Finding Exact and Approximate Repeating Patterns to Build Database for Content-Based Music Information Retrieval in Carnatic Music

[1]G Sai Naresh,[2]Archana Raghuvamshi,[3]Dara Vikram
[1]M.tech (CST), Assistant professor, Assistant professor
[1,2]Adikavi Nannaya University,[3]Andhra University

Abstract -- In music information retrieval (MIR) the task is to find the relevant musical files based on a query given in the form of index terms. Existing Music Retrieval System uses metadata like singer name, movie name, album name etc. as index terms. For better storage and retrieval, music must be represented in digital form. The problem of converting raw audio to symbolic representation has a great significance. It is possible to extract a sequence of symbols taken from an alphabet corresponding to attributes of the music such as the duration and the pitch of a note available from its MIDI representation. The pitch of the notes is more relevant for MIR purposes. The proposed system is aimed at retrieval of music files based on its content instead of metadata. This requires indexing (data) of music files in terms of frequently occurring patterns represented in terms of note sequences. The database of such a sophisticated Music Retrieval System contains music files represented as note sequences. Hence, it is essential to develop an algorithm to convert (automatically) midi file into corresponding note sequences.

Keywords— MIDI, MIR, QBH, CBMIR

I. INTRODUCTION

The music is an emotional phenomenon and it has no other purpose than to inspire in our feelings of happiness, melancholy, contentedness, excitement, relaxation and other emotions. The music is a fascinating combination of art and science. Music has always been present in the lives of human beings, both individually and socially, through the cultural, professional, leisure or religious aspects of life. Music is a way by which composers express their innermost feelings. A musical tone is characterized by its duration, pitch, intensity (or loudness), and timbre (or quality). The notes used in music can be more complex than musical tones, as they may include periodic aspects, such as attack transients, vibrato, and envelope modulation [4].

1.1 Music Classification

Based on the amount of concurrency present music is divided into three categories 16. They are Monophonic, Homophonic, polyphonic. 1. Monophonic: music in which only one note sounds at a time. 2. Homophonic: music in which multiple notes may sound at once - but all notes start and finish at the same time.

The left hand of a piano performance, or folk guitar performance, is often homophonic - producing chords rather than a series of individual notes. 3. Polyphonic: the most general form of music, in which multiple notes may sound independent of each other 15, 17.

1.2 Representation of Music Objects

The music objects are represented in three formats as shown in Fig.1. Such as

1. *Conventional Music Notation (CMN)* [2] which 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.

2. The *audio file format* which represents general songs digital form which can be played by CD players and iPods. These files are available in original format as a.*wav*file and in compressed format as *.mp3* file. 3. *Musical Instrument Digital Interface (MIDI) file format* which Provides event messages 5 about the pitch and intensity, control signals for parameters such as volume, vibrato, and panning, cues and clock signals to set the tempo Fig.1 depicts arepresentation of music files in three formats 12.

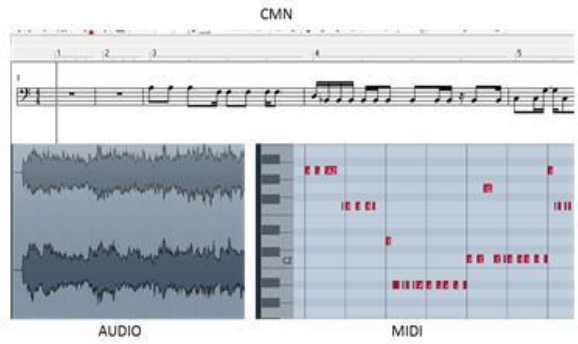


Fig.1 Representation of Music files

The main melody of music object is often captured by the pitch/frequency which is represented in terms of MIDI note number which intern represents a semitone in an octave.

The musical frequencies are divided into 11 octaves numbered from '-1' to '9' each containing 12 semitones named 'C C# D D# E F F# G G# AA# B'. The range frequencies encompassed by an octave doubles as you go for higher octaves successively [3]. The name of the note reflects the semitone and the octave like 'A4' represents semitone 'A' in 4th octaveand it has a distinct MIDI note number '60' as shown Fig.2 and Fig.3.

