# CONTENT BASED INDEXING OF MUSIC OBJECTS USING APPROXIMATE SEQUENTIAL PATTERNS

D.Vikram[1] and Dr.M.Shashi[2]

[1]SRF(CSIR) and [2]Professor
Department of Computer Science and Systems Engineering
Andhra University, Visakhapatnam, India

## *ABSTRACT*

*The music objects are classified into Monophonic and Polyphonic. In Monophonic there is only one track which is the main melody that leads the song. In Polyphonic objects, there are several tracks that accompany the main melody. Each track is a sequence of notes played simultaneously with other tracks. But, the main melody captures the essence of the music and plays vital role in MIR. The MIR involves representation of main melody as a sequence of notes played, extraction of repeating patterns from it and matching of query sequence with frequent repeating sequential patterns constituting the music object. Repeating patterns are subsequences of notes played time and again in a main melody with possible variations in the notes to a tolerable extent. Similarly, the query sequence meant for retrieving a music object may not contain the repeating patterns of the main melody in its exact form. Hence, extraction of approximate patterns is essential for a MIR system. This paper proposes a novel method of finding approximate repeating patterns for the purpose of MIR. The effectiveness of methodology is tested and found satisfactory on real world data namely 'Raga Surabhi' an Indian Carnatic Music portal.*

## *KEYWORDS*

*MIR, MIDI, Query by Humming, Repeating Patterns.*

## 1. INTRODUCTION

Sequence of data objects maintains an order among its constituents and hence they are found to be suitable to represent data such as DNA sequences, stock market data streams, time series weather/climatic conditions at one or more locations, audio signals, video signals etc. Sequential pattern mining is a specialized field of data mining which focuses on extracting sequential patterns from sequence data repositories. Sequential pattern mining [1] has separate set of techniques to extract repeating pattern from long sequences and frequent sequential patterns from a large collection of shorter sequences of fixed or variable length constituting a sequence data base. This paper focuses on repeating pattern extraction from a single long sequence representing a monophonic music object.

The music objects are represented in three formats:

1. Conventional Music Notation (CMN) ([2],[12]) represents music objects with symbols and time signature and it does not support automated processing as it is only human readable but not machine readable.

DOI : 10.5121/ijdkp.2015.5207

2. Audio file format represents general songs which can be played by CD players and iPods. These files are available in original format as *.wav* file and in compressed format as *.mp3* file.

3. Musical Instrument Digital Interface (MIDI) file format provides event messages about the pitch and intensity, control signals for parameters such as volume, vibrato and panning, cues and clock signals to set the tempo[5]. See Fig.[1] for representation of music files in three formats

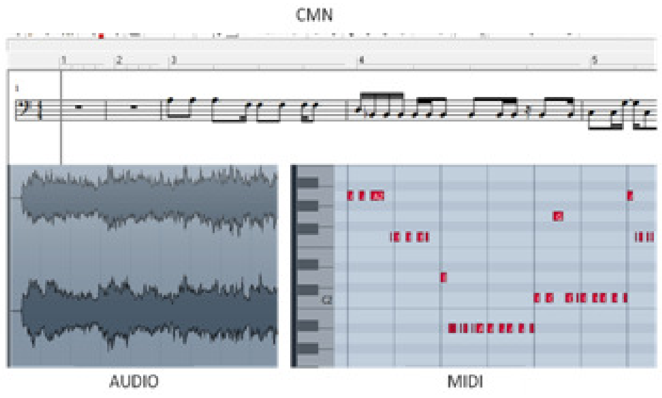


Fig.[1] Representation of music files

The music objects represented in audio and MIDI formats are machine processable and hence becomes amicable for automated retrieval. A song or a piece of music with suitable accompaniment are generally represented as a polyphonic music object [Fig.2a] containing separate tracks for various accompaniment in addition to main melody, as a MIDI file. The main melody [4] contains most of the information pertaining to the music object and hence demands special focus while processing music objects in the context of music information retrieval. The main melody [Fig.2b] is extracted by separating [6] the track representing it from originally polyphonic music object to create a monophonic music object.

The theme of a song is inherently captured by the track representing main melody as it provides data regarding the sequence of notes played at various time stamps along with velocity etc. In the context of music information retrieval in response to Query by Humming (QBH) [7] the note sequence representing the main melody is totally ordered. In other words the notes are strictly ordered because at every time stamp no more than one note is played excluding the accompaniments.

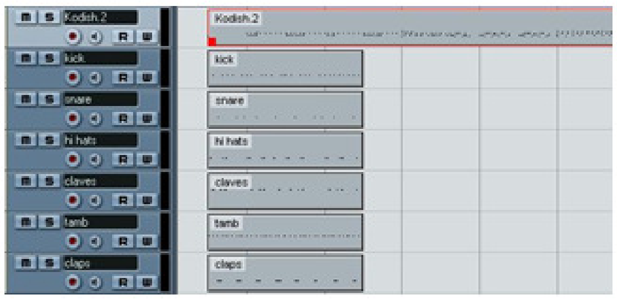


Fig. [2a] MIDI file with multiple tracks as polyphony

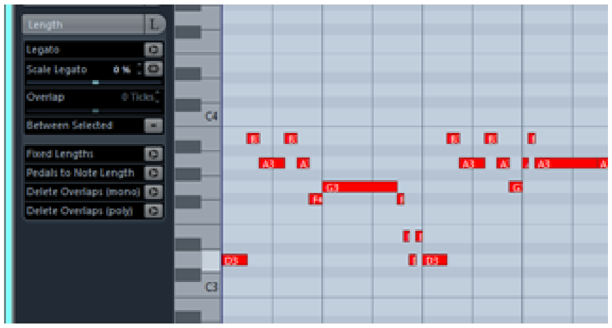


Fig. [2b] MIDI file with separated track as monophony as main melody

This research work aims at extracting features of monophonic music objects for the purpose of indexing them in support of Music Information Retrieval [14]. Specifically, the authors have developed a frame work for representing the main melody of a monophonic music object as a long sequence of notes along with the time stamps and applied sequential mining techniques for extracting repeating patterns allowing tolerance for minor alterations in the notes played which is essential for dealing with real world applications. Sequential patterns with tolerance are referred to as approximate sequential patterns which contain one or more exactly repeating patterns that are joinable as they co-occur close to one another frequently. Hence, mining exactly repeating patterns provides seeds for formation of lengthier approximate sequential patterns with tolerance.

