TITLE: **ASTUTE SONG HUNTER: PLAYBACK FILE RETRIEVAL FROM MEDIA SERVERS BASED ON HUMMING PATTERNS**

THEME: **MULTIMEDIA AND GAMING**

Guided By: **R.Ashok Kumar,** email: professorashok@gmail.com

Presented By,

MADHUSUDAN.C.S, email: madhusudancs@gmail.com

PUNEETH BHAT.A.H, email: puneeth.bhat@gmail.com

SANTOSH G VATTAM, email: vattam.santosh@gmail.com

INSTITUTE: **B.M.S COLLEGE OF ENGINEERING,**
POST BOX NUMBER 1908, BULL TEMPLE ROAD,
BANGALORE – 560019

EMAIL: principal@bmsce.ac.in

ASTUTE SONG HUNTER:
PLAYBACK FILE RETRIEVAL FROM MEDIA SERVERS BASED ON HUMMING PATTERNS

1. Introduction:

"**Music expresses that which cannot be said and on which it is impossible to be silent."**
--Victor Hugo.

Music is an integral and indispensable part of everyone's life. Music is one thing that all people irrespective of their language, color, race and nationality can relate to and enjoy. A brief browsing of the internet is enough to understand that the amount of music on the internet is simply overwhelming to say the least. Search and retrieval of songs from remote multimedia servers is performed mainly based on song keywords like the name of the song, the artist or the movie the songs belong to. Various music websites, such as mp3.com, esnips.com etc., use this method of searching based on keywords. Although this method is widely used and accepted, it depends exclusively on the keywords and is completely ignorant of the actual song file contents. This has led to the need for a more intelligent and a more content sensitive search method. And the solution is to implement song search based on humming the tune of the song.

Music cannot be stored on the computer as an analog signal. It has to be digitized before being stored on a computer. Digitized music is represented on the computer in two ways. One way is to separate the various characteristics of musical data such as the notes, the pitch, the duration, the strength and so on and store them as individual components. This representation is called storage based on musical scores. MIDI – Musical Instrument Digital Interface, is one such method of storage based on musical scores. The other way is to store the music as raw data. In this method the intensity of the audio is recorded as a function of time and then sampled at a particular frequency depending on the medium of storage and stored. Recording raw data makes the audio files unmanageably large. Hence the audio is generally compressed and then stored. This method is called storage based on acoustic signals. ".wav" is an example of this representation (here the audio file is stored without compression). MP3 files are also stored in the same manner but they are compressed to remove all the sounds that the human ear cannot perceive.

Any audio signal has the following important characteristics, namely:

- **Frequency:** The number cycles of the wave per second. Usually measured in kHz or MHz.
- **Pitch:** Represents the perceived fundamental frequency of a sound. It is one of the three major auditory attributes of sound.
- **Intensity:** It is defined as the sound power per unit area.
- **Loudness:** It is the quality of a sound that is the primary psychological correlate of physical strength (amplitude).
- **Sound Spectrum:** It is a representation of a sound - usually a short sample of a sound - in terms of the amount of vibration at each individual frequency.

As said earlier in order to store audio on computers, the audio must be digitized. Here we will look into the basics of digitization and then processing these digital signals. Digitization of audio signals is called Analog to Digital Conversion (ADC). This is done in two steps, viz., "discretization" and "quantization". Once the signal is digitized, it needs to be processed. This process is called Digital Signal Processing (DSP). The following are the domains of digital signal processing: Time Domain, Frequency Domain, Spatial Domain,

Auto Correlation Domain, and Wavelet Domain. The domain in which to process the signal is decided by making an informed guess as to which domain represents the signal characteristics in the best possible manner.
Table 1, shows various file formats used to store audio on multimedia servers and their characteristics:

Audio File Format	File Extension	Developed By	Applications	characteristics	Limitations
MIDI – Musical Instrument Digital Interface	.midi	IEEE, IETF and ISO	Used in amateur and professional music making.	Extremely small and the characteristics are stored separately	Limited amplitude of music files.
Real Audio	.rm or .ram	Real Netwoks	Internet radio	Low bandwidth usage.	Reduction in audio quality.
WAV	.wav	IBM and Microsoft	Used for Cd's and broadcast.	Uncompressed, Loss less format	Large file size.
MP3	.mp3	MPEG – Moving Pictures Experts Group	Most popular storage format.	Lossy compression, compression ratio as low as 10% with almost CD quality.	Not a good format for looping.
WMA – Windows Media Audio	.wma	Microsoft	Supported well on most players.	Compares in quality to mp3, delivered as a continuous flow of data.	Not supported on iPods.