The MIDI note number is determined by the following formula [6] as shown in Fig.2 that transforms hummed notes into the representation of MIDI values (semitones):

$$\text{MIDI value} = 69 + \left[12 \times \log_2\left(\frac{\text{freq}}{440}\right)\right].$$

Fig.2 Finding MIDI Note Number

where *freq* is the frequency of hummed note and the operator '[ ]' calculates the nearest integer value, 12 leads to the classic dodecaphonic musical scale, and 69 is the MIDI note number Fig.[4] that corresponds to central A with apitch equal to 440 Hz. By convention middle C (MIDI note Number 60) is C4. A MIDI note number of 69 is used for A440 tuning, that is the note A above middle C. The octave representation has shown in Fig.3.

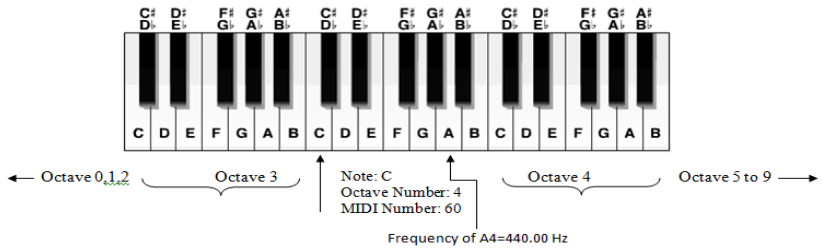
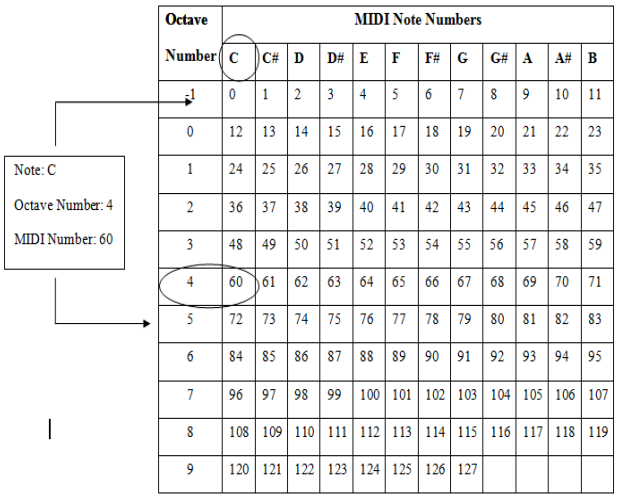


Fig.3 Frequency for A4 in 4th Octave



Octave Number	MIDI Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Fig.4 MIDI Note Numbers

1.3 The Indian Music

The Indian Music is often composed with raga orientation. A raga is based on a scale with a given set of notes, a typical order in which they appear in melodies, and characteristic musical motifs. All compositions and artiste's improvisations that we hear in concert platforms are all raga based. The basic components of a raga can be written down in the form of a scale differing in ascent and descent referred to as arohanam and avarohanam.There are several hundred ragas in present use, and thousands are possible in theory. The classification of ragas plays a major role in Indian music theory.

In north India *(Hindustani music)*, ragas are classified according to such characteristics as mood, season, and time, whereas in south India *(Carnatic music)*, ragas are grouped by the technical traits of their scales. There are 72 melakartha ragas and they are categorized into 12 chakras namely*Indu, Netra, Agni, Veda, Bana, Rutu, Rishi, Vasu, Brahma, Disi, Rudra, Aditya.* The two systems may use different names for similar ragas or the same name for different ragas. The 12 semitones are shown in Figure 5 with the semitones of amusical keyboard.

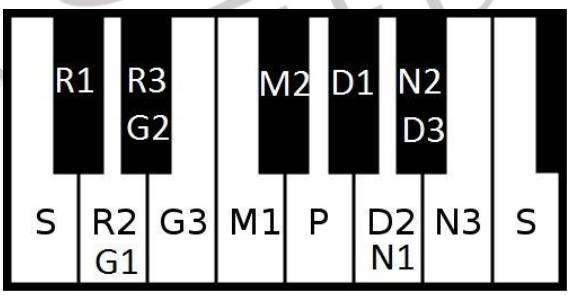


Fig.5 Musical Notes in Carnatic Music

1.4 Dwadasa Swarasthanas

Out of the seven swaras, Shadjam (Sa) and Panchamam (Pa) are constant. Theyare called AchalaSwaras. The remaining five swaras admit varieties and they are called ChalaSwaras. The combination, both Achala and Chalaswaras yield 12 different musical notes and they are called DwadhasaSwarasthanas.