Table [1]: Musical notes for a given string

| MIDI Note Number | 50 | 46 | 46 | 48 | 56 | 50 | 46 | 46 | 50 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|
| English Alphabets | F | B | B | D | L | F | B | B | F | D |
| Musical Notes | D3 | A#2 | A#2 | D3 | G#3 | D3 | A#2 | A#2 | D3 | C3 |

## 2. METHODOLOGY

This project on feature extraction from monophonic music objects is implemented in three phases:

1. Representation of the main melody track as a note sequence [6]
2. Finding maximal exactly repeating patterns in linear time [8]
3. Extracting approximate sequential patterns with tolerance [8]

### 2.1 Representation of main melody as a note sequence:

Monophonic [3] music objects containing the main melody is available as a sequence of MIDI note numbers. Each MIDI note number has a two dimensional symbolic name representing the name of the note and its octave for example MIDI note number 45 is named/referred to as A2 as they represents note A in octave 2. Similarly, the name of the MIDI note number 96 is C7 representing note C in octave 7. Though there are 128 MIDI note numbers the human perception is limited to a sub range of these 128 distinct notes.

Table [2]: Mapping of alphabets with MIDI note number and musical notes for given range

| S. No. | Musical Note | MIDI Note Number | English Alphabet |
|---|---|---|---|
| 1 | A2 | 45 | A |
| 2 | A#2 | 46 | B |
| 3… | B2… | 47… | C… |
| … | … | … | … |
| 25 | A4 | 69 | Y |
| 26 | A#2 | 70 | Z |
| 27 | B4 | 71 | a |
| 28 | C5 | 72 | b |
| 29… | C#5… | 73… | c… |
| … | … | … | … |
| 50 | A#6 | 94 | x |
| 51 | B6 | 95 | y |
| 52 | C7 | 96 | z |

For the purpose of indexing songs/music objects, no song spans over more than three octaves and hence it is possible and convenient to represent each MIDI note number by a single symbol of each English alphabet [A-Z…a-z ] which can represent more than four octaves. Specifically, each of the musical notes starting from A2 (45) to C7 (96) are represented using single symbol starting from A to z for simplicity as listed in Table [2]. Accordingly a music sequence " D3 A#2 A#2 C3 G#3 D3 A#2 A#2 D3 C3 " with MIDI note numbers "50 46 46 48 56 50 46 46 50 48" is represented as "F B B D L F B B F D" to transform it into a string Table [1].

Thus any musical note sequence that is totally ordered can be represented as a string of alphabets. Hence, the data structures and algorithms developed for string processing are directly applicable to music sequences represented as strings.

## 2.2 Finding maximal exactly repeating patterns

Once a music object or a song is represented as a sequence of alphabet in the form of a string, the process of locating maximal exactly repeating sub sequences at different positions of the long sequence proceeds in the second phase.

The [fig. 3] depicts the suffix tree for string *F B B D L F B B F D*. For any non-leaf node '*v*' the number of leaf nodes in the sub-tree routed at '*v*' gives the frequency as well as indexes of the string formed by the concatenation of edge labels along the path to '*v*' referred to as path label of '*v*'.

The suffix tree shown in [Fig. 3] contains a non-leaf node whose path label is '*F*' as it contains 3 leaf nodes in its sub tree. Its frequency is *3* and the indexes of suffixes are 1, 6 and 9 representing *F B B D L F B B F D, FBBFD, FD* respectively. Similarly '*BB*' has 2 leafs and 'FBB' has 2 leafs representing their frequency and location of occurrence in the string S.

Wiener et.al.[9] proposed an efficient algorithm for constructing a suffix tree whose time as well as space complexity is *O(n)*. This research work adapts Wieners algorithm for extracting maximal exactly repeating patterns from a musical note sequence represented as a string. Every non-leaf node of the suffix tree with more than a threshold number $\theta$ of leaf nodes in its sub tree identifies a maximal exactly repeating pattern defined by its path label.

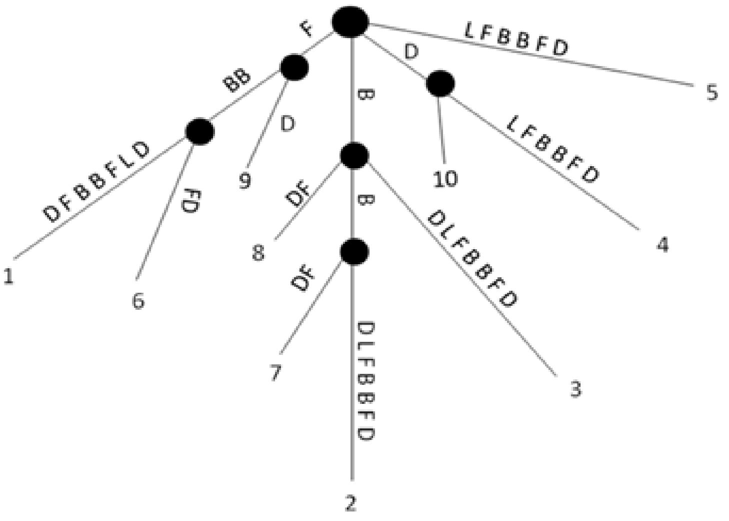


Fig.[3] The suffix tree construction for given string

Suffix tree is an efficient non-linear data structure that supports efficient implementations of many string operations including extraction of exactly repeating maximal substrings from a long sequence/string ([10], [15]). A suffix tree representing a string of length *n* is a rooted directed tree with exactly *n* leaves numbered 1 to *n* representing the location/index of the suffix represented by label of the path ending at the leaf node. Every internal node except for the root node has two or more branches labeled with non empty suffixes with distinct starting character.

