq \phi_y(k) \tag{5.13}$$

This constraint implies that the path along which $d_\phi(X, Y)$ is evaluated will never have a negative slope, which eliminates the possibility of time reverse warping along the time axis.

5.3.3 Local Continuity Constraints

The main purpose of time normalization is to find best temporal match as defined in equation (5.6), so we should not omit any sound segment containing important information. To minimize the risk of important information loss, a set of local continuity constraints is included on the warping function. Such constraint can take many forms, one of them suggested by Sakoe and Chiba [19] is

$$\phi_x(k+1) - \phi_x(k) \leq 1 \tag{5.14}$$

$$\phi_y(k+1) - \phi_y(k) \leq 1 \tag{5.15}$$

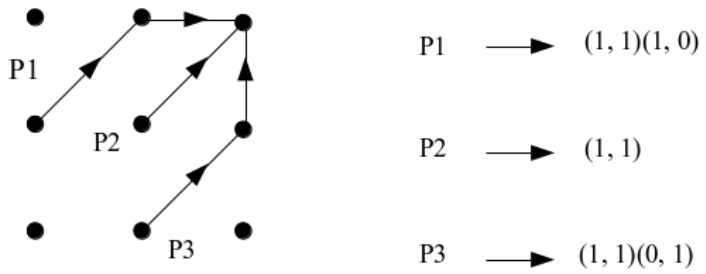


Figure 5.4: An example of local continuity constraints.

Sometimes such constraints get quite complicated; hence it is easy to express them in term of incremental path change. Let path $\mathcal{P}$ be a sequence of moves, each move is specified by a coordinate pairs,

$$\mathcal{P} \rightarrow (p_1, q_1)(p_2, q_2)(p_3, q_3) \ldots (p_T, q_T) \tag{5.16}$$

where subscript is in term of time k. Figure 5.4 illustrates three paths $\mathcal{P}_1 \rightarrow (1,1)(1,0)$, $\mathcal{P}_2 \rightarrow (1,1)$, $\mathcal{P}_3 \rightarrow (1,1)(0,1)$.

For a path that begins at (1, 1), we usually set $p_1 = q_1 = 1$.

$$\phi_x(k) = \sum_{i=1}^{k} p_i \tag{5.17}$$

$$\phi_y(k) = \sum_{i=1}^{k} q_i \tag{5.18}$$

If path ends at (T_x, T_y), then we have

$$T_x = \sum_{k=1}^{T} p_k \tag{5.19}$$

$$T_y = \sum_{k=1}^{T} q_k \tag{5.20}$$

Figure 5.5 shows the few sets of local constraints and their path specifications in the (i_x, i_y) plane. Type I constraint is identical to what mentioned in equation 5.14 and 5.15. In path I and path II, all paths are single-move path, whereas in type III, path $\mathcal{P}_1$ and $\mathcal{P}_2$ takes two moves and paths $\mathcal{P}_3$ and $\mathcal{P}_4$ takes one moves. All types of constraints can be interpreted accordingly to the path specification shown in Figure 5.5, except path type proposed by Itakura, at bottom of the figure which prohibits

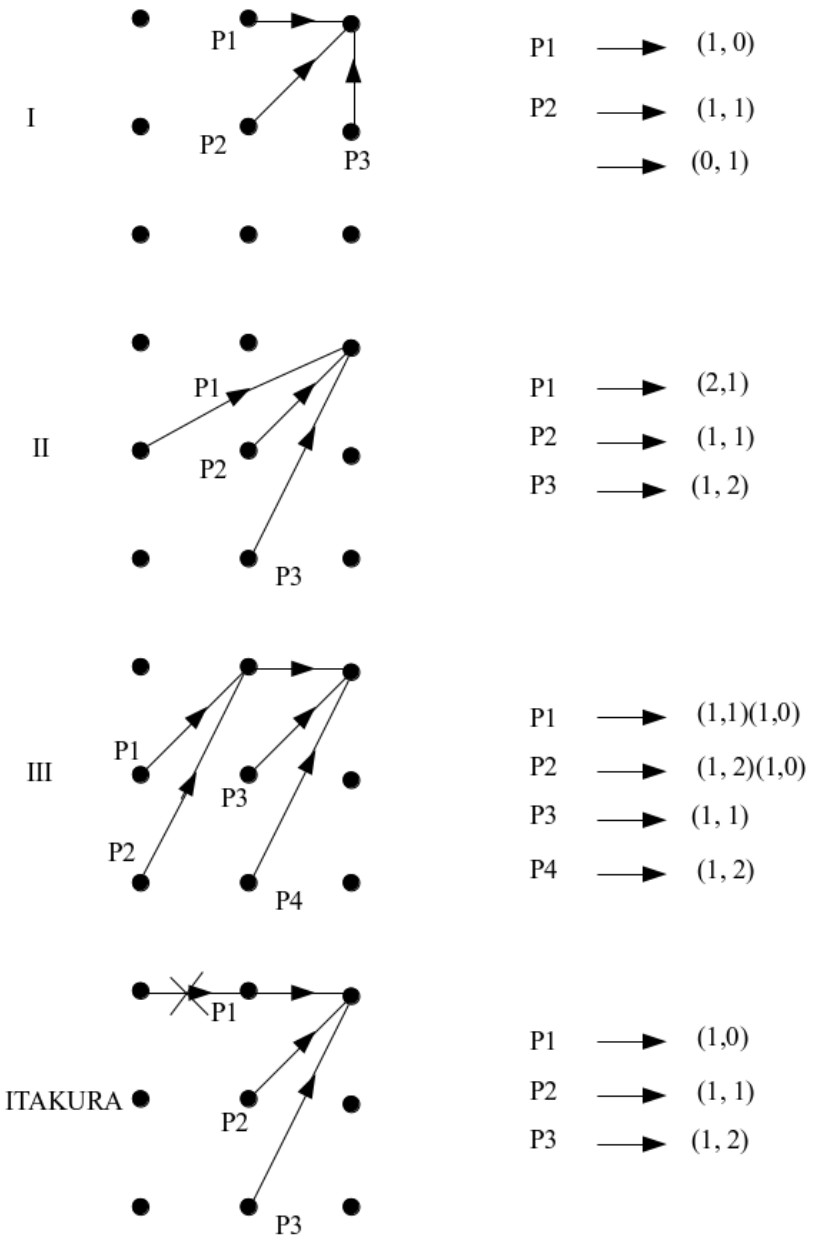


Figure 5.5: Set of local constraints and their resulting path specification.

successive $(1, 0)(1, 0)$ moves.

Due to local continuity constraints certain region of the (i_x, i_y) plane are excluded from the region where optimal warping path can traverse. Because of this, global constraints can be imposed to allowable warping path. Such constraints speed up the computation of DTW as well as control the route of a warping path. Such

allowable region of the (i_x, i_y) can be defined as:

$$Q_{min} = \min_{\ell} \left[\sum_{i=1}^{T_\ell} p_i^{(\ell)} \Big/ \sum_{i=1}^{T_\ell} q_i^{(\ell)} \right] \tag{5.21}$$

$$Q_{max} = \max_{\ell} \left[\sum_{i=1}^{T_\ell} p_i^{(\ell)} \Big/ \sum_{i=1}^{T_\ell} q_i^{(\ell)} \right] \tag{5.22}$$

where ℓ is the allowable path index, $\mathcal{P}_\ell$ is the local continuity constraints set and T_ℓ is total moves in $\mathcal{P}_\ell$. For example, in type I constraints, $\ell = 1, 2, 3$ and $T_\ell = 1, 1, 1$ respectively for $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$. Q_{min} and Q_{max} gives minimum and maximum possible expansion in warping. Normally $Q_{min} = 1/Q_{max}$.

Using minimum and maximum values for path expansion, global path constraint can be defined as

$$1 + \frac{[\phi_x(k) - 1]}{Q_{max}} \leq \phi_y(k) \leq 1 + Q_{max}[\phi_x(k) - 1] \tag{5.23}$$

$$T_y + Q_{max}[\phi_x(k) - T_x] \leq \phi_y(k) \leq T_y + \frac{[\phi_x(k) - T_x]}{Q_{max}} \tag{5.24}$$

Equation 5.23 gives the range of a point that can be reached starting from point $(1, 1)$ in the (i_x, i_y) plane. Whereas equation 5.24 gives the range of points having legal path to the end point (T_x, T_y).

Sakoe and Chiba [19] proposed additional global path constraint to prevent excessive time stretch or compression.

$$|\phi_x(k) - \phi_y(k)| \leq T_0, \tag{5.25}$$

where T_0 gives the maximum allowable absolute time deviation between two patterns at any moment. Constraints from equation 5.25 is known as range-limiting constraints as they limit absolute difference in the warped time scale.

5.3.4 Slope Weighting

Slope weighting adds another level of control while searching warping path. As mentioned in equation 5.5, the weighting function, $m(k)$ is used to minimize the effect of each short-time distortion $d(\phi_x(k), \phi_y(k))$. In order to prefer certain locally allowable path, one can associate the weighting function to prescribed path constraints. Such weighing functions defined locally are known as slope weighting function as they are responsible for the slope of the local path constraints.

Sakoe and Chiba [19] proposed four type of slope weighting functions

$$m(k) = min[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)] \tag{5.26}$$

$$m(k) = max[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)] \tag{5.27}$$

$$m(k) = \phi_x(k) - \phi_x(k-1) \tag{5.28}$$

$$m(k) = \phi_x(k) - \phi_x(k-1) + \phi_y(k) - \phi_y(k-1)], \tag{5.29}$$

For initialization, let us assumed that $\phi_x(0) = \phi_y(0) = 0$. The number is associated with each path which is known as weight for that path. For less preferable paths, we use higher weighting value. Hence the weight attempts to maintain a bias towards the path with less weighting value.