The DwadasaSwarasthanas are:

1. Shadjam.....Sa
2. SuddhaRishabam.....Su Ri
3. ChatusruthiRishabam.....Cha Ri
4. SadharanaGandharam.....Sa Ga
5. AntharaGhandaram.....An Ga
6. SuddhaMadhyamam.....Su Ma
7. PrathiMadhyamam.....Pra Ma
8. Panchamam.....Pa
9. SuddhaDhaivatham.....Su Dha
10. ChatusruthiDhaivatham.....Cha Dha
11. KaisikiNishadham.....Kai Ni
12. KakaliNishadham.....Ka Ni2

Arohanais the series of Swaras in the ascending order of pitch.Avarohanais the series of swaras in the descending order of pitch.Moorchana is Arohana and Avarohana together.

Semitones	Western Notes	Names of Western Notes	Hindustani	Carnatic
C	Do	TONIC	S	S
C#/Db			r	R1
D	Re	SUPERTONIC	R	R2
D#/Eb			g	G2
E	Mi	MEDIANT	G	G3
F	Fa	SUBDOMINANT	M	M1
F#/Gb			m	M2
G	So	DOMINANT	P	P
G#/Ab			d	D1
A	La	SUBMEDIANT	D	D2
A#/Bb			n	N2
B	Ti	LEADING TONE	N	N3

Table 1 Semitones in Western, Hindustani and Carnatic music

Music is universal and relies on the same musical frequencies with different naming for semitones representing a musical frequency in different conventions. The correspondence between the naming of the semitones in Western, Hindustani and Carnatic music systems is tabulated in Table1.

II METHODOLOGY

A framework is developed to extract approximate sequential patterns from a music sequence. The implementation is tested on the database created and found a number of repeating patterns and the time taken for extraction to establish the correctness and scalability of the framework. 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
2. Finding maximal exactly repeating patterns in linear time
3. Extracting approximate sequential patterns with tolerance

2.1 Representation of the main melody track as a note sequence

The theme of a song is inherently captured by the track representing themain melody as it provides data regarding the sequence of notes played at various time stamps along with velocity etc. In the context of

monophonic music information retrieval in response to Query by Humming (QBH) 7, 13the note sequence representing the main melody is totally ordered. In other words, the notes are strictly ordered because at everytime stamp no more than one note is played excluding the accompaniments. Hence a music object can be treated as a string of characters and string matching techniques are applied in Content-based MIR system.

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 homophonic music object containing separate tracks for various accompaniment in addition to themain melody, as a MIDI file. The main melody contains most of the information pertaining to the identification of the music object and hence demands special focus while processing music objects in the context of music information retrieval. The main melody is extracted by separating the track representing it from originally homophonic music object to create a monophonic music object. The midi notes corresponding to the musical notes has shown in Table 2.

Table 2 Musical Notes with MIDI Note Numbers

MIDI Note Number	50	46	46	48	56	50	46	46	50	48
Musical Notes	D3	A#2	A#2	D3	G#3	D3	A#2	A#2	D3	C3

Monophonic 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 represent 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 is depicted in Table 3.

Table 3: 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 [2]. 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 thealphabet in the form of a string, the process of locating maximal exactly repeating sub-sequences at different positions of the long sequence [1] proceeds to the second phase.

The Fig. 6 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 rooted 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*'.

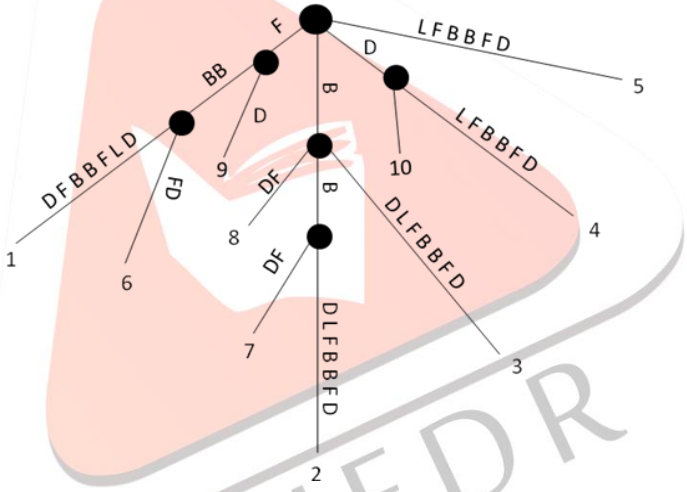


Fig.6The suffix tree construction for given string

The suffix tree is shown in Fig. 6 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 θ of leaf nodes in its sub tree identifies a maximal exactly repeating pattern defined by its path label.