Table 1: Audio formats and their properties

2. Objective:

In this paper we present a mathematical approach for searching a multimedia song database based on the song content. Here we implement an algorithm to search a song in a database based on humming. This is achieved by exploiting the fact that the intensity variations in the original song and the hummed tune have similarities. We will demonstrate, using this search algorithm, the following facts:

- Faster search implementation when compared to all other content based search methods.
- Exhaustive and efficient search implementation – any part of a song can be searched using pre-calculated intensity variations without searching the entire song file.

3. Review of Literature:

We analyzed some of the methodologies, approaches and algorithms that have been proposed already through various publications and research papers. Here is a brief summary of the important ones we analyzed. McNab, Smith et al. [McNab-1] propose a methodology which is based on a standard musical scale of pitches based on octave. It uses only songs in the MIDI format, which represents music on the same scale. It then uses a time domain pitch tracking algorithm and constructs notes based on amplitude and pitches. Further they perform approximate string matching on this intermediate representation of the queried hum and the database of songs. That this methodology is proven only against two particular type of song sets they have chosen for experiments, namely Essen and Digital Tradition folksong is the biggest drawback. Another drawback is that it supports only MIDI representation of the songs.

An M.S thesis by Wei Chai[Chai-2] again works only on the MIDI files. It proposes a methodology where in an intermediate <T, P, B> representation of both the database of songs and the query is constructed and an approximate matching is again performed. Here T is the time signature of the song, P is the pitch contour vector and B is the absolute beat number. It additionally uses a Q vector to represent different contour resolutions and quantization boundaries. It then does string matching based on Dynamic Programming method in 2 varied steps both of which basically measures the degree of beat similarity. The <T, P, B> representation is constructed by Note Segmentation, Pitch tracking and and Beat tracking.

A paper by Cheng Yang[Yang-3] again works only for MIDI file formats. The methodology proposed here constructs Intermediate Data based on Instantaneous power function of time. It identifies local maximum value within a neighbourhood of a fixed size in the above function and takes 180 samples of frequency components near each peak. This results in n spectral vectors of 180 dimensions each, where n is the number of peaks obtained. The methodology then proposes two types of matching on this intermediate data. First type of matching is Minimum-distance matching and the next is Linearity filtering to better the results obtained by former type. This methodology does not work with music pairs with same underlying melody but different transposition which is the heart of the problem around which this paper revolves.

From the above reviews we found that most of the methodologies proposed so far work only on MIDI file formats but not on the standard audio file formats in which millions and millions of songs are spread all over the World Wide Web and we propose this paper to address this problem

4. Methodology:

In this section we present the heart of our search engine, the algorithms and the working of the “Astute Song Hunter”. The working of the search engine is divided into two significant phases:

1. Pre-Processing of Songs – Tag Construction
2. Searching – Dual Filtering Approach

4.1 Pre-Processing of Songs – Tag Construction

As mentioned in the objective of this paper, an “Exhaustive and Efficient” search method of songs is implemented. Let us further look into what we exactly mean, when we say, “Exhaustive” and “Efficient”. The challenge here is that in order for the search to be exhaustive each song has to be searched in its entirety for the hummed tune. Searching each song in its entirety on a multimedia database with thousands of songs is impractical and is in

no way close to efficient. Thus, an innovative solution is devised in order to overcome this problem – the concept of "Tags".

Tags are the snapshots of a song at various points. Each song has various characteristics that fluctuate throughout the length of the song. All this information is captured in the tags using the algorithm presented here.