A repeating pattern '*P*' of length 'n' has a nearly $2^n$ sub pattern which also repeats with the repeating pattern '*P*'. A maximal repeating pattern is a lengthiest subsequence that repeats in a string frequently and none of its extensions in either direction has equal frequency with it[16].

Considering a frequency threshold of *2*, '*F*' as well as '*FBB*' are considered maximal repeating patterns individually as their frequency is different. While '*FB*' is a subsequence of '*FBB*' which is not considered as maximal repeating pattern as its frequency being same as that of '*FBB*'.

The following algorithm is applied to identify and locate repeated occurrences of maximal repeating patterns in the string. Each repeating pattern '*i'* has a strand defined by an ordered pair $<pat_i, sup\text{-}set_i>$ where, $pat_i$ is the pattern that repeats and $sup\text{-}set_i$ is a list of indexes of the subsequences supporting the $pat_i$.

### 2.2.1 Algorithm for finding exact repeating patterns and their strands:

Input:

Music note sequence represented as string S, minimum frequency threshold $\theta$ and minimum length threshold $l_{min}$.

Output: Strands of exact repeating patterns $<pat_i, sup\text{-}set_i>$
$sup\text{-}set_i$ is a set of locations of repeated occurrences of $pat_i$ in S

```
1.    construct suffix tree for the string S

2.    traverse the tree from root
      i=0
```

```
at every non-leaf node ‘$v$’

 if the path length ($v$) > $l_{min}$
  if leaf node count($v$) > $\theta$
   ($i$=$i$+$1$, $pat_i$ = prefix($v$)
          then store the indexes of
          leafs into $sup\text{-}set_i$)
```

The above algorithm gains its efficiency as it uses suffix tree which is a compressed form of trie. The height of suffix tree is much less than the worst case possible height of a trie which is equal to the length of the string.

## 2.3 Extracting approximate sequential patterns with tolerance:

Phase2 discovers exact repeating patterns that are significant based on user specified length and frequency thresholds. An approximate sequential pattern is a combination of maximal repeating patterns that occur close to one another. Specifically an Approximate sequential Pattern (*AP*) can be expressed as a series of Exact repeating Patterns (*EP*) separated by allowable gap, $G_i$ which is the number of differing characters occurring in between $EP_i$ and $EP_{i+1}$ in the subsequences that support both $EP_i$ and $EP_{i+1}$.

For example, approximate pattern *P= < ‘AB’, 1, ‘BCE’, 2, ‘DA’ >* is a series of three exact patterns; ‘*AB*’ followed by ‘*BCE*’ with a gap of one mismatching characters. The subsequences *‘…ABCBCEEFDA…’* as well as *‘ABABCEABA’* contain the pattern *P* and hence support it.

The length of an approximate pattern is the sum of the lengths of exact patterns and gaps constituting it. The length of the pattern *P* is *10*. The ratio of the number of mismatching characters to the length of approximate pattern should be less than tolerance threshold specified by user. The tolerance threshold is limited in the range of *0* to *0.4*; while ‘*0*’ tolerance imposes stringent matching, ‘*0.4*’ tolerance allows very liberal matching.

The strand of a pattern represents the subsequences supporting the pattern in the form of list of indexes. The strands of multiple exact patterns (constituting an approximate pattern) are carefully merged to form strands of approximate patterns.

Two strands can be merged to form a strand of a lengthier approximate pattern if they contain indexes close to one another on either side within a specified gap. Suppose there are two strands namely $strand_j$ with a pattern $P_j$ of length $l_j$ and $strand_k$ with a pattern $P_k$ of length $l_k$. Inorder to be mergeable an index $i$ in $strand_j$ should have a corresponding index $m$ in $strand_k$ with in a distance of $d_j$ where $d_j$ is equal to $d_j=(1+2\delta)*l_j$ where $\delta$ is error threshold[17]. If $P_k$ occurs after $P_j$ the merged pattern is $<P_j,\ gap,\ P_k>$ otherwise, it is $<P_k,\ gap,\ P_j>$. The following algorithm gives details of merging smaller patterns to form larger approximate patterns and maintaining their strands.

Step1 finds the allowable gap between two patterns based on their lengths and error tolerance $\delta$. Step2 discusses the process of merging patterns and strands in the forward direction while step3 discusses the process of merging patterns and strands in the backward direction. Step 4 increments $j$ to repeat first three steps for extending each $strand_j$ on both sides. The final step screens away infrequent strands based on index counting. The resulting strands may in turn be merged with other strands and the process continues until no new strands can be merged.

### 2.3.1 Algorithm for finding Approximate Patterns:

Input: Original sequence *S*, list of *t* strands of exact repeating patterns *strand[]*, min frequency threshold $\theta$, error threshold $\delta$.