5.4 Dynamic Time Warping

Now we can use dynamic programming algorithm to solve the minimization problem involved in the definition of pattern dissimilarity measure in equation 5.6 with embedded time normalization and alignment. Dynamic Programming equation, mainly equation 5.9, and optimality principle are directly applicable to this problem. Slope weighting and local path constraints mentioned in an earlier section require some adjustment to the original algorithm. Because of the endpoint constraints (both X and Y terminates at T_x and T_y respectively), we can rewrite equation 5.6 in terms of T_x and T_y as

$$M_\phi d(X,Y) \triangleq D(T_x, T_y) = \min_{\phi_x, \phi_y} \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k)) m(k) \tag{5.30}$$

Also the minimum partial distortion accumulated along the path connecting $(1,1)$ and (i_x, i_y) is

$$D(i_x, i_y) \triangleq \min_{\phi_x, \phi_y, T'} \sum_{k=1}^{T'} d(\phi_x(k), \phi_y(k)) m(k), \tag{5.31}$$

where $\phi_x(T') = i_x$ and $\phi_y(T') = i_y$. Thus dynamic programming recursion with constraints becomes

$$D(i_x, i_y) = \min_{(i'_x, i'_y)} [D(i'_x, i'_y) + \zeta((i'_x, i'_y), (i_x, i_y))], \tag{5.32}$$

where ζ is the weighted accumulated distortion between two points (i_x, i_y) and (i'_x, i'_y)

$$\zeta((i'_x, i'_y), (i_x, i_y)) = \sum_{\ell=0}^{L_s} d(\phi_x(T' - \ell), \phi_y(T' - \ell))m(T' - \ell), \tag{5.33}$$

where L_s is the total number of moves along the path from (i_x, i_y) to (i'_x, i'_y) and $\phi_x(T' - L_s) = i'_x$ and $\phi_y(T' - L_s) = i'_y$.

Now we evaluate the incremental distortion ζ along the allowable path given by chosen local continuity constraints. The range of (i'_x, i'_y) for minimization in equation 5.32 is limited by the chosen set of local constraints. Figure 5.6 shows recursive formula for few sets of local constraints with slope weighing.

As stated in the previous section, there is a clear distinction between type III local constraint and the Itakura constraints. For Itakura constraints, one function $g(k)$ should be introduced in order to prevent paths traversing horizontally for more than one move. If the best path to reach a point $(i_x - 1, i_y)$ is from point $(i_x - 2, i_y)$, then the algorithm will reject all other allowable connection to point $(i_x - 1, i_y)$ except from the point $(i_x - 2, i_y)$. If such case occurs then the best path to reach point (i_x, i_y) can only be either from point $(i_x - 1, i_y - 1)$ or from point $(i_x - 1, i_y - 2)$, because Itakura constraints do not allow two consecutive horizontal moves. The path from point $(i_x - 2, i_y - 1)$ to point (i_x, i_y) through point $(i_x - 1, i_y)$ is not considered even if the accumulated distortion at (i_x, i_y) through this particular path is smaller as compare to through other points. But in case of Type III local constraints, the partial accumulated distortion at two successive frames is maintained hence the path satisfying the constraints of no two successive horizontal moves can be found.

Since normalizing factor M_ϕ in equation 5.5 is considered to be independent to warping path, we can remove it out of the dynamic programming recursion. Once the optimal path is discovered, factor M_ϕ can be restored as in equation 5.30. Now let us summarize the dynamic programming implementation of discovering best path through T_x by T_y grid which starts at point $(1, 1)$ and ends at (T_x, T_y).

- Initialization

$$D_A(1, 1) = d(1, 1)m(1, 1). \tag{5.34}$$

- Recursion

  For $1 \leq i_x \leq T_x$, $1 \leq i_y \leq T_y$ such that i_x and i_y stay within the allowable grid, calculate

$$D_A(i_x, i_y) = min_{(i'_x, i'_y)}[D_A(i'_x, i'_y) + \zeta((i'_x, i'_y), (i_x, i_y))], \tag{5.35}$$

  where $\zeta((i'_x, i'_y), (i_x, i_y))$ is defined in equation 5.33.

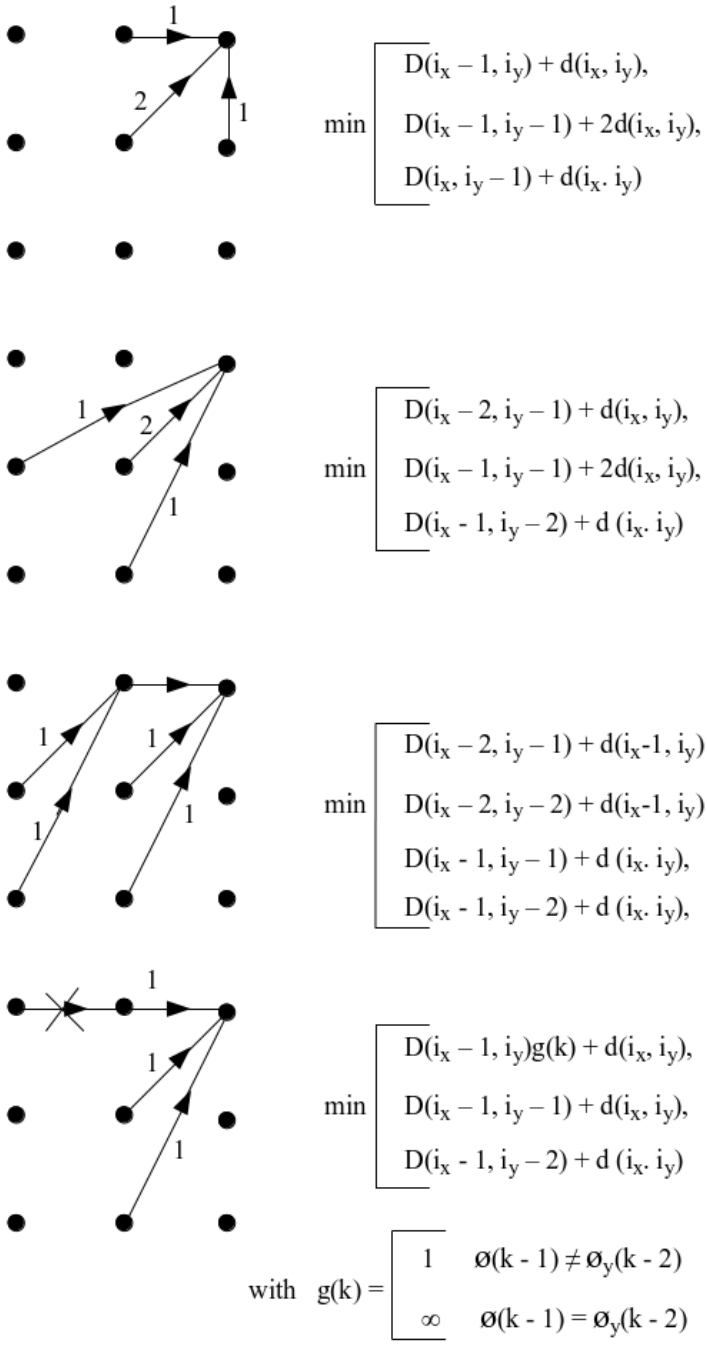


Figure 5.6: Few sets of local constraints with slope weights and DP recursion formula.

- Termination

$$d(X,Y) = \frac{D_A(T_x, T_y)}{M_\phi}. \tag{5.36}$$

This algorithm performs recursion steps to all local path that leads us to point (i_x, i_y) in just a single step from the point (i'_x, i'_y) using chosen local path constraints

for the implementation. In these recursion steps, algorithm only evaluates the value of those points (i_x, i_y) that can only be reached from the starting point $(1, 1)$ and that can end at (T_x, T_y). Depends upon the chosen local path constraints, there are less number of points in allowable path region. The number of points falling in this region is also the number of calculation, $\zeta((i'_x, i'_y), (i_x, i_y))$, required for implementing DTW algorithm. In general, it is believed that implementation of DTW requires about 80% of the computation and rest is about optimal path search. So by proper use of constraints, we can reduce the amount of calculation required.

When using a local path constraints which allows k-to-1 time scale of contraction and expansion, then according to [33], the ratio of grid points in allowable path region to the grid point in rectangle of $T_x X T_y$ is

$$R = \frac{\left(k - \frac{T_x}{T_y}\right)\left(k - \frac{T_x}{T_y}\right)}{k^2 - 1} \tag{5.37}$$

Therefore if we take a case where $T_x = T_y$ with $k = 2$, then we get $R = 1/3$. Similarly on setting $T_x = T_y$ and $k = 3$, we obtain $R = 1/2$, that is, about half of grid points are allowed with contraction and expansion of 3:1 scale.

5.5 Implementation and result

Dynamic Time Warping experiment was conducted in two data sets, RWC and TUT using two types of feature vectors, first with chroma (CENS) feature and then with beat synchronous chroma feature. Our experiment starts with offline processing which included synthesizing database items (MIDI files) to audio format using timidity software, extracting features (chroma, beat and beat synchronous chroma) from those items and storing it for future use.

For DTW, one should choose the type of local continuity constraints and slope weights aka path weight. For this set of experiment, we will be using Type I and Type II local continuity constraints as mentioned in section 5.3.3 and Figure 5.5. Slope weight is varied in between 1 to 5.

Figure 5.7 and 5.8 shows the dynamic warping path for audio and midi version of same song (audio and midi from the same source, as in the case of RWC database). Here red line shows the dynamic time warping path with minimal cost. Figure 5.8 is the zoomed version of path in Figure 5.7. Now let us check the dynamic time warping path for real world situation, where audio and midi are not from the same source and in such cases there may be structural difference between midi and audio. Figure 5.9 shows the dynamic time warping path for four songs whose source are different, as in the case of TUT database. Structural differences were evident for the songs shown in Figure 5.9(a) and 5.9(b), where we can see that distance matrix

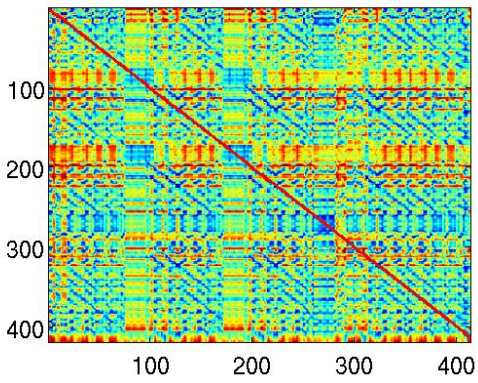


Figure 5.7: DTW Path on similarity matrix for Midi and audio version of same song.