Nomenclature:

Є = *db store of tags*
Γ = *tag list of a single song*
S= *sampling rate*
β = *bytes per sample*
γ = *number of channels*

for every Song in the media database:

1. *reconfigure song with* **S**=*22050 and* **β**=*1 and* **γ**=*1*
2. *Frames = divide the song into 20 seconds frame with 75 % overlap*
3. *for each F in Frames:*
   - 3.1 **λ** = *extract dbpowerspectrum for F*
   - 3.2 **Θ** = *slope of regression line of spectrum*
   - 3.3 **M** = *list of maximum amplitudes of F*
   - 3.4 **μ** = *list of minimum amplitudes of F*
   - 3.5 *append (***Θ**,**M**,**μ***) to* **Γ**
4. *append (Song,***Γ***) to the* **Є**

Each song in the multimedia database is re-sampled at 22050 Hz, 1 byte per sample and converted from Stereo to Mono sound. The song is then divided into frames of length 20 seconds each with 75% overlap that is the last 15 seconds of the first frame overlap with the first 15 seconds of the second frame. This way any pattern of the hummed tune can be searched within the song. For each of these frames, the dB power spectrum is extracted using log FFT with a default spacing of 16384 and 'Hanning Window' type and window length of 1024. The slope of the regression line of the spectrum is then calculated. This is clearly explained in the following figure.

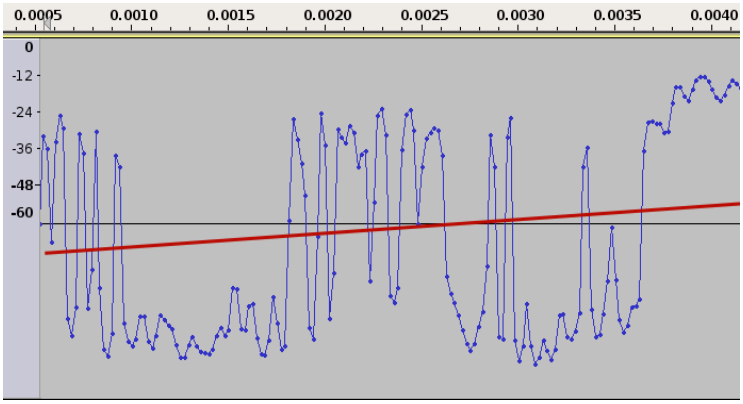

Fig 1: Regression Line Approach

Secondly each frame, extracted as above, is further divided into 256 sub-parts and the values of maximum & minimum amplitudes of the each sub-part are collected to get a list of maxima and minima of the amplitudes for the entire frame. The extraction can be shown

diagrammatically as follows -

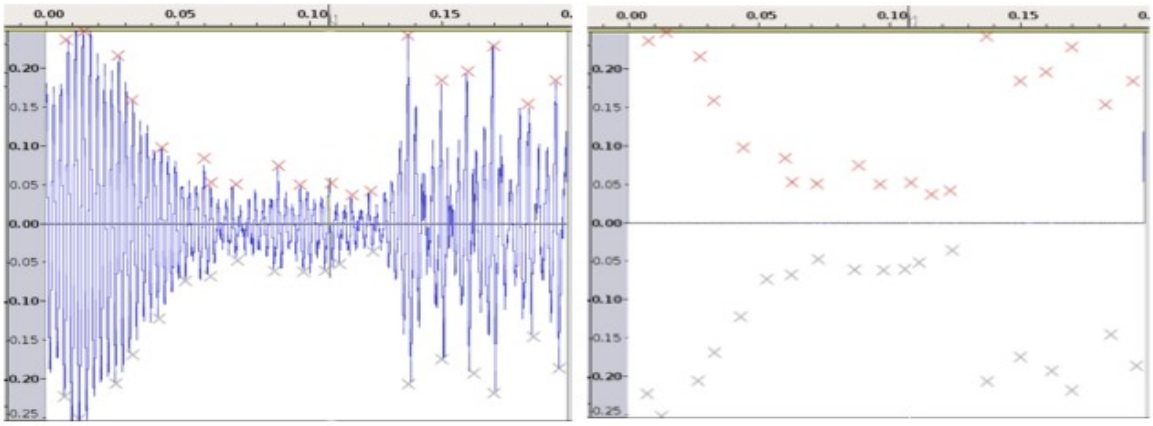

Fig 2: MaxMin Variations Approach

The slope, the maxima and the minima of a frame constitute the Tag. Such tags are generated for all the frames and a list of such tags is constructed. This is called the Tag-List. This tag-list is then mapped on to the song. Such tag-lists are constructed for all songs in the database.