Output: Strands of approximate patterns
Process:

```
1. for each $j=1$ to $t$
   for each $strand_j$ with pattern $P_j$ find $l_j = len(P_j)$,
   $d_j = (1+2\delta) * l_j$

2. Search forward:

`  for $k=j+1$ to $t$
   for each index $i$ in $strand_j$
   for each index $m$ in $strand_k$
   if $(i+l_j) \leq m \leq (i+d_j)$;
   {
   gap = $m-(i+l_j)$
                    create new strand with pattern = <$P_j$, gap, $P_k$>
      insert $i$ into the list of indexes of the new strand
         repeat
          $i$ = next index in the $strand_j$
          $m$ = next index in the $strand_k$
          if ($m-(i+l_j)$= gap);
              {
                append $i$ to the list indexes
                           of new strand
                $i$= next indexes in $strand_j$
              }
              $m$= next index in $strand_k$
   }
   until null.

3. Search backward:
    if ($j=1$), goto step4;
      for $k=j-1$ down to $1$
      for each index $i$ in $strand_j$
      for each index $n$ in $strand_k$
        if(($i-l_k$) > n $\geq$ (i-$d_j$));
        {
          $l_k$=len ($P_k$)
          gap=($i$-($n$+$l_k$))
          insert new pattern = <$P_k$, gap, $P_j$>
          insert $n$ into the list of indexes of new strand
             repeat
               $i$ = next index in the $P_j$ strand
               $n$= next index in $k^{th}$ strand
               if ($i$-($n$+$l_k$)= gap);
                   {
                   append $n$ to the list of indexes of
                                       new strand
                    $i$ = next index in $strand_j$
                    }
                    n=next index in $strand_k$
            }
    until null
4. $j=j+1$
5. count the indexes of each strand and return those with at least $\theta$ frequency.
```

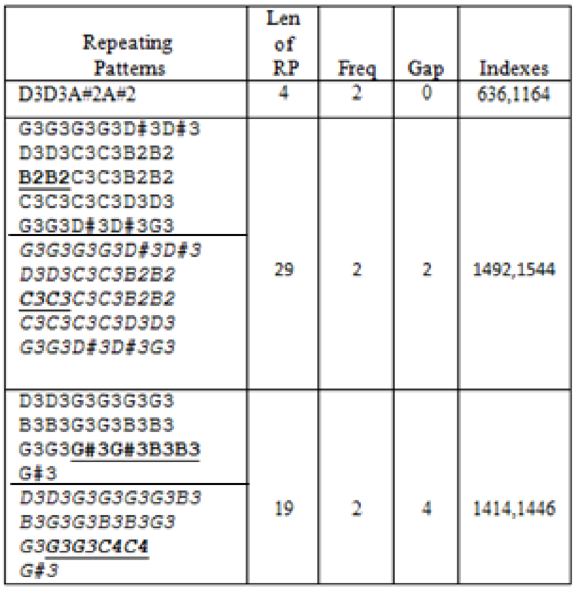
## 3. EXPERIMENTATION AND RESULTS

Raga Surabhi provides a collection of 185 songs belonging to various ragas of Carnatic music in mp3 format. Each song is represented as a note sequence during the preprocessing steps by converting wave files into strings. The length of the songs varies extensively resulting in a range of 238 to 6144 long sequences/strings. The sum of the lengths of all note sequences is 2,28,542.We implemented [2.2.1] and[2.3.1] algorithms and found the number of repeating patterns with user specified minimum length of patterns {2,3,4,5} and frequency θ={2,3,4,5} and error threshold as gaps δ={0.1,0.2,0.3,0.4} as shown in Tables [4,5 and 6] and their graphs in [Fig. 4, and 5]. See annexure [1] for detailed results.

**Example1:**

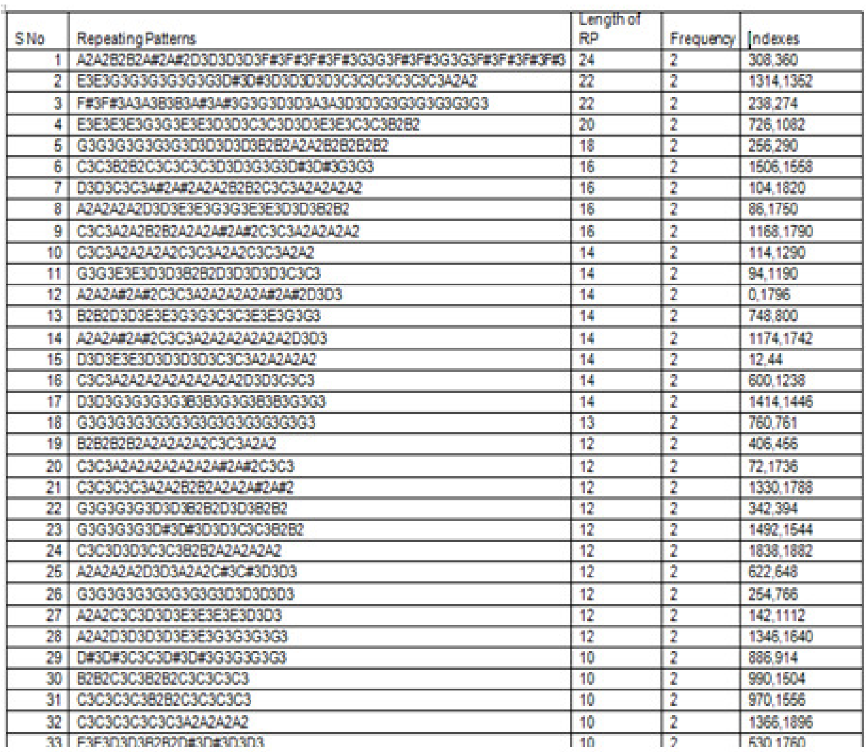
The features identified from the song 'Arabhimanam' are shown in Table.[3a]