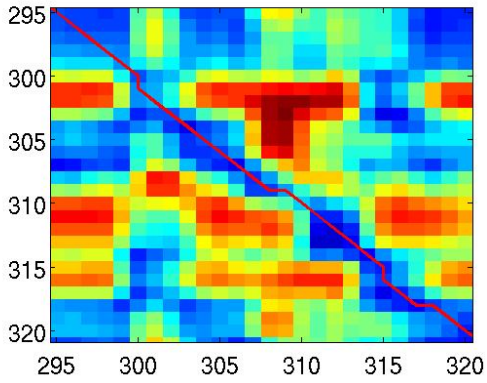
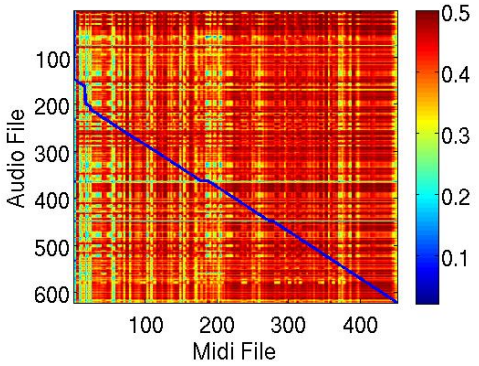
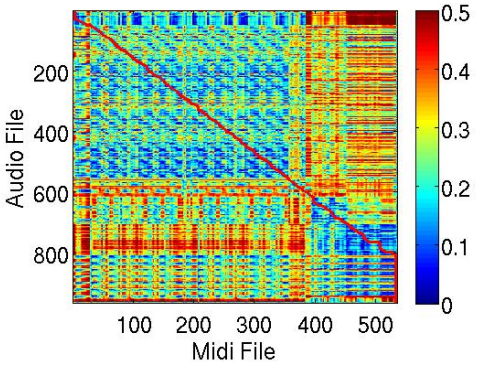


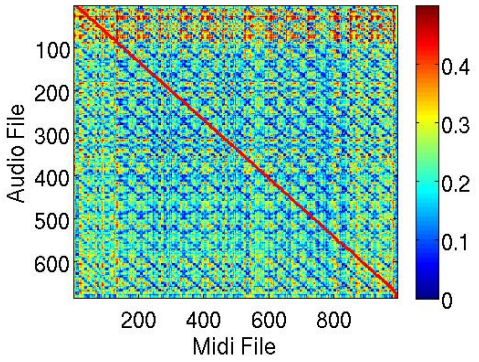
Figure 5.8: Zoomed version of DTW path in Figure 5.7



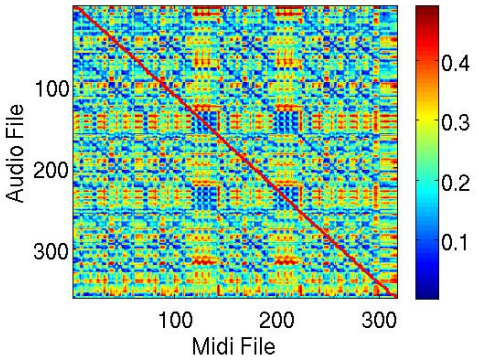
(a) DTW Path for song I want to break free.



(b) DTW Path for song you are not alone.



(c) DTW Path for song sweet home.



(d) DTW Path for song mamma mia.

Figure 5.9: DTW path over similarity matrix for real world real world scenario where audio and midi are from different source.

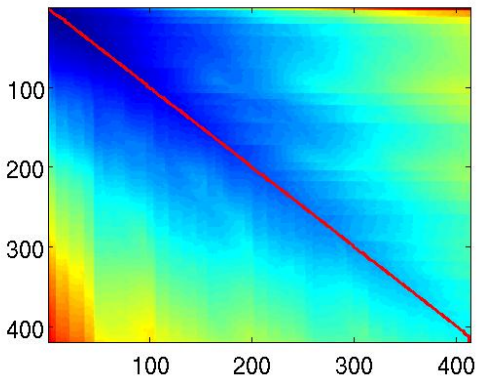


Figure 5.10: DTW Path over DTW matrix for Midi and audio version of same song.

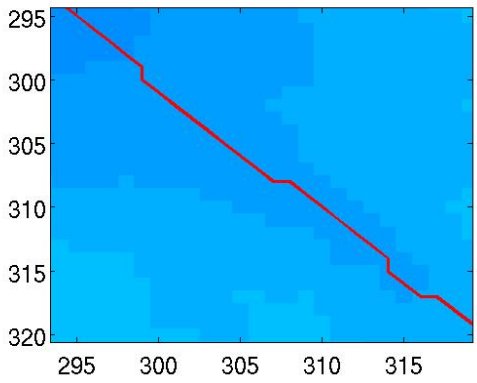
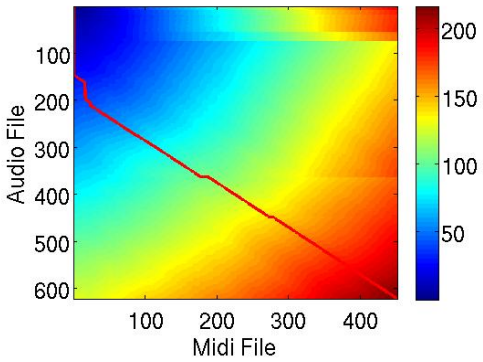
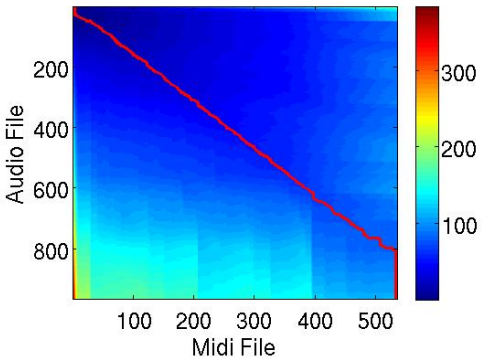


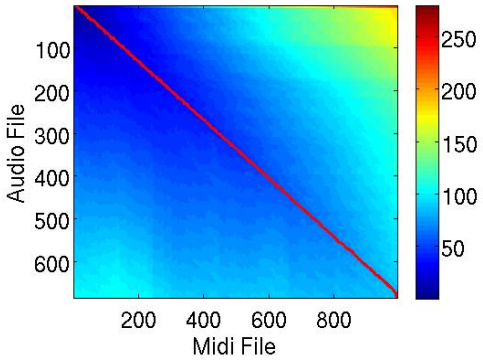
Figure 5.11: Zoomed version of DTW path in Figure 5.10



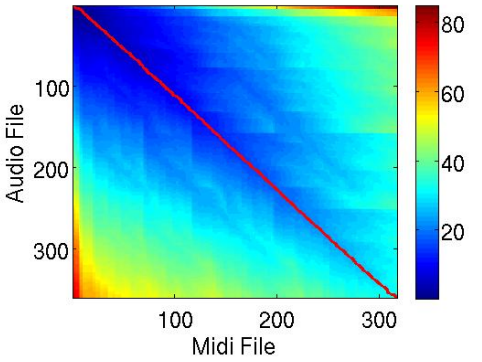
(a) DTW Path for song I want to break free.



(b) DTW Path for song you are not alone.



(c) DTW Path for song sweet home.



(d) DTW Path for song mamma mia.

Figure 5.12: DTW path over DTW matrix for real world real world scenario where audio and midi are from different source.

Table 5.1: Results of DTW, On RWC using different features.

Weight	CENS Feature		BSC Feature		Mod BSC Feature	
$[W_{hor}, W_{dia}, W_{ver}]$	Path 1	Path 2	Path 1	Path 2	Path 1	Path 2
[1 1 1]	92	92	89	70	**89**	84
[2 1 2]	92	**94**	**91**	**73**	88	84
[3 1 3]	**93**	**95**	**91**	72	88	82
[4 1 4]	92	**94**	90	72	87	79
[5 1 5]	**93**	93	85	72	88	77
[1 2 1]	81	92	83	69	79	84
[2 2 2]	92	92	89	70	**89**	84
[3 2 3]	92	92	90	71	**89**	84
[4 2 4]	92	**94**	90	**73**	88	84
[5 2 5]	92	**95**	**91**	**73**	88	84
[1 3 1]	81	92	78	69	71	84
[2 3 2]	85	92	85	69	80	84
[4 3 4]	92	92	90	71	**89**	84
[5 3 5]	92	93	90	71	**89**	84
[1 4 1]	81	92	71	69	70	84
[2 4 2]	81	92	83	69	79	84
[3 4 3]	85	92	86	70	84	84
[5 4 5]	92	92	90	71	**89**	84
[1 5 1]	81	92	68	69	68	84
[2 5 2]	81	92	79	69	73	84
[3 5 3]	84	92	85	69	80	84
[4 5 4]	87	92	87	70	86	84

and path were broken. Such situation might occur because songs may be performed by different singer at different speed; length of two song are not same resulting in a difference in starting point and/or end point of two songs; some song may have extra repetitive chorus. Such off-diagonal lines visible in figure 5.9(a) and 5.9(b) makes matching of the song harder. Whereas in case Figure 5.9(c) and 5.9(d) there is not any noticeable structure difference and in such case where DTW path remains within a diagonal makes matching comparatively easy.