4.2 Searching – Dual Filtering Approach

The most important aspect of any search engine is the search algorithm. In this section the all important search algorithm is presented. Before going to the actual algorithm, the following aspects need to be touched upon namely, the search criteria and the Dual Filtering Approach.

Firstly, the search criteria that are used in this search algorithm are the dB Power Spectrum and the amplitudes of each of the frames of the song. The main characteristics of a song are its frequency, pitch, intensity or power, and the amplitude. The main challenge in the search algorithm is to choose the right search criteria. Frequency and pitch could have been chosen as the search criteria. For monophonic song comparisons pitch and hence the notes can be used for exact song matching. In case of polyphonic songs, the presence of background music in the song interferes, thus deeply affecting the frequency and notes extraction. Hence matching 'notes' becomes extremely tedious and inefficient. Moreover, when we use the regression line approach (explained in the following section) it can be observed that the background music increases the Y-intercept, while the slope, which is the variation of power, is not altered. Thus we have chosen the power spectrum and the amplitudes as the search criteria.

Secondly, we call this search algorithm the Dual Filtering Approach since the selection of songs is done using 2 distinct methods and matched for final result. The slope of the db power spectrum and maxima & minima lists are extracted for the recorded hum from the user in the same manner as described above. Then the tag_list is fetched from the multimedia database. The slope of the regression line of all the frames for all the songs are scanned and the songs that make the least angle with the recorded hum are filtered. This is the Regression Line approach. Separately, the sum of squares of amplitude deviation of the song frames with the hum is calculated once each for the maxima and minima lists. The songs that generate same deviation both with maxima and minima are filtered. This is called the MaxMin Variations approach. Then the songs that fall in the intersection of the songs selected from the Regression Line approach and the MaxMin Variations approach are selected and

retrieved. Hence the name Dual Filtering Approach. The Search algorithm is presented below.

Nomenclature

S = *sampling rate*

β = *bytes per sample*

Ω = *match set using regression line approach*

σ = *match set using max_min variations approach*

Π = *final match set*

ϵ = *{(song1,[Θ_1,(M_1, μ_1)]), (song2,[Θ_2,(M_2, μ_2)]), ...} where*

Θ = *list of slopes of regression lines of each frame in song*

M = *list of lists of maximum amplitude values of each frame in song*

μ = *list of lists of minimum amplitude values of each frame in song*

Upon recording the hum-H from the user :

1. *reconfigure H with S=22050 and β=1*
2. λ = *extract dbpowerspectrum for H*
3. Φ = *slope of regression line of* λ
4. *for each Θ_1 in ϵ :*
   - 4.1 *for each Θ in Θ_1 :*
     - 4.1.1 δ = *angle b/w line with slope Θ_1 and Φ*
     - 4.1.2 *if δ < min_threshold:*
       - *add song to Ω*
5. ω_h = *list of maximum amplitudes of H*
6. ψ_h = *list of minimum amplitudes of H*
7. *for each (ω_s, ψ_s) in ϵ:*
   - 7.1 *for each (max_s , min_s) in (ω_s,ψ_s):*
     - 7.1.1 κ = *sum of square of deviation of ω_h from max_s*
     - 7.1.2 X = *sum of square of deviation of ψ_h from min_s*
     - 7.1.3 *if abs(κ-X) < min_threshold:*
       - *add song to σ*
8. Π = Ω *intersection* σ
9. *return Π*

The hum recorded from the user is re-sampled at 22050Hz with 1 byte per sample and converted from Stereo to Mono. The dB power spectrum is extracted and the slope of the regression line is calculated. For each song in the tag-list the slopes of the regression lines are extracted and the angle between the regression lines from the song and those from the recorded hum are compared and all the songs that fall under the min_threshold limit are selected and stored as set of songs from regression lines. For each of the tags in the tag-list the max and min values are compared with those from the recorded hum. If the difference in the sum of the squares of the deviation fall under the min_threshold limit then the song is selected and put in a set of songs from MaxMinVariation method. The intersection of these two sets is calculated and the songs that come under this intersection are selected for retrieval.