Table [3a]: Song 'Arabhimanam' with input length 1906, min len= 2, min freq θ= 2 and error threshold δ=0.3



| Repeating Patterns | Len of RP | Freq | Gap | Indexes |
|---|---|---|---|---|
| D3D3A#2A#2 | 4 | 2 | 0 | 636,1164 |
| G3G3G3G3D#3D#3 D3D3C3C3B2B2 B2B2C3C3B2B2 C3C3C3C3D3D3 G3G3D#3D#3G3 | 29 | 2 | 2 | 1492,1544 |
| *G3G3G3G3D#3D#3 D3D3C3C3B2B2 C3C3C3C3B2B2 C3C3C3C3D3D3 G3G3D#3D#3G3* | | | | |
| D3D3G3G3G3G3 B3B3G3G3B3B3 G3G3G#3G#3B3B3 G#3 | 19 | 2 | 4 | 1414,1446 |
| *D3D3G3G3G3G3B3 B3G3G3B3B3G3 G3G3G3C4C4 G#3* | | | | |

Table [3b] some features of the song "Arabhimanam"



| S No | Repeating Patterns | Length of RP | Frequency | Indexes |
|---|---|---|---|---|
| 1 | A2A2B2B2A#2A#2D3D3D3D3F#3F#3F#3F#3F#3G3G3F#3F#3G3G3F#3F#3F#3F#3 | 24 | 2 | 308,360 |
| 2 | E3E3G3G3G3G3G3G3D#3D#3D3D3D3D3C3C3C3C3C3C3A2A2 | 22 | 2 | 1314,1352 |
| 3 | F#3F#3A3A3B3B3A#3A#3G3G3D3D3A3A3D3D3G3G3G3G3G3G3 | 22 | 2 | 238,274 |
| 4 | E3E3E3E3G3G3E3E3D3D3C3C3D3D3E3E3C3C3B2B2 | 20 | 2 | 726,1082 |
| 5 | G3G3G3G3G3G3D3D3D3D3B2B2A2A2B2B2B2B2 | 18 | 2 | 256,290 |
| 6 | C3C3B2B2C3C3C3C3D3D3G3G3D#3D#3G3G3 | 16 | 2 | 1506,1558 |
| 7 | D3D3C3C3A#2A#2A2A2B2B2C3C3A2A2A2A2 | 16 | 2 | 104,1820 |
| 8 | A2A2A2A2D3D3E3E3G3G3E3E3D3D3B2B2 | 16 | 2 | 86,1750 |
| 9 | C3C3A2A2B2B2A2A2A#2A#2C3C3A2A2A2A2 | 16 | 2 | 1168,1790 |
| 10 | C3C3A2A2A2A2C3C3A2A2C3C3A2A2 | 14 | 2 | 114,1290 |
| 11 | G3G3E3E3D3D3B2B2D3D3D3D3C3C3 | 14 | 2 | 94,1190 |
| 12 | A2A2A#2A#2C3C3A2A2A2A2A#2A#2D3D3 | 14 | 2 | 0,1796 |
| 13 | B2B2D3D3E3E3G3G3C3C3E3E3G3G3 | 14 | 2 | 748,800 |
| 14 | A2A2A#2A#2C3C3A2A2A2A2A2A2D3D3 | 14 | 2 | 1174,1742 |
| 15 | D3D3E3E3D3D3D3D3C3C3A2A2A2A2 | 14 | 2 | 12,44 |
| 16 | C3C3A2A2A2A2A2A2A2A2D3D3C3C3 | 14 | 2 | 600,1238 |
| 17 | D3D3G3G3G3G3B3B3G3G3B3B3G3G3 | 14 | 2 | 1414,1446 |
| 18 | G3G3G3G3G3G3G3G3G3G3G3G3G3 | 13 | 2 | 760,761 |
| 19 | B2B2B2B2A2A2A2A2C3C3A2A2 | 12 | 2 | 406,456 |
| 20 | C3C3A2A2A2A2A2A2A#2A#2C3C3 | 12 | 2 | 72,1736 |
| 21 | C3C3C3C3A2A2B2B2A2A2A#2A#2 | 12 | 2 | 1330,1788 |
| 22 | G3G3G3G3D3D3B2B2D3D3B2B2 | 12 | 2 | 342,394 |
| 23 | G3G3G3G3D#3D#3D3D3C3C3B2B2 | 12 | 2 | 1492,1544 |
| 24 | C3C3D3D3C3C3B2B2A2A2A2A2 | 12 | 2 | 1838,1882 |
| 25 | A2A2A2A2D3D3A2A2C#3C#3D3D3 | 12 | 2 | 622,648 |
| 26 | G3G3G3G3G3G3G3G3D3D3D3D3 | 12 | 2 | 254,766 |
| 27 | A2A2C3C3D3D3E3E3E3E3D3D3 | 12 | 2 | 142,1112 |
| 28 | A2A2D3D3D3D3E3E3G3G3G3G3 | 12 | 2 | 1346,1640 |
| 29 | D#3D#3C3C3D#3D#3G3G3G3G3 | 10 | 2 | 886,914 |
| 30 | B2B2C3C3B2B2C3C3C3C3 | 10 | 2 | 990,1504 |
| 31 | C3C3C3C3B2B2C3C3C3C3 | 10 | 2 | 970,1556 |
| 32 | C3C3C3C3C3C3A2A2A2A2 | 10 | 2 | 1366,1896 |
| 33 | E3E3D3D3B2B2D#3D#3D3D3 | 10 | 2 | 630,1760 |

The experiments were done in the following steps

1. Songs collected from Raga Surabhi [11] which is available in *.mp3* audio file format.
2. The *.mp3* files were converted into *.wav* audio file format.
3. The *.wav* files were converted into *.mid* (MIDI) file format
4. Notes belonging to octaves beyond the selected range are removed as they do not represent main melody and note sequence of each song (within the selected range) is represented as a string of characters and stored as separate file

The above data preparation steps creates a folder of 185 files each consisting of a string representing a song. The memory requirement reduces to a great extent as we apply the data preparation steps as shown in the Table 3b below.