Now take a query audio file (which may be a full song or a small segment of a song), extract its relevant features and then search for a similar item in the database using DTW. For every item in the database, calculate the distance matrix with query item and then calculate the path cost of traversing from the initial point to the final point. The database item with minimum path cost is considered as the most similar song to the query song. We can also retrieve N similar songs by retrieving the songs with N minimum path cost.

Table 5.2: Result of applying DTW on RWC database for 30 seconds query songs.

Weight	BSC Feature		Mod BSC Feature	
$[W_{hor}, W_{dia}, W_{ver}]$	Path 1	Path 2	Path 1	Path 2
[1 1 1]	65	60	61	60
[2 1 2]	54	60	49	59
[3 1 3]	50	59	42	48
[4 1 4]	46	51	36	41
[5 1 5]	43	47	32	37
[1 2 1]	65	60	61	**61**
[2 2 2]	65	60	61	60
[3 2 3]	60	**62**	54	59
[4 2 4]	54	60	49	59
[5 2 5]	52	58	44	50
[1 3 1]	60	61	57	60
[2 3 2]	**66**	**62**	**63**	**62**
[3 3 3]	65	60	61	60
[4 3 4]	62	61	54	60
[5 3 5]	57	61	51	59
[1 4 1]	56	62	54	60
[2 4 2]	65	60	60	**61**
[3 4 3]	**67**	**62**	**65**	**61**
[4 4 4]	65	60	61	60
[5 4 5]	64	61	55	60

RWC Database

For this experiment, we have 100 MIDI songs stored in the database and we use their corresponding audio format as query item. Table 5.1 shows the retrieval rate of DTW using CENS feature on RWC database. Here Path I and path II means local continuity constraints type I and type II as shown in Figure 5.5. Another set of experiment was conducted using beat synchronous chroma feature on RWC database. Furthermore we also modified the beat synchronous chroma feature by up sampling and down sampling beat sequences based on the value of tempo (110 in our case). If tempo of the song is less that 110, then its beat sequences is up sampled else down sampled and then calculate beat synchronous chroma feature. Table 5.1 shows the retrieval rate of DTW using BSC and modified BSC feature on RWC database.

From the result, we can see that the performance rate is higher when we favor diagonal move which means when we give less weight for diagonal path as compared to horizontal and vertical one. We can also see from results that if weight for diagonal path is too less than horizontal and vertical then also performance degrades. So we can conclude that, for CENS feature, we have best performance rate by using

Table 5.3: Results of DTW, On RWC by deleting and repeating segments.

Weight	Deleting Segment		Repeating Segment	
$[W_{hor}, W_{ver}, W_{dia}]$	Path 1	Path 2	Path 1	Path 2
[1 1 1]	87	85	**84**	63
[2 1 2]	**88**	85	**83**	65
[3 1 3]	**88**	83	78	65
[4 1 4]	87	79	77	64
[5 1 5]	**88**	76	72	63
[1 2 1]	76	85	74	64
[2 2 2]	87	85	**84**	63
[3 2 3]	**88**	85	82	64
[4 2 4]	**88**	85	82	64
[5 2 5]	**88**	85	81	64
[1 3 1]	74	85	69	63
[2 3 2]	81	85	78	64
[3 3 3]	87	85	**84**	63
[4 3 4]	**88**	85	**83**	65
[5 3 5]	**88**	85	82	65
[1 4 1]	72	85	67	64
[2 4 2]	76	85	72	63
[3 4 3]	82	85	80	64
[4 4 4]	87	85	**84**	63
[5 4 5]	86	85	**83**	65
[1 5 1]	69	83	66	63
[2 5 2]	75	83	70	62
[3 5 3]	81	85	75	63
[4 5 4]	83	84	80	64
[5 5 5]	87	85	**84**	63

local continuity constraint type II with path weight [3 1 3] and [5 2 5]. However, path weight [2 1 2], [4 2 4] and [4 1 4] also produce a good result for local continuity constraint type II. But for the case of beat synchronous chroma feature, local continuity constraint type I outperform type II. In this case, path weight [2 1 2], [3 1 3] and [5 2 5] produce a good result local continuity constraint type I. While using modified BSC, local continuity constraint I produce good results.

From the table 5.1, we can see that the retrieval rate for local continuity constraint type II using BSC feature is not good as compared to local continuity constraint type I. On close examination of DTW path for the songs that fails for local continuity constraint type II using BSC feature, we found that there were some horizontal and vertical line segment. Since Local continuity constraint type II do not allow us to move through low cost horizontal and vertical path, we were forced to move through higher cost path increasing the overall path cost and resulting in incorrect match.

Table 5.4: Applying DTW in TUT dataset

Weight	BSC Feature		Modified BSC Feature	
$[W_{hor}, W_{ver}, W_{dia}]$	Path 1	Path 2	Path 1	Path 2
[1 1 1]	67.5862	**54.1379**	73.1034	**58.6207**
[2 1 2]	**69.3103**	53.4483	**79.3103**	57.5862
[3 1 3]	65.5172	51.7241	71.0345	54.4828
[4 1 4]	61.7241	50.0000	64.1379	51.3793
[5 1 5]	58.2759	47.5862	60.0000	49.6552
[1 2 1]	42.0690	53.7931	48.9655	57.9310
[2 2 2]	67.5862	**54.1379**	73.1034	**58.6207**
[3 2 3]	**72.7586**	**54.1379**	**78.9655**	**58.9655**
[4 2 4]	**69.3103**	53.4483	**79.3103**	57.5862
[5 2 5]	66.8966	52.0690	74.1379	55.5172
[1 3 1]	27.9310	53.7931	41.3793	57.9310
[2 3 2]	52.4138	**54.4828**	60.0000	58.2759
[4 3 4]	**72.0690**	**54.1479**	**78.6207**	**58.9655**
[5 3 5]	**71.7241**	**54.4828**	**79.6552**	**59.3103**
[1 4 1]	24.1379	53.4483	38.9655	57.2414
[2 4 2]	42.0690	53.7931	48.9655	57.9310
[3 4 3]	57.2414	**54.4828**	63.7931	58.2759
[5 4 5]	**72.4138**	**54.1379**	76.8966	**58.9655**
[1 5 1]	22.7586	53.4483	38.2759	57.2414
[2 5 2]	33.4483	53.4483	44.1379	57.9310
[3 5 3]	47.9310	**54.4828**	54.4828	57.9310
[4 5 4]	60.0000	**54.4828**	65.5172	58.2759

Moreover regarding processing time, when I query with the song 315 seconds, it took almost 55 seconds to compare it against 100 database items and return the result. When I query with song 136 seconds long, it took almost 11 seconds to return the result (using beat synchronous chroma feature) using local continuity constraint type I and giving equal weight to all paths.

There may also be a situation when there is distortion in the query songs, meaning there may be a modification in original songs like deletion of a certain portion from a song or repetition of a certain portion of the song. We were also interested to see the retrieval performance of our method in such case. In order to simulate such situation, one set of experiment was conducted by deleting certain portion of a query song (approximately 25 - 30 seconds) from a random position. And another set of experiment was conducted by repeating certain section (approximately 25 - 30 seconds) at random position from every query item. Table 5.3 shows the performance rate using BSC features on RWC database. It shows that the matching is robust against deletion and duplication of song at random position. Here also we can see that local continuity constraint type I perform better than local continuity

Table 5.5: Result of applying DTW on TUT database for 30 seconds query songs.

Weight	BSC Feature		Mod BSC Feature	
$[W_{hor}, W_{dia}, W_{ver}]$	Path 1	Path 2	Path 1	Path 2
[1 1 1]	**45.86**	43.45	**50.34**	49.31
[2 1 2]	41.03	**44.48**	46.55	**51.03**
[3 1 3]	37.24	40.69	43.45	49.31
[4 1 4]	35.52	38.97	39.66	47.24
[5 1 5]	33.45	35.17	36.90	43.79

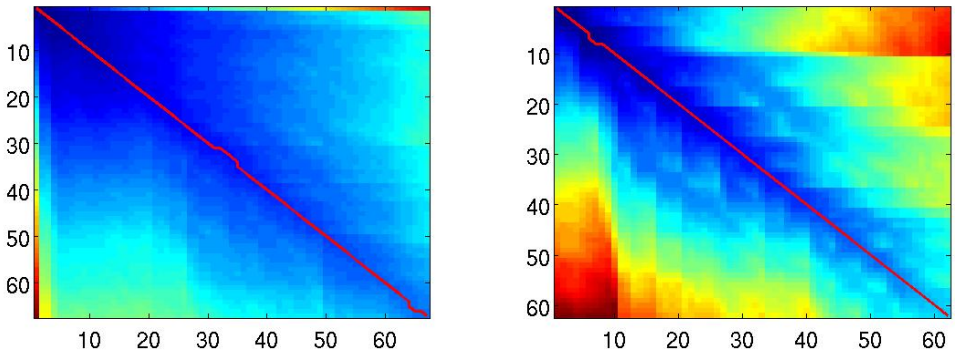
constraint type II, because while deleting and repeating certain segment there occurs a horizontal or vertical line segment in DTW path causing matching harder for local continuity constraint type II.

Next experiment was performed using a small segment of audio query. From the set of 100 query songs, each songs were cut into 30 seconds segment from a random position. And those 30 seconds segments were used as query song. It do not make any sense applying DTW between 30 seconds query segment with full songs from the database. So a sliding window is created, whose length is equal to the length of the query sequence. This window is moved through every index of a song from the database. DTW is applied between the database sequences captured by a window with the query sequence. Table 5.2 shows the retrieval rate using BSC and modified BSC feature on RWC database. For path weight [3 4 3] and local continuity constraint type I, out of 100 query samples, 67 were able to identify their corresponding song. Out of those 67 matching songs, 42 were able to identify the correct position of query clip in full song from the database (Here, deviation within the range of 10 seconds was considered as the correct match for the position). Instead of applying dynamic path, we also tried to apply static path that moves through diagonal line. In this case, retrieval rate reduced to 55%.