Finally it is necessary to explain the need of using 2 unique approaches and finding the intersection of two matched sets. Primarily, considering MaxMin approach, since the method just considers maxima and minima of all the 256 divided sub-parts it does not specify the order in which the maxima and minima appear. Meaning that in some songs a maxima

may occur after a minima or vice versa. Hence there is a possibility that several songs may be selected. In the Regression Line method, since regression is purely an approximation process, all songs that have the similar variation in the dB power spectrum are selected. In order to overcome these short comings we devised to use both approaches so that only those songs that have their amplitude variations as well as the power variations similar to that of recorded hum are selected. Hence the use of the Dual Filtering Approach is justified.

5. Findings and Analysis

The above search engine has been implemented using Python programming language. The following Python modules have been used to implement the sound processing aspects:

- Audioop

The Audioop module contains some useful operations on sound fragments. It operates on sound fragments consisting of signed integer samples 8, 16 or 32 bits wide, stored in Python strings.

- TkSnack

The Snack Sound Toolkit is designed to be used with a scripting language such as Tcl/Tk or Python.

The simulation was done with real world parameters in consideration. The following are the parameters considered for simulation:

- ✔ No. of songs in the multimedia database
- ✔ No. of songs selected.
- ✔ Time taken to perform the search.

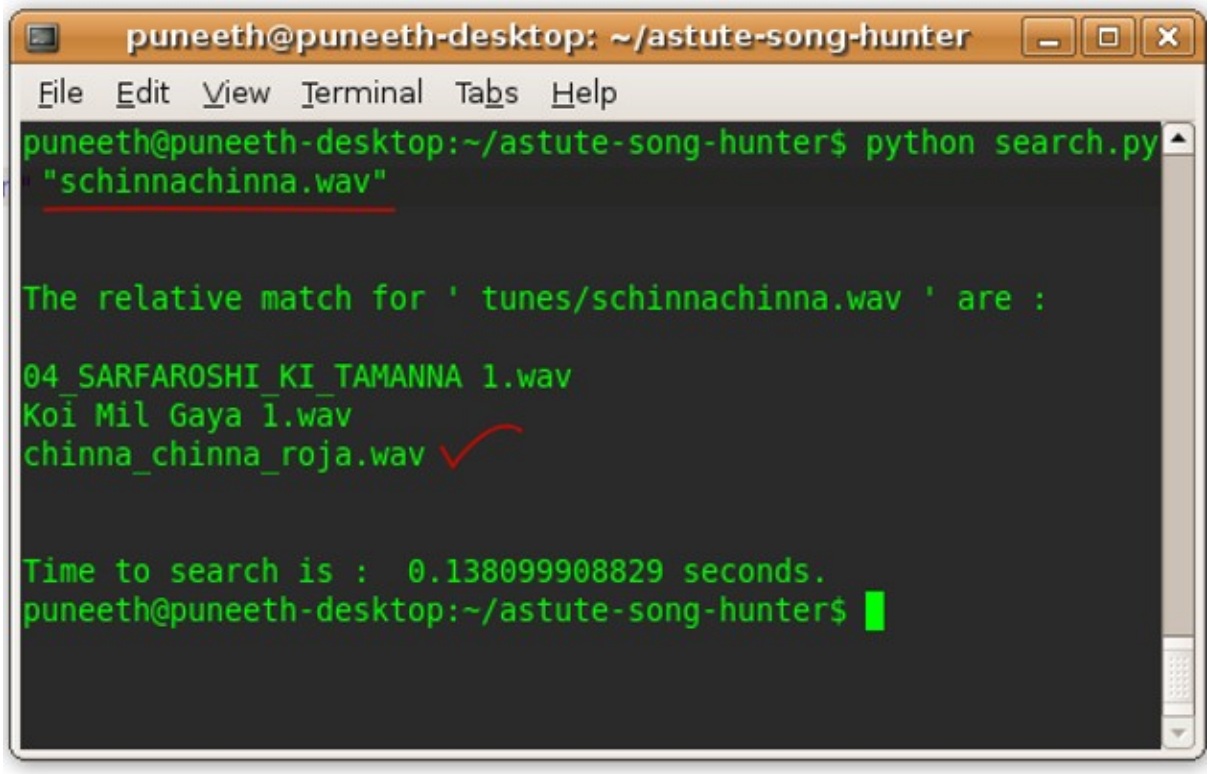

Fig 3: Screenshot of the simulation

The simulation was done with 50 songs in the database and Fig 3 is the screenshot taken during the simulation. The file “schinnachinna.wav” is the recorded hum and the file “chinna_chinna_roja.wav” is the tagged song in the database. The screenshot shows the list

of songs that belong to the matched set and also the time taken to search. From this screenshot we can observe that it took about 0.138s to search a database of 50 songs. The set of matched songs provided is also shown to be minimal.

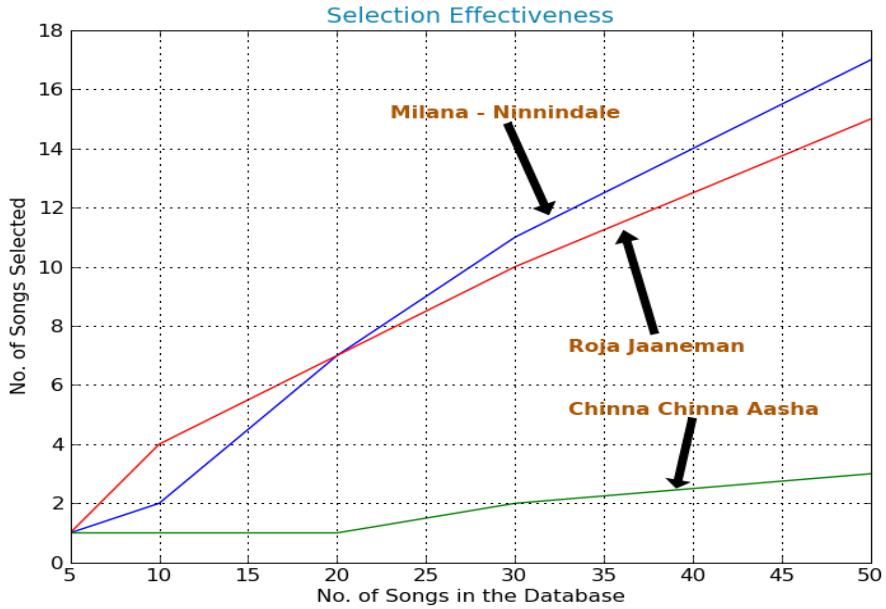

Fig 4: Selection Effectiveness

The graph shown in Fig 4. shows the selection effectiveness of the search engine. Here we have searched for three songs and plotted the graph of the number of songs that are selected into the matching set from the database to the number of songs present in the database. In each case, the song queried was selected by the algorithm as a part of the selected set. For each song we take the angular difference of the slopes and this changes for each song. We have set a lower limit on this angular difference as mentioned in the algorithm. Thus, the number of songs selected each time is dependent on the characteristics of the song.