The 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.

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 thelabel 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^nsub-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 anequal frequency with it.

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 [10] 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 θ, 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 ofpat_i in S

1. construct suffix tree for the string S

2. traverse the tree from theroot
   i=0
   at every non-leaf node '*v*'
   if the path length (*v*) >l_{min}
   if leaf node count(*v*) >θ
   (*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 thetrie. 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*) [8], [5] can be expressed as a series of Exact repeating Patterns (*EP*) separated by anallowable 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 character. 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 theapproximate pattern should be less than tolerance threshold specified by theuser. 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 alist 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 of 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 δ is error threshold. 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 δ. 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, alist of t strands of exact repeating patterns $strand[]$, min frequency threshold θ, error threshold δ.

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$(i+l_j) \le m \le (i+d_j)$;
   {
   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 ≥ (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 *newstrand*
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 θ frequency.

III. EXPERIMENTATION AND RESULTS

Raga Surabhi [7] provides a collection of 383 Carnatic songs and 238 ragalapana, arohana, avarohana Carnatic ragas of total 621 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. We have observed two data sets such as Carnatic songs and Carnatic ragaalapana.

3.1 Carnatic Songs

The length of the songs varies extensively resulting in a range of 14 to 6144 long sequences/strings. The sum of the lengths of all note sequences is 2,67,530.

3.2 Carnatic Ragaalapana

The length of the songs (ragalapana, arohana, avarohana) varies extensively resulting in a range of 34 to 234 long sequences/strings. The sum of the lengths of all note sequences is 24920.

.

We implemented [2.2.1] and[2.3.1] algorithms and found the number of repeating patterns with user specified theminimumlength 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]. Annexure [2, 3, 4, 5] shows the detailed results.

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

3.3 Experiments on Carnatic Songs Dataset

The following Tables [4.a, 4.b, 4.c, 4.d] shows that various inputs given as minimum length (Min Len) Minimum number of each pattern repeated called frequency (Min Rep) and Error Threshold ()the Carnatic Songs Collection of 383 Songs and the output shows number of repeating patterns obtained and the execution time is taken for each test.

Table 4.a

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
1	2	2	0.0	24.5	20145
2	2	2	0.1	24.5	20162
3	2	2	0.2	25.3	20327
4	2	2	0.3	26.5	20341

Table 4.b

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
5	2	3	0.0	14.15	14448
6	2	3	0.1	12.38	14450
7	2	3	0.2	12.45	14456
8	2	3	0.3	12.39	14465

Table 4.c

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
9	2	4	0.0		11410
10	2	4	0.1	12.56	11433
11	2	4	0.2	10.27	11436
12	2	4	0.3	10.32	11439

Table 4.d

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
13	3	2	0.0	18.8	19992
14	3	2	0.1	19.15	20009
15	3	2	0.2	18.11	20122
16	3	2	0.3	19.31	20185

3.4 Experiments on Carnatic Ragalapana Dataset:

The following Tables [5.a, 5.b, 5.c, 5.d] shows that various inputs given as minimum length (Min Len) Minimum number of each pattern repeated called frequency (Min Rep) and Error Threshold ()the Carnatic Songs Collection of 238 Songs and the output shows number of repeating patterns obtained and the execution time is taken for each test.

Table 5.a

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
1	2	2	0.1	2.37	2433
2	2	2	0.2	2.40	2452
3	2	2	0.3	3.14	2648
4	2	2	0.4	2.37	2488

Table 5.b

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
5	2	3	0.1	2.12	1797
6	2	3	0.2	2.48	1797
7	2	3	0.3	2.12	1798
8	2	3	0.4	2.41	1803

Table 5.c

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
9	2	4	0.1	2.5	1476
10	2	4	0.2	2.35	1476
11	2	4	0.3	1.47	1477
12	2	4	0.4	1.49	1479

Table 5.d

Test No	Min Len	Min Rep	Error Threshold	Execution Time	No. RP
13	3	2	0.1	2.30	2313
14	3	2	0.2	2.15	2331
15	3	2	0.3	2.33	2347
16	3	2	0.4	2.38	2365

The input has given to framework and observed the change in a number of approximated patterns obtained. When the minimum length and minimum frequency is raised and the error threshold remains constant then number of obtained patterns decreased as shown in Fig7. The minimum length and minimum frequency remain constant and the error threshold is raised then obtained thenumber of patterns also raised which shows that the system allows the error tolerance as shown in Fig.8 The approximation is essential because the user may not sing query input as the original song composed it may be vary due to noise or scale.