Using modified beat synchronous chroma feature for a query item only, retrieval rate was 60% and 29% using dynamic and static path respectively. And using such modified beat synchronous chroma feature for both database and query item, retrieval rate was 65% and 48% for dynamic and static path respectively.

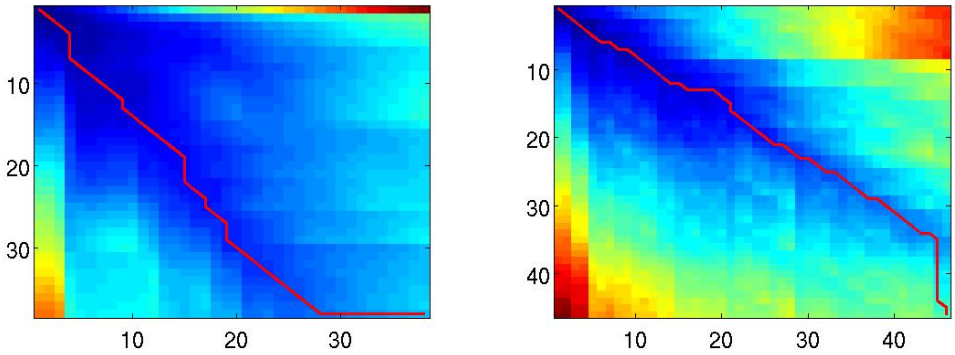
TUT

In the next step, DTW algorithm was applied on TUT database containing songs from a different source. Same procedure was repeated for this database as it was done for RWC database. Here we use beat synchronous chroma feature to represents query and database songs. Here also we modify beat synchronous chroma feature based on the value of tempo, as we did for RWC database. For this, we first compute the value of tempo. If tempo is greater than 110 then beat sequences were down sampled else



(a) Song ID 3. (b) Song ID 5.

Figure 5.13: DTW path where static and dynamic path give correct match.



(a) Song ID 16. (b) Song ID 55.

Figure 5.14: DTW path where static path fails but dynamic path works.

up sampled and then finally beat synchronous chroma feature were computed. Table 5.4 below shows the retrieval rate. We can observe that performance increase by modifying beat synchronous chroma feature based on tempo and favoring diagonal move.

Here we can see that the retrieval rate for TUT database was low as compared to RWC database, it is because the midi and audio songs in TUT database are from a different source and have structural difference. Such structural difference makes the matching harder. For the same reason as explained in the case of RWC database local continuity constraint type I perform better than local continuity constraint type II.

We also performed an experiment using 30 seconds of audio clip as a query. Table 5.5 shows the performance for DTW. Here we were able to achieve retrieval rate up to 50%. We can also see that the retrieval rate was low as compared to RWC database due to the structural difference.

For the case of static path, the retrieval rate was 34.48% for BSC feature and 32.07% for modified BSC feature.

5.6 Static Vs Dynamic

In the next step, we were interested in the cases where both static and dynamic path leads us to the correct result. Here we use a small segment of a query song and try to find out correct song with correct time index from the database item. Figure 5.13 shows the resulting DTW path for two songs where both dynamic and static path give correct time index matching (deviation of 2 seconds is considered as the correct match). Here we can see DTW path almost being diagonal. Furthermore, we were also interested in the path where dynamic path leads to the correct match but static path fails. Figure 5.14 shows the resulting dynamic path. We can see that the path has clearly vertical or horizontal segment.

6. LOCALITY SENSITIVE HASHING

A nearest neighbor (NN) problem involves the collection of a relevant features of an object (music, image, video, documents, etc.) represented as points in d-dimensional space; with a query point, we are required to retrieve the similar objects (nearest points) to the query object. The feature of each object is represented as a point in high dimensional space. The similarity between them is measured by the distance matrix between them. NN problem can be thought up as "given a collection of n data points, built an algorithm that returns a set of data points that are near (similar) to a given query point". In theory, this seems like an easy walk, but as the size of database increases and the objects are complicated (high dimensional) then the processing time increases with the size of database and complexity of an object. The most researched instance is when data point lies on d-dimensional Euclidean space. The importance of this problem lies in many areas like: machine learning, pattern recognition, information retrieval, data compression, image and video database, statistics and data analysis.

There are many efficient algorithms for low dimensional cases. One could be done by building tree like structure for an object. With a query point, we can start from the root node and determine whether our query lies to the left or right of the current node. This process continues till we descend to the leaf node of a tree. For a perfectly balanced tree, this problem is solved in $O(logN)$ times, where N is the number of nodes of the tree. For a single dimension, this works well. For a multidimensional case, this idea becomes as kd-tree and was first coined by Jon Bently [29] in 1975. And it remained popular data structure for multidimensional search. Later many more multidimensional data structure were introduced, see [20] for detail. Currents solution suffers from either query time or space that is exponential in d. In reality, for large d, either in theory or in practice, current solutions provide little improvement to linear algorithms that compare a query point to all other points from the database. This phenomenon is called "the curse of dimensionality". It was shown in [56] that all current indexing techniques, other than LSH based indexing, degrades to linear search for high dimensional data. This problem is serious for large scale similarity search.

In order to remove this bottleneck, many researchers have proposed an idea of using approximation. In many cases, instead of finding exact nearest neighbor,

finding the approximate nearest neighbor is sufficient. In these techniques, the algorithms return a point that is at a distance of at most c times the distance between the query and its nearest point; $c > 1$ is known as approximation factor. Most of the cases, it happens that the relevant answer lies closer to the query point as compared to irrelevant points. This is advantageous property of good similarity search. Moreover, another desirable point is that approximate similarity search are faster than the exact one.

In this section, we focus on one of the most popular approximate similarity search algorithm for high dimension based on the concept of locality-sensitive hashing (LSH) [54]. The main idea behind it is to hash a point using several hash functions. For each hash function, near (similar) objects have higher probability of collisions than the objects that are far apart. Then one can retrieve near neighbors by hashing the query point with hash function and retrieve all the elements from bucket containing the query point. LSH algorithm is popular in a variety of areas like computer vision, computational biology, web clustering etc.

Here l_s^d is used to denote the Euclidean space $\Re^d$ under the l_s norm, which means the length of a vector $(x_1, x_2, x_3, \ldots, x_d)$ is defined as $(|x_1|^s + |x_2|^s + |x_3|^s + \ldots + |x_d|^s)^{1/s}$. Let us denote d-dimensional set of data point by P, and points p chosen from from P also belong to d-dimensional space $\Re^d$. Let the i^{th} coordinate of p, for $i = 1, 2, 3, \ldots, d$ is denoted by p_i. Further, the distance between two points p and q in l_s^d is denoted by $d_s(p, q) = \|p - q\|_s$ and is defined as

$$\|p - q\|_s = \left(\sum_{i=1}^{d} |p_i - q_i|^s \right)^{1/s} \tag{6.1}$$

where $s > 0$. Generally for Euclidean distance $s = 2$ or for Manhattan distance $s = 1$. Often subscript 2 is skipped while representing Euclidean distance, that is, $\|p - q\| = \|p - q\|_2$.

6.1 R-near Neighbor

Any point p is said to be R-near neighbor of a query point q, if those two points are at most R distance apart. In such case, if there are any points in R-near neighbor, the algorithm returns one of them. If there are not any point in R-near neighbor, then the algorithm indicates that there do not exist any point for the parameter R. R-near neighbor and nearest neighbor are related because one can check that the point returned by nearest neighbor is within a distance of R or not. But the reverse is somehow complicated, which involves creating a different instance of R-near neighbor, changing the value of R. We query the data structure in increasing order of R and when data structure returns a point then the process is stopped.

The original LSH solves the approximate version of R-near neighbor problem known as (R, c)- near neighbor problem. In this formulation, if there is any point in P which is at most R distance from the query point q, then it is sufficient for an algorithm to return any point within a distance of at most cR from the query point q. Approximate version of near neighbor can be defined as follows.

6.1.1 Randomized c-approximate R-near neighbor or (c, R)-NN

Given a set of data points, P, in a d-dimensional space $\Re^d$, with parameters failure probability $\delta > 0$ and $R > 0$, build a data structure such that, with any query point q, if there exists an R-near neighbor of q in p, gives you some cR-near neighbor of q in P, with probability $1-\delta$. Let us assume δ is an absolute constant that is bounded away from 1 (for example 1/2). In order to increase the probability of success, we can build and query several instance of data structure. For example, if we build two independent data structures both with $\delta = 1/2$, this will gives you the overall failure probability of $\delta = 1/2.1/2 = 1/4$. We can scale down the coordinates of all points by R. We can assume $R = 1$, so often it is skipped and refer to c-approximate near neighbor problem or c-NN.

6.1.2 Randomized R-near neighbor reporting

Given a set of data points, P, in a d-dimensional space $\Re^d$ with parameters $\delta > 0$ and $R > 0$, built a data structure such that, with any query point q, gives you the R-near neighbor of q in P with probability of $1 - \delta$.

6.2 Basic LSH

The basic building block of Locality-sensitive hashing is locality-sensitive hash functions. Let us assume $\mathcal{H}$ denotes the family of hash functions mapping to $\Re^d$. For any two points p and q, we choose a function h from $\mathcal{H}$ uniformly at random to analyze the probability of $h(p) = h(q)$. The family $\mathcal{H}$ is called locality sensitive, or (R, cR, P_1, P_2)-sensitive, for any two points p and q belong to $\Re^d$, if it satisfy following conditions.

- if $\|p - q\| \leq R$ then $P_{r_\mathcal{H}}[h(q) = h(p)] \geq P_1$.
- if $\|p - q\| \geq cR$ then $P_{r_\mathcal{H}}[h(q) = h(p)] \leq P_2$.

LSH family is useful if and only if $P_1 > P_2$.