Table [3c]: Memory size for 185 songs in various audio file formats

| S. No. | Type of Songs | Size in Memory |
|---|---|---|
| 1 | Mp3 | 814.83 Mb |
| 2 | Wav | 8925.9 Mb |
| 3 | MIDI | 1268.7 Kb |
| 4 | MIDI with main melody | 1024.94 Kb |

The number of patterns as well as execution time decreases with an increase in minimum support in the form of minimum number of repetitions of a pattern/frequency. The number of repeating patterns increases with increase in error threshold. But the variation is not as significant as in the case of variation of minimum number of repetitions and minimum pattern length. However the variation of number of patterns with minimum length is not as sensitive as that of variation in minimum frequency is shown in Fig. [4 and 5] and input lengths for all songs are shown in Fig [6].

It is also observed that the execution time decreases with decrease in number of patterns irrespective the constraints imposed in terms of min length, min frequency and error threshold as shown in Table [4].

Table [4]

| | δ = 0.2 | | δ = 0.3 | |
|---|---|---|---|---|
| Min. freq Threshold | No. of pat | Execution Time in Sec | No. of pat | Time in Sec |
| 2 | 16898 | 4:00 | 16932 | 4:03 |
| 3 | 11850 | 3:11 | 11856 | 3:28 |
| 4 | 9039 | 2:50 | 9041 | 2:51 |
| 5 | 7285 | 2:33 | 7285 | 2:36 |

**Table 5: Error threshold δ=0.1**

Table [5a]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 2 | 2 | 17124 | 4:32 |
| 2 | 3 | 12168 | 3:33 |
| 2 | 4 | 9505 | 3:01 |
| 2 | 5 | 7723 | 2:42 |

Table [5b]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 3 | 2 | 17035 | 4:20 |
| 3 | 3 | 11955 | 3:29 |
| 3 | 4 | 9244 | 2:54 |
| 3 | 5 | 7422 | 2:38 |

Table [5c]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 4 | 2 | 16810 | 4:19 |
| 4 | 3 | 11846 | 3:29 |
| 4 | 4 | 9038 | 2:42 |
| 4 | 5 | 7285 | 2:30 |

Table [5d]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 5 | 2 | 15806 | 3:59 |
| 5 | 3 | 10459 | 3:08 |
| 5 | 4 | 7623 | 2:20 |
| 5 | 5 | 5780 | 2:07 |

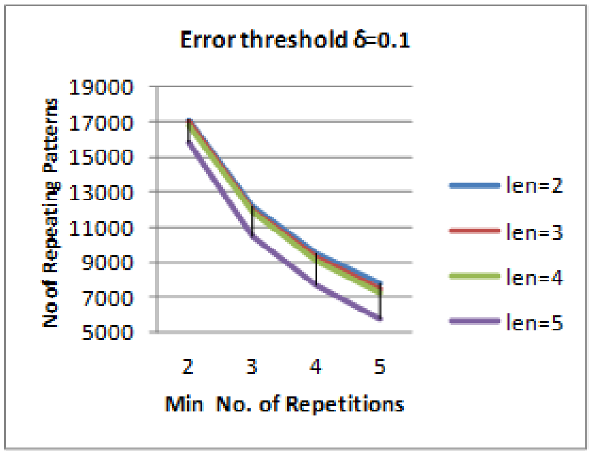


Fig.[4] Variation in length of repeating patterns

**Table 6: Error threshold δ =0.2**

Table [6a]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 2 | 2 | 17213 | 4:08 |
| 3 | 2 | 17123 | 4:05 |
| 4 | 2 | 16898 | 4:00 |
| 5 | 2 | 15890 | 2:06 |

Table [6b]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 2 | 3 | 12172 | 3:16 |
| 3 | 3 | 11959 | 3:14 |
| 4 | 3 | 11850 | 3:11 |
| 5 | 3 | 10463 | 2:54 |

Table [6c]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 2 | 4 | 9056 | 2:54 |
| 3 | 4 | 9245 | 2:52 |
| 4 | 4 | 9039 | 2:50 |
| 5 | 4 | 7624 | 2:21 |

Table [6d]

| Min. length | Min. frequency | No. of patterns | Time |
|---|---|---|---|
| 2 | 5 | 7723 | 2:40 |
| 3 | 5 | 7422 | 2:36 |
| 4 | 5 | 7285 | 2:33 |
| 5 | 5 | 5780 | 2:14 |

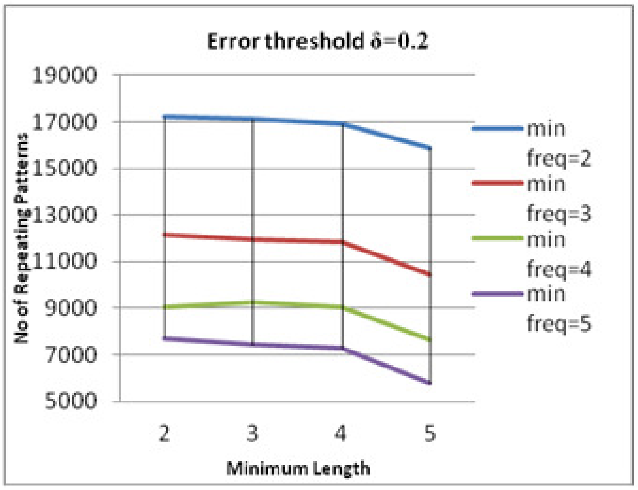


Fig. [5] Variation of frequency (θ = min no of repetitions)