Min Len	Min Rep = θ	No. RP
2	2	20341
2	3	14465
2	4	11439

Observation 1: When the frequency increased the obtained number of patterns will decrease

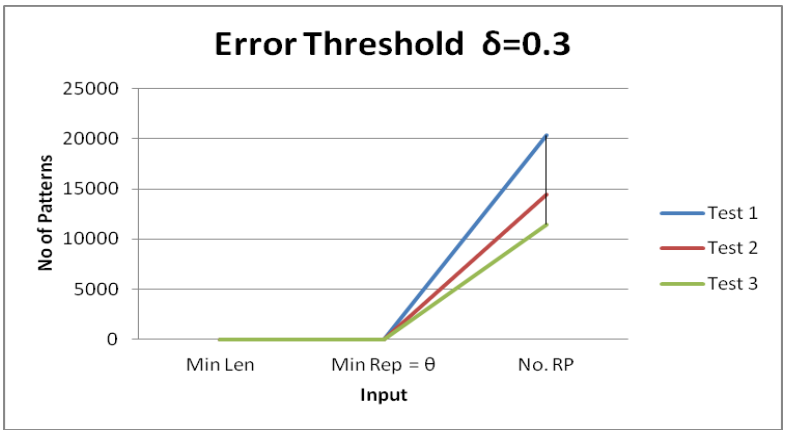


Fig. 7

Min Len	Error Threshold =δ	No. RP
2	0	20145
2	0.1	20162
2	0.2	20327
2	0.3	20341

Observation 2: When the error threshold value δ is increased then obtained number of patterns also increased means the patterns depends on the allowable tolerance

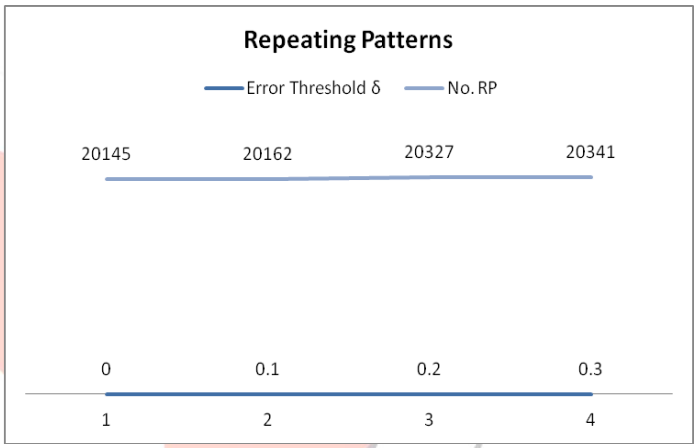


Fig. 8

IV. CONCLUSION

The ability to extractapproximate sequential patterns from music objects is essential for building an effective/robust Music Information Retrieval System.In this paper, we have developed a framework that identifies approximate repeating patterns in a given musical sequence as astring. 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.

V. ACKNOWLEDGEMENTS

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

VI. REFERENCES

[1] Mahdi Esmaeili, Fazekas Gabor "Finding Sequential Patterns from Large Sequence Data" IJCSIInternational 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 journalon Criticism, Commentary, Research, and Scholarship. Music Theory online ISSN 1067-3040.

[3] D.Vikram, M.Shashi, B.SatyaSaiVani, V.NagaLakshmi "Music Databases and Data MiningApproaches" 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] Jean-louisDurrieu, 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.

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

[7] 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.

[8] Feida Zhu,Xifeng Yan,Jiawei Han,Philip S. Yu "Efficient Discovery of Frequent ApproximateSequential 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 forindexing genomes using on-disk suffix trees", *CIKM '08: Proceedings of the 17th ACMConference on Information and Knowledge Management*, New York, NY, USA: ACM,pp. 649–658.