We can use LSH family $\mathcal{H}$ for designing an efficient algorithm for approximate near neighbor search. Since the gap between high probability P_1 and low probability

P_2 could be quite small, one needs to amplify the gap between them to achieve desired collision probability. Having a family of hash functions $\mathcal{H}$ with parameters (R, cR, P_1, P_2), the gap between the two probabilities P_1 and P_2 can be amplified by concatenating several hash functions. For parameter k and L (discussed later), choose L functions $g_j(q) = (h_{1,j}(q), h_{2,j}(q), \ldots, h_{k,j}(q))$, where $h_{t,j}(1 \leq t \leq k, 1 \leq j \leq L)$ are chosen independently and uniformly at random from $\mathcal{H}$. These are the actual hash function used to hash data points.

One can construct a data structure by placing each and every point p into a bucket $g_j(p)$, for $j = 1, 2, 3, \ldots, L$. We may have a large number of buckets; only non-empty buckets are retained. Instead of storing data points, pointers to the data points are stored into buckets. While processing any query point q, all the buckets $g_1(q), g_2(q), g_3(q), \ldots, g_L(q)$ are scanned and all points stored on the buckets are retrieved. After having such points, their distance to the query point is computed and any point that is valid answer to query is returned. Two scanning strategies are possible.

- For some parameter L', the search is interrupted after finding L' points (including duplicates).
- The search is continued till all data points from buckets are retrieved; additional parameter is not required.

Strategy 1 and Strategy 2 solves (c, R)-near neighbor problem and R-near neighbor reporting problem respectively. Algorithm is correct if it satisfies both strategies. According to Theorem 4 of [54], if you set $k = log_{1/P_2} n$ and $L = n^{\rho}$, where $\rho = \frac{\ln 1/P_1}{\ln 1/P_2}$ and n number of data in d-dimensional space, then both strategies hold with constant probability.

6.3 LSH scheme based on stable distribution

In this section, we introduce and analyze LSH family based on s-stable distribution, for any $s \in (0, 2]$. A distribution $\mathcal{D}$ over $\Re^d$ is called s-stable, if there exists $s \geq 0$ such that for any n real number $v_1, v_2, \ldots, v_n$ and i.i.d. variables $X_1, X_2, \ldots, X_3$ with distribution $\mathcal{D}$, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^s)^{1/s} X$, where X is a random variable with distribution $\mathcal{D}$ [43]. Stable distribution exists for $s \in (0, 2]$, particularly

- a Cauchy distribution $\mathcal{D}_C$, defined by density function $c(x) = \frac{1}{\pi} \frac{1}{1+x^2}$, is 1-stable.
- a Gaussian (normal) distribution $\mathcal{D}_G$, defined by density function $g(x) = \frac{1}{\sqrt{2\pi}} \exp^{-x^2/2}$, is 2-stable.

Stable distribution is applied in various applications. [55] used it for sketching of high dimensional vectors and since it is used in numerous applications. The main idea is to generate a random vector a with dimension d and elements of this vector is selected independently from s-stable distribution. Given a d-dimensional vector v, whose dot product $\langle a, v\rangle$ gives you a random variable distributed as $(\sum_i |v_i|^s)^{1/s} X$ (i.e., $\|v\|_s X$), where X is a random variable with s-stable distribution. The collection of such random vectors $\langle a, v\rangle$, corresponding to different a's, is known as sketch of vector v and it is used to estimate $\|v\|_s$, refer [55] for more details. This sketch is distributive, i.e., for any $p, q \in \Re^d$, $a.(p-q) = a.p - a.q$.

6.3.1 Hash family

In this technique, hash value to each vector is assigned by calculating dot products $(a.v)$. Since hash functions are locality sensitive, closer points should collide with high probability. Dot product $(a.v)$ is used to project all vector onto real line. We know from s-stability that for any two vectors p and q, distance between their projection $(a.p - a.q)$ is distributed as $\|p-q\|_s X$, where X is s-stable distribution. When real line is cut into equal width segments of appropriate width w, and hash value is assigned according to the segment on which they project onto, then it is clear that the hash function will preserve locality. Each hash functions $h_{a,b}(v) : \mathcal{R}^d \to \mathcal{N}$ will map vector v onto the set of integers. All hash functions are indexed by choosing random a and b, where a as mentioned earlier is d-dimensional vector whose entries are chosen independently from s-stable distribution and b is a real number that is chosen homogeneously from range $[0, w]$. For known a and b, hash function is given by $h_{a,b}(v) = \lfloor \frac{a.v+b}{w} \rfloor$.

For any two vectors p and q, assume $u = \|p-q\|_s$ and let $p(u)$ be the collision probability of p and q for hash function that is chosen uniformly from hash family $\mathcal{H}$.

$$p(u) = P_{a,b}[h_{a,b}(p) = h_{a,b}(q)] = \int_0^w \frac{1}{u} f_s(\frac{t}{u})(1 - \frac{t}{w})dt \tag{6.2}$$

where $f_s(t)$ is probability density function (pdf) of the absolute value of s-stable distribution.

For a fixed value of w, the collision probability decrease monotonically with $u = \|p-q\|_s$. Thus hash function is (R, cR, P_1, P_2)-sensitive for $P_1 = p(1)$ and $P_2 = p(c)$.

The choice of w depends to data set and query point, but according to [43], we obtain good result by setting $w = 4$, so we will use $w = 4$ in our implementation. Refer [43] for detail discussion on choosing optimal value of other parameters.

6.4 Exact Euclidean LSH

Exact Euclidean LSH (E^2LSH) is based on LSH scheme as explained in [43]. The original LSH scheme solves approximate version of R-near neighbor problem, which we call (R, c)-near neighbor problem where it return any point within distance of at most cR from a query point q, if there exists a point in $\mathcal{P}$ within a distance of R from q with constant probability. Whereas E^2LSH solves randomized version of R-near neighbor problem, $(R, 1-\delta)$-near neighbor problem, where δ is a probability that a near neighbor is not reported. In order to solve randomized version of R-near neighbor problem, that is, $(R, 1-\delta)$-near neighbor problem, E^2LSH use the concept of basic LSH scheme to retrieve all near neighbor point and then drops the approximate near neighbor. The running time of this scheme depends on data set $\mathcal{P}$. This scheme is slower for bad data set (i.e., when there are many approximate near neighbor), meaning for any query point q, we have many points clustered outside the ball of radius R centered at q.

Three main parameters responsible for the performance are projection per hash value (K), Number of the hash table (L) and width of the projection (w). The parameter K gives the tradeoff between time for computing hash value and time for pruning false positive. The larger the value of K, the more time is spent for hash computation. Parameter w has a same effect as K, decreasing w will reduce the collision probability for any two points. However, if w is decreased below a certain level then similar data point will fall under different buckets and require L to increase.

6.5 Implementation and Results

Similar to the DTW, Locality sensitive hashing was also conducted in two data sets, RWC and TUT using two types of features CENS and beat synchronous chroma. Here also we need to perform offline processing of database file to extract feature and store it.

Here we need to select appropriate number of the hash table (L) (20 for LSH and 50 for E^2LSH), number of projection per table (K) (24 for LSH and 20 for E^2LSH) and the width of an interval in the projection line (W) = 4.

In order to perform this experiment, feature vectors need to be converted to the column vector. If our query item contains 50 sequences of feature vectors (12 x 50), then it need to be first converted to the column vector (1 x 600). We need to convert database items also to the column vector, each column vector representing 50 sequences of feature vector. This can lead us to the situation where we can have a different type of database.

DataBase1: We can create a database such that first column vector contains

Table 6.1: Result of LSH on RWC database using BSC.

DataBase	**Average**	**Max**	**Min**	**Deviation**
DataBase1	78.44	83	73	10
DataBase2	59.47	74	47	27
DataBase3	46.43	69	33	36

Table 6.2: Result of E^2LSH on RWC database using BSC.

DataBase	**Average**	**Max**	**Min**	**Deviation**
DataBase1	80.17	84	77	7
DataBase2	64.79	77	55	22
DataBase3	50.53	74	38	36

feature from index 1 to 50, second column vector contains feature from index 2 to 51, third from index 3 to 52 and so on.

DataBase2: We can create a database such that first column vector contains feature from index 1 to 50, second column vector contains feature from index 6 to 55, third from index 11 to 61 and so on.

DataBase3: We can create a database such that first column vector contains feature from index 1 to 50, second column vector contains feature from index 11 to 60, third from index 21 to 70 and so on. Since LSH is based on approximate near neighbor search, so its result might differ each time. So this test was repeated 100 times and retrieval rate was averaged for each type of database.

Table 6.1 and 6.2 shows the performance rate of LSH and E^2LSH respectively on above mentioned database using beat synchronous chroma feature. We can see that E^2LSH perform better than LSH in all three types of database.

We also test LSH for different query length on database type I. Table 6.3 shows the performance rate. We can clearly see that performance is almost saturated when the query length is 40. Beyond 40, increase in query length do not make any significant difference in performance, it only just increase processing time. So query size of 40 or 50 would be a good choice.

Although Database1 performs better than rest 2 types of database, in practice for large database one prefer Database3 (as Database1 will have more column vector as compared to Database3). But the performance of Database3 is worst than its counterpart. So we would like to make several sub queries from a query file. For example, suppose we have a query length of 60 beats, and then we would like to have a 10 sub queries using beats 1...50, 2...51, ..., 10...60. Then this will return the best among the 10 matches. We expect that the result should be equivalent to Database1. When performing the same experiment 100 times for all query songs, we found that the result obtained was almost similar to that of Database1 with an average efficiency of 80.72 and 79.89 for LSH and E^2LSH respectively.

Table 6.3: Result of changing query length for LSH on RWC.

Length	Mean	Max	Min	Difference
10	62.15	71	53	18
20	73.63	79	56	14
30	75.90	85	69	16
40	78.34	82	73	9
50	78.69	85	73	12
60	79.63	84	76	8
70	80.33	85	78	7
80	80.44	85	77	8
90	81.12	86	76	10
100	80.87	84	78	6

Table 6.4: Result of changing query length for LSH on TUT database.