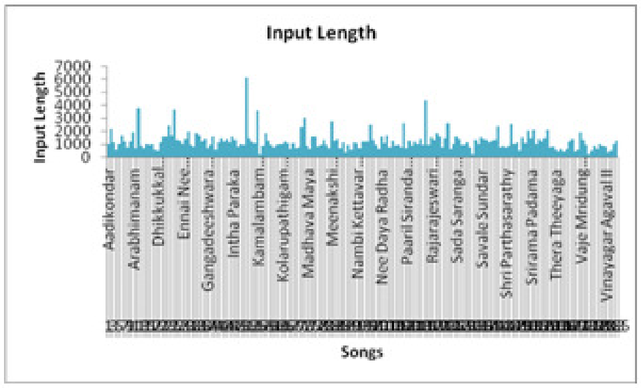


Fig. [6] Input lengths versus songs

## 4. CONCLUSION

The ability to extract approximate sequential patterns from music objects is essential for building an effective/robust Music Information Retrieval System. In this paper, we have developed a frame work that identifies approximate repeating patterns in a given musical sequence as string. We have adapted an algorithm, which finds approximate patterns in a DNA sequence, in our paper. Our algorithm is based on the notion of aggregating a pattern's support set into strands, to achieve efficient computation and compact representation. By combining a suffix-tree-based initial strand mining and iterative strand growth, we adopt a local search optimization technique to reduce time complexity.

## 5. FUTURE WORK

The proposed approach converts the music objects into strings in the most compressed form requiring minimum memory space. Feature extraction in terms of approximate sequential patterns helps in development of effective Content Based Music Information Retrieval Systems which is equally applicable to any type of music.

## ACKNOWLEDGEMENTS

This work was supported by the Council of Scientific & Industrial Research (CSIR) and Andhra University, Visakhapatnam, Andhra Pradesh, India.

## REFERENCES

[1] Mahdi Esmaeili, Fazekas Gabor "Finding Sequential Patterns from Large Sequence Data" IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 1, No. 1, January 2010.

[2] Kyle Adams (Indiana University) "On the Metrical Techniques of Flow in Rap Music" A journal on Criticism, Commentary, Research, and Scholarship. Music Theory online ISSN 1067-3040.

[3] D.Vikram, M.Shashi, B.SatyaSaiVani, V.NagaLakshmi "Music Databases and Data Mining Approaches" Page: 220-227The 2010 International Conference on Data Mining DMIN 2010, July 12-15, 2010 Losvegas Nevada, USA.

[4] Giovanni De Poli, Nicola Orio "Music Information Processing" 2007 Chapter 6, Page 6.21.

[5] Andreas Spanias,Ted Painter, Venkatraman Atti "AUDIO SIGNAL PROCESSING AND CODING" page no 264 chapter 10.2 MIDI VERSUS DIGITAL AUDIO.

[6] Hung-Che Shen, Chungnan Lee "Whistle for music: using melody transcription and approximate string matching for content-based query over a MIDI database" Multimed Tools Appl (2007) 35:259–283.

[7] Justin Salamon, Joan Serrà, Emilia Gómez "Tonal representations for music retrieval: from version identification to query-by-humming" Int J Multimed Info Retr (2013) 2:45–58 DOI 10.1007/s13735-012-0026-0.

[8] Feida Zhu, Xifeng Yan, Jiawei Han, Philip S. Yu "Efficient Discovery of Frequent Approximate Sequential Patterns".

[9] Weiner [Wei73] "Suffix Trees and its Construction"www.cbcb.umd.edu/confcour/Fall2012/suffixtrees.pdf.

[10] Barsky, Marina; Stege, Ulrike; Thomo, Alex; Upton, Chris (2008), "A new method for indexing genomes using on-disk suffix trees", CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management, New York, NY, USA: ACM, pp. 649–658.

[11] Raga Surabhi is a collection of audio files containing raga snippets and songs for the process of understanding and learning Carnatic music. http://www.ragasurabhi.com/

[12] Giovanni De Poli, Nicola Orio "Music Information Processing" 2007 Chapter 6, 6.1.1.3, Page 6.4.

[13] Erdem Unal, Elaine Chew, Panayiotis G Georgiou, Shrikanth S. Narayanan "Challenging Uncertainty in Query By Humming Systems: A Fingerprinting Approach" IEEE Transactions on Audio Speech and Language Processing, Vol.16, No.2 February 2008, Page: 359-371.

[14] Jean-louis Durrieu, 2005400106, "Music Information Retrieval A query-by-humming (QBH) system segmentation of the songs and Approximative melody matching Based on The DTW algorithm", July 2006.

[15] D.Vikram, M.Shashi "Feature Extraction from Monophonic Music Objects using Approximate Sequential Patterns" 3rd International Conference on Communications, Signal Processing Computing and Information Technologies [ICCSPCIT-14] Page 330-337, December 2014 .

[16] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large databases. In Proceedings of the Third SIAM International Conference on Data Mining, 2003.

[17] Y. Kudo and T. Murai. A note on characteristic combination patterns about how to combine objects in object-oriented rough set models. In Third International Conference on Rough Sets and Knowledge Technology (RSKT 2008), pages 115– 123, 2008.