Length	Mean	Max	Min	Difference
10	23.50	27.93	2034	7.59
20	31.03	35.17	26.90	8.28
30	33.01	36.90	30.00	6.90
40	33.97	36.90	30.69	6.21
50	34.78	37.59	32.41	5.17
60	35.20	38.62	30.69	7.93
70	34.81	37.24	32.07	5.17
80	35.70	38.28	33.10	5.17
90	34.90	37.59	32.07	5.52
100	34.32	37.93	31.38	6.55

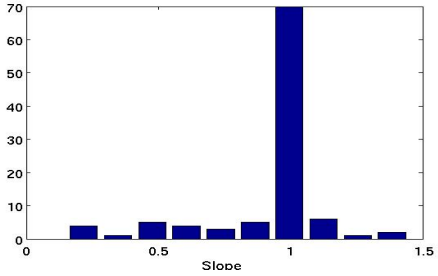
TUT Database

Same set of experiment was also performed in TUT database as well. Table 6.4 shows the retrieval rate for different query length. Here also we can see that the result is almost saturated when the query length is 40.

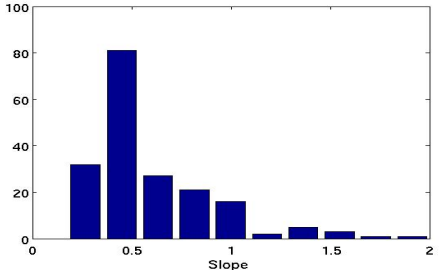
Here we can see that LSH result for TUT dataset is worse as compared to result of RWC database. It is because audio and MIDI of TUT dataset are from different sources and have structural difference. Such structural difference makes matching harder of LSH algorithm.

6.5.1 Slope Histogram as a factor of LSH failures

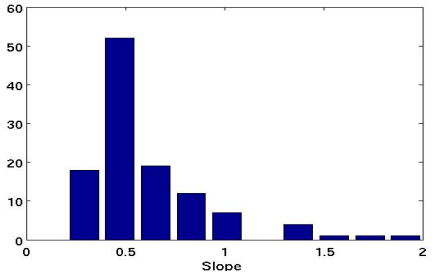
The aim of this experiment was to find out the cause for LSH failure. While observing DTW path, we often encounter a situation where path is vertical/horizontal for a certain distance. Hence we conduct this experiment for finding a section on self distance matrix such that the length of the path is greater than 10 seconds and slope of the path is less than 0.1 (or slope greater than 10 in the opposite case).



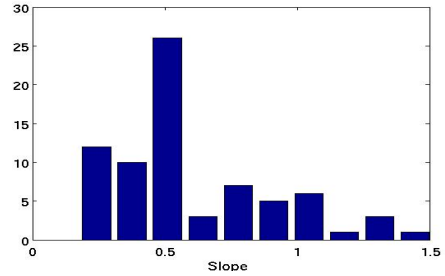
(a) Slope histogram for the case where LSH succeeds.



(b) Slope histogram for the case where LSH fails.

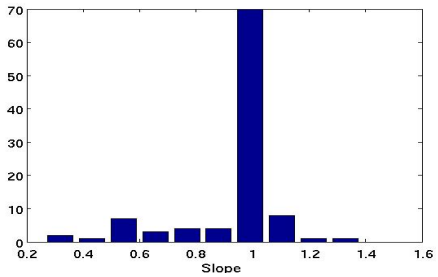


(c) Slope histogram for the case where LSH fails but DTW succeeds.

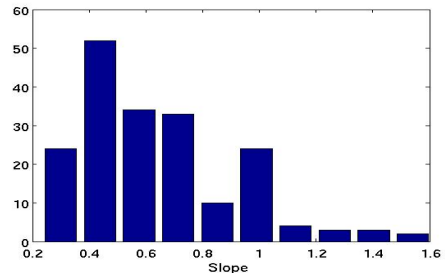


(d) Slope histogram for the case where LSH and DTW both fail.

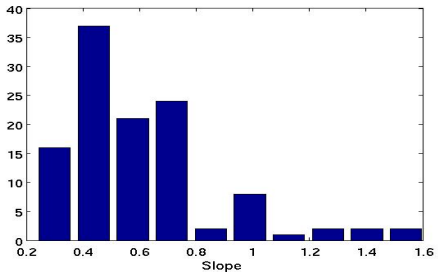
Figure 6.1: Slope histogram of distance matrix for different cases.



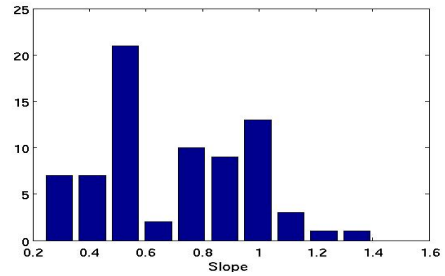
(a) Slope histogram for the case where LSH succeeds.



(b) Slope histogram for the case where LSH fails.



(c) Slope histogram for the case where LSH fails but DTW succeeds.



(d) Slope histogram for the case where LSH and DTW both fail.

Figure 6.2: Slope histogram for different cases by deleting the segments from DTW path whose slope is greater than 10 of less than 0.1.

Among the 101 correct matched songs (by LSH), such case was evident in 28 cases. In 20 songs, above mentioned case was detected for midi axis and in eight songs above mentioned case was detected for audio axis. Among 189 incorrect matched songs, such case appears in 107 cases. In 66 songs, above mentioned case was detected for midi axis and in 41 songs above mentioned case was detected for audio axis. After deleting such segments, overall slope of a DTW path was calculated. Figure 6.1 shows the histogram of the overall slope of DTW and Figure 6.2 shows the overall slope of DTW path after deleting above mentioned cases for all correctly matched songs and incorrectly matched songs respectively.

We can see from Figure 6.1 and 6.2 that LSH works best for the case when slope approach towards unity and fails if its slope deviated from unity.

From Figure 6.1(a) and 6.2(a), we can see that horizontal/vertical segment as mentioned above do not occur for the case where LSH works. Because the height of each block in histogram is almost identical in both figures. So we can say that LSH works well if there is not any vertical/horizontal segment in DTW path. Figure 6.2(b), 6.2(c) and 6.2(d) shows that for all other case such segments were evident. From Figure 6.1(b) and 6.2(b) we can also conclude that LSH mostly fails when slope of DTW path is almost half (or double in opposite case).

7. CONCLUSION AND FUTURE WORK

This section summarizes the thesis with some concluding remarks. This section also discusses possibilities for improvements in the near future.

7.1 Conclusion

In this thesis, we investigated the different issues regarding retrieval of musical score data based on audio query. We discussed the musical features that describe characteristics of any musical segment. The primary goal of this thesis was to propose a system that applies different methods used in music information retrieval algorithms to retrieve musical scores.

Among the main component of this system are feature extraction block and information retrieval algorithms to retrieve similar music in terms of pitch content. A 12-dimensional chroma feature vector represents pitch content of any musical piece and is robust to variation of timbre. Beat sequences were also extracted to address the tempo variation problem. Chroma features and beat sequences were combined in a sophisticated manner for effective representation of a musical piece. These features were processed by two well-known algorithms DTW and LSH to retrieve musical score data that best describes the query audio. Parameters of those algorithms were chosen smartly to enhance the performance of the proposed system.

Our system was implemented in two different datasets RWC dataset (which contain audio and musical score data from the same source) and TUT dataset (which contain audio and musical score data from different sources) and the results illustrate the effectiveness of the system. On RWC dataset, choosing appropriate parameters for DTW, gives you up to 91% retrieval rate, using BSC feature. On TUT dataset, with appropriate parameters retrieval rate was up to 72.76%. Further enhancement can be obtained by up sampling or down sampling beat sequences based on threshold tempo value and the calculating BSC features. This technique increases the retrieval rate from 72.76% to 79.65%. When using a smaller segment of query audio, say approximately about 30 seconds in length, the performance dropped to 40% and 33.44% for RWC and TUT dataset respectively for BSC features.

LSH algorithm is fast as compared to DTW and performs well in a situation where query audio and its corresponding database item are structurally similar. But its performance degrades as query audio and database items are structurally different.

7.2 Future Work

Lastly, this thesis is a small effort towards the field of Music Information Retrieval. There are many possibilities for improvement in the current proposed system. There are few ideas that were left out due to resource and time limitation. One possibility of improvement is to make LSH work for the cases where audio and MIDI are from a different source. One could also try with different features instead of chroma and beat.

Another possibility of improvement is to develop an UI that shows harmonically similar songs in real time while listening to the music.

REFERENCES

[1] A. Klapuri, M. Davy. Signal Processing Methods for Music Transcription. Springer. 2006.

[2] A. P. Klapuri, A. J. Eronen and J.T. Astola. Analysis of the meter of acoustic music signals. IEEE Transactions on Audio, Speech, Language Processing. vol. 14, no. 1, pages 342-355. January 2006.

[3] A. Eronen, A. Klapuri. Musical instrument recognition using cepstral coefficients and temporal features. In IEEE International Conference on Acoustics, Speech, and Signal Processing. Vol. 2, pages 753 - 756, Istanbul. 2000.

[4] A.L.P. Chen, M. Change, J. Chen, J. Shu, S.Y.S. Hua. Query by music segments: an efficient approach for song retrieval. In IEEE International Conference on Multimedia and Expo. Vol. 2, pages 873 - 876, New York. 2000.

[5] A. Ghias, J. Logan, D. Chamberlin, B.C. Smith. Query by humming: musical information retrieval in an audio database. In Proceedings of the third ACM international conference on Multimedia. New York. 1995.

[6] A. Wang. The Shazam music recognition service. Communication of ACM. vol. 49, no. 8. pages 44-48, August 2006.