**Annexure [1]**

**Dataset: RAGA SURABHI**

| Test No | Min Len | Min Rep | Error Threshold | Execution Time | No. RP |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 0.0 | 4:34 | 17111 |
| 2 | 2 | 2 | 0.1 | 4:32 | 17124 |
| 3 | 2 | 2 | 0.2 | 4:08 | 17213 |
| 4 | 2 | 2 | 0.3 | 4:13 | 17250 |
| 5 | 2 | 2 | 0.4 | * | * |
| 6 | 2 | 3 | 0.0 | 3:23 | 12166 |
| 7 | 2 | 3 | 0.1 | 3:33 | 12168 |
| 8 | 2 | 3 | 0.2 | 3:16 | 12172 |
| 9 | 2 | 3 | 0.3 | 3:20 | 12178 |
| 10 | 2 | 3 | 0.4 | * | * |
| 11 | 2 | 4 | 0.0 | 2:57 | 9504 |
| 12 | 2 | 4 | 0.1 | 3.01 | 9505 |
| 13 | 2 | 4 | 0.2 | 2:54 | 9056 |
| 14 | 2 | 4 | 0.3 | 3:00 | 9058 |
| 15 | 2 | 4 | 0.4 | * | * |
| 16 | 2 | 5 | 0.0 | 2:49 | 7723 |
| 17 | 2 | 5 | 0.1 | 2:42 | 7723 |
| 18 | 2 | 5 | 0.2 | 2:40 | 7723 |
| 19 | 2 | 5 | 0.3 | 2:47 | 7723 |
| 20 | 2 | 5 | 0.4 | * | * |
| 21 | 3 | 2 | 0.0 | 4:20 | 17022 |
| 22 | 3 | 2 | 0.1 | 4:20 | 17035 |
| 23 | 3 | 2 | 0.2 | 4:05 | 17123 |
| 24 | 3 | 2 | 0.3 | 4:08 | 17158 |
| 25 | 3 | 2 | 0.4 | * | * |
| 26 | 3 | 3 | 0.0 | 3:24 | 11953 |
| 27 | 3 | 3 | 0.1 | 3:29 | 11955 |
| 28 | 3 | 3 | 0.2 | 3:21 | 11959 |
| 29 | 3 | 3 | 0.3 | 3:17 | 11965 |
| 30 | 3 | 3 | 0.4 | * | * |
| 31 | 3 | 4 | 0.0 | 2:46 | 9243 |
| 32 | 3 | 4 | 0.1 | 2:54 | 9244 |
| 33 | 3 | 4 | 0.2 | 2:52 | 9245 |
| 34 | 3 | 4 | 0.3 | 2:56 | 9247 |
| 35 | 3 | 4 | 0.4 | * | * |
| 36 | 3 | 5 | 0.0 | 2:38 | 7422 |
| 37 | 3 | 5 | 0.1 | 2:38 | 7422 |
| 38 | 3 | 5 | 0.2 | 2:36 | 7422 |
| 39 | 3 | 5 | 0.3 | 2:40 | 7422 |
| 40 | 3 | 5 | 0.4 | * | * |
| 41 | 4 | 2 | 0.0 | 4:15 | 16797 |
| 42 | 4 | 2 | 0.1 | 4:19 | 16810 |
| 43 | 4 | 2 | 0.2 | 4:00 | 16898 |
| 44 | 4 | 2 | 0.3 | 4:03 | 16932 |
| 45 | 4 | 2 | 0.4 | * | * |
| 46 | 4 | 3 | 0.0 | 3:08 | 11844 |
| 47 | 4 | 3 | 0.1 | 3:29 | 11846 |
| 48 | 4 | 3 | 0.2 | 3:11 | 11850 |
| 49 | 4 | 3 | 0.3 | 3:28 | 11856 |
| 50 | 4 | 3 | 0.4 | * | * |

| | | | | | |
|---|---|---|---|---|---|
| 51 | 4 | 4 | 0.0 | 2:36 | 9037 |
| 52 | 4 | 4 | 0.1 | 2:42 | 9038 |
| 53 | 4 | 4 | 0.2 | 2:50 | 9039 |
| 54 | 4 | 4 | 0.3 | 2:51 | 9041 |
| 55 | 4 | 4 | 0.4 | * | * |
| 56 | 4 | 5 | 0.0 | 2:33 | 7285 |
| 57 | 4 | 5 | 0.1 | 2:30 | 7285 |
| 58 | 4 | 5 | 0.2 | 2:33 | 7285 |
| 59 | 4 | 5 | 0.3 | 2:36 | 7285 |
| 60 | 4 | 5 | 0.4 | * | * |
| 61 | 5 | 2 | 0.0 | 3:58 | 15793 |
| 62 | 5 | 2 | 0.1 | 3:59 | 15806 |
| 63 | 5 | 2 | 0.2 | 4:17 | 15890 |
| 64 | 5 | 2 | 0.3 | 3:53 | 15923 |
| 65 | 5 | 2 | 0.4 | * | * |
| 66 | 5 | 3 | 0.0 | 2:56 | 10457 |
| 67 | 5 | 3 | 0.1 | 3:08 | 10459 |
| 68 | 5 | 3 | 0.2 | 2:54 | 10463 |
| 69 | 5 | 3 | 0.3 | 2:58 | 10468 |
| 70 | 5 | 3 | 0.4 | * | * |
| 71 | 5 | 4 | 0.0 | 2:27 | 7622 |
| 72 | 5 | 4 | 0.1 | 2:20 | 7623 |
| 73 | 5 | 4 | 0.2 | 2:21 | 7624 |
| 74 | 5 | 4 | 0.3 | 2:25 | 7626 |
| 75 | 5 | 4 | 0.4 | 2:25 | 7628 |
| 76 | 5 | 5 | 0.0 | 2:02 | 5780 |
| 77 | 5 | 5 | 0.1 | 2:07 | 5780 |
| 78 | 5 | 5 | 0.2 | 2:14 | 5780 |
| 79 | 5 | 5 | 0.3 | 2:12 | 5780 |
| 80 | 5 | 5 | 0.4 | 2:19 | 5781 |

*The result not obtained due to more approximation (error threshold= $\delta$).