[7] B. Zhang, J. Shen, Q. Xiang, Y. Wang. CompositeMap: a novel framework for music similarity measure. In proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. Pages 403-410. NY, USA. 2009.

[8] B. Zhang, Q. Xiang, Y. Wang, J. Shen. CompositeMap: a novel music similarity measure for personalized multimodal music search. In proceedings of the 17th ACM international conference on Multimedia. Pages 973-974. NY, USA. 2009.

[9] C. C. Aggarwal, J. L. Wolf, K. Wu and P. S. Yu. Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. Pages 201-212. New York, USA. 1999.

[10] D. P. W. Ellis. Identifying cover songs with Beat-Synchronous Chroma Features. In proceeding of Music Information Retrieval Evaluation and eXchange. 2006.

[11] D. P. W. Ellis. Beat Tracking by Dynamic Programming. Journal of New Music Research, Special Issue on Beat and Tempo Extraction. Volume 36, Issue 1, pages 51-60. March 2007.

[12] D. P. W. Ellis, G. E, Poliner. Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking. IEEE International Conference on Acoustics, Speech and Signal Processing. Honolulu HI. 2007.

[13] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. In Communications of the ACM - Special issue on information filtering. Volume 35, Pages 61-70. December 1992.

[14] D. Byrd and M. Schindele. Prospect for improving OMR with multiple recognizer. In Proceedings of ISMIR. pages 624-630 Victoria, Canada, 2006.

[15] E. Gómez. Tonal Description of Music Audio Signal. PhD thesis. UPF, Barcelona, 2006.

[16] F. Kurth and M. Müller. Efficient index-based audio matching. In IEEE Transactions in Audio, Speech, and Language Processing. vol. 16, no. 2, February 2008.

[17] F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano. An Experimental Comparison of audio tempo induction algorithms. IEEE Trans. Audio, Speech, Language Processing. vol. 14, no. 5, pages 1832-1844, September 2006

[18] G. Peter. Deriving a musical structure from signal analysis for music audio summary generation : "sequence" and "state" approach. In Computer Music Modeling and Retrieval. Volume 2771 of Lecture Notes in Computer Science, Pages 143-166, Springer, Berlin / Heidelberg 2004.

[19] H. Sakoe and S. Chiba. Dynamic programming optimization for spoken word recognition. In Proceedings of IEEE Trans. Acoustic, Speech, Signal (ASSP). pages 57-72, February 1978.

[20] H. Samet. Foundations of Multidimensional and Metric Data Structure. Elsevier 2006.

[21] J.A. Moorer. On the Transcription of musical sound by computer. Computer Music Journal, vol. 1, no. 4, pages 32-38, 1977.

[22] J. Foote. Visualizing music and audio using self-similarity. In Proceedings Of ACM Multimedia, pages 77-80, Orlando Fla. USA, 1999.

[23] J. Paulus and A. Klapuri. Music structure analysis by finding repeated parts. In Proceedings of 1st ACM Audio and Music Computing Multimedia Workshop, pages 59-68, Santa Barbara, California, USA, 2006.

[24] J. Paulus and A. Klapuri. Acoustic features for music piece structure analysis. In Proceedings Of 11th International Conference on Digital Audio Effects, pages 309-312, Espoo, Finland, Sept 2008.

[25] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. In Communications of the ACM. Volume 40 Issue 3, Pages 77-87, March 1997.

[26] J. P. Bello and J. Pickens. A Robust Mid-level Representation For Harmonic Content in Music Signal. In Proceedings of 6th International Symp. Music Information Retrieval, London U.K., pages 304-311. 2005.

[27] J. S. Breese, D. Heckerman, C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Pages 43-52. San Francisco, CA, USA. 1998.

[28] J. S. Downie. The music information retrieval evaluation exchange (2005-2007): A window into music information retrieval research. Acoustical Science and Technology. vol. 29, no. 4, pages 247-255. 2008.

[29] J.l. Bentlet. Multidimensional binary search trees used for associative searching. In Communications of the ACM. vol 18, no. 9, pages 509-517, September 1975.

[30] K. C. Pohlmann. Principle of Digital Audio. McGraw-Hill Professional, 2000.

[31] K. Goldberg, T. Roeder, D. Gupta and C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. Information Retrieval. Volume 4, Issue 2. Pages 133-151. July 2001.

[32] K. Jensen. Multiple Scale Music Segmentation Using Rhythm, Timbre, and Harmony. EURASIP Journal on Applied Signal Processing. NY, USA. January 2007.

[33] L. Rabiner, B. H. Juang. Fundamentals of Speech Recognition. Prentice Hall. 1993.

[34] L. Terveen, W. Hill, B. Amento, D. McDonald and J. Creter. PHOAKS: a system for sharing recommendations. Communications of the ACM. Volume 40 Issue 3, Pages 59-62, New York, USA. March 1997.

[35] L. H. Ungar, D. P. Foster. Clustering Methods for Collaborative Filtering. In workshop on Recommender Systems at the 15th National Conference on Artificial Intelligence. 1998.

[36] M. Goto and Y. Muraoka. Music understanding at beat level - Real-time beta tracking for audio signals. In Proceedings of IJCAL-95 Workshop on Computational Auditory Scene Analysis, pages 68 - 75. 1995.

[37] M. Goto. An Audio-based Real-time Beat Tracking System and its application. In proceedings of International Computer Music Conference (ICMC). 1998.

[38] M. Goto. An Audio-based Real-time Beat Tracking System for Music With or Without Drum-Sounds. Jornal of New Music Research. volume 30, issue 2. 2001.

[39] M. Goto. Development of RWC Music Database. In Proceedings of 18th International Congress on Acoustic (ICA). 2004.

[40] M.A. Bartsch and G.H. Wakefield. Audio thumbnailing of popular music using chroma-based representation. IEEE Transactions on Multimedia, vol. 7, no. 1, pages 96-104, Feb. 2005

[41] M. Levy, M. Sandler, and M. Casey. Extraction of high level musical structure from audio data and its application to thumbnail generation. In Proceedings of IEEE International Conference Acoustic, Speech, Signal Processing (ICASSP), vol 5, pages 13-16, 2006.

[42] M. Ryynänen and A. Klapuri. Polyphonic music transcription using note event modeling. In IEEE Workshop on Application of Signal Processing to Audio and Acoustics, New Platz, USA 2005.

[43] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distribution. In Proceeding of 20th annual symposium on computational geometry. pages 253-262, New York, USA, 2004.

[44] M. A. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes and M. Slaney. Content-Based Music Information Retrieval: Current Directions and Future Challenges. In Proceedings of the IEEE, vol. 96. no. 4, pages 668-696. April 2008.

[45] M. Müller, F. Kurth and M. Clausen. Audio matching via chroma-based statistical features. In Proceedings of the 6th International Conference on Music and Information Retrieval (ISMIR), pages 288-295, 2005.

[46] M. Müller. Information Retrieval for Music and Motion. Springer Verlag 2007.

[47] M. Müller, F. Kurth. In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP), Pages 437-440, Toulouse, France, May 2006.

[48] M. Müller and M. Clausen. Transposition-invariant self-similarity matrices. In Proceedings of 8th International Conference on Music Information Retrieval, pages 47-50, Vienna, Austria, Sept 2007.

[49] M. Müller and S. Ewert. Chroma Toolbox: Matlab Implementation For Extracting Variants of Chroma-Based Audio Features. In Proceedings of the International Society for Music Information Retrieval Conference (ISMIR). 2011.

[50] M. Goto. A chorus-section detecting methods for musical audio signals. In Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing, pages 437-440, Hong-Kong 2003.

[51] N Jiang. An Analysis of Automatic Chord Recognition Procedures for Music Recordings. Saarland University. February 2011.

[52] N. Hu, R.B. Dannenberg and G. Tzanetakis. Polyphonic Audio Matching and Alignment for Music Retrieval. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New York. October 2003.

[53] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM conference on Computer supported cooperative work. Pages 175-186. New York 1994.

[54] P. Indyk and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. In Proceedings of the thirtieth annual ACM Symposium on Theory of Computing. pages 604-613, USA, 1998.

[55] P. Indyk. Stable distribution, pseudorandom generators, embeddings and data stream computation. In Proceedings of 41st Annual Symposium on Foundation of Computer Science, 2000

[56] R. Weber, H. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity Search Methods in High Dimensional Spaces. In Proceedings of the 24th International Conference on Very Large Data Bases, pages 194-205, 1998.

[57] S. Dixon. Automatic Extraction of Tempo and Beat From Expressive Performances. Journal of New Music Research. Volume 30, Issue 1. 2001.

[58] S. Dixon, F. Gouyon and G. Widmer. Towards Characterisation of Music via Rhythmic Patterns. In Proceedings of 5th International Conference Music Information Retrieval, pages 509-517, Barcelona Spain, 2004.

[59] S. Dixon. Evaluation of the Audio Beat Tracking System BeatRoot. Journal of New Music Research. Volume 36, Issue 1, 2007.

[60] T. Jehan. Creating Music by Listening. PhD thesis, Massachusetts Institute of Technology, Boston, Mass, USA, September 2005

[61] U. Shardanand, P. Maes. Social information filtering: algorithms for automating "word of mouth". In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Pages 210-217. New York. 1995.

[62] W. Hill, L. Stead, M. Rosenstein and G. Furnas. Recommending and evaluating choices in a virtual community of use. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Pages 194-201, New York 1995.

[63] Y. Cheng. Music Database Retrieval Based on Spectral Similarity. Technical Report. Stanford. March 2001.

[64] http://www.theguardian.com/technology/2006/jan/17/news.science/ (Last accessed 25-03-2014).

[65] http://en.wikipedia.org/wiki/File:Chopin_Prelude_7.svg. (Last accessed 02-12-2013).

[66] http://staff.aist.go.jp/m.goto/RWC-MDB/ (Last accesses 04-12-2013).

[67] http://ttic.uchicago.edu/ gregory/index.html.