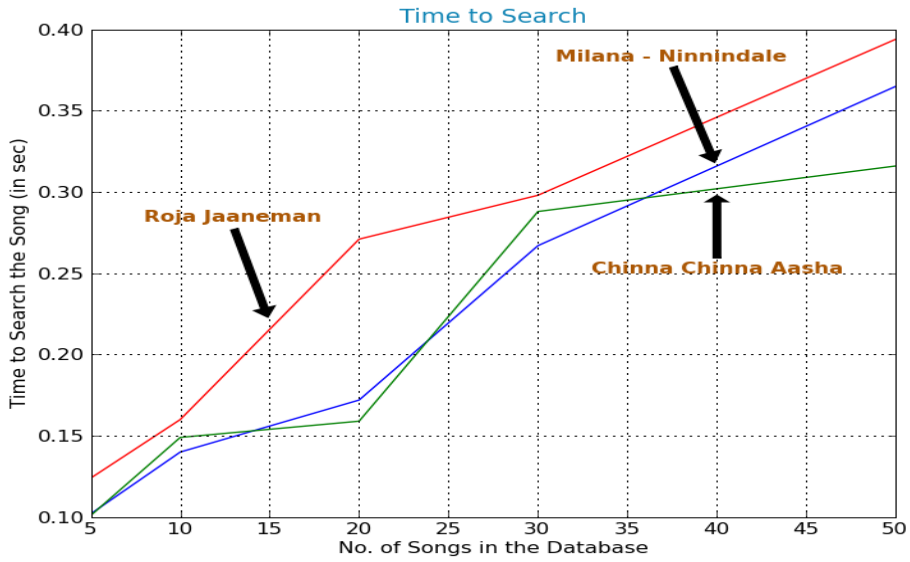

Fig 5: Time to search

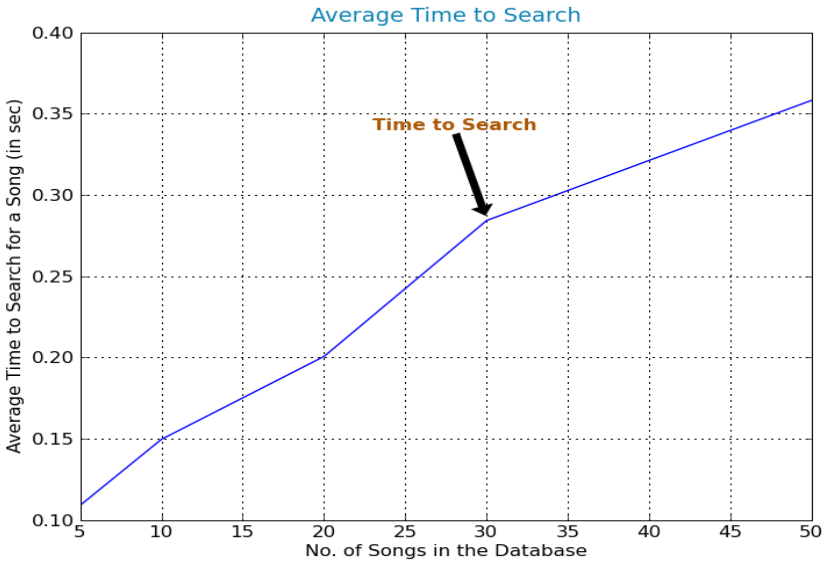

Fig 6: Average time to Search

In Fig 5, the graph shows the time taken to search each of the three songs selected previously, versus the number of songs that are present in the database. Considering the worst case scenario, the song "Roja Janeman" which seems to have recorded the maximum time to search has been searched within 0.40s on a database of 50 songs. Considering the best case scenario, the song "Chinna Chinna" seems to have recorded the minimum search time of just over 0.30s for a database of 50 songs.

Fig 6, shows the graph of average time taken to search a song versus the number of songs in the multimedia database. This average has been taken over varying number of songs depending upon the size of the database. The graph shows that the average time taken to search a song comes to just over 0.35s for a database of 50 songs.

From the above shown graphs we can clearly infer that the following:

- ➔ The time taken to search for a song in the worst case comes out to be 0.40s and in the best case comes out to be 0.30s for a database of 50 songs.
- ➔ The average time taken to search for a song in a database of 50 songs is around 0.35s

6. Commercial Viability

The above presented search engine is already close to become a full fledged product. The algorithm has been implemented in Python programming language with the use of additional modules all of which are Free and Open. The front end which is under development is built using a Python web framework, Django which is also Free and Open. None of the above said tools cost any money and thus make the construction of this product highly commercially viable. The hardware requirements include CPU time, memory and Secondary storage space. It has been shown from above results that the algorithm uses CPU time and memory optimally. It is acceptable that the algorithm places a high Secondary storage requirement because the tags along with Songs themselves are stored. But secondary storage has become so cheap these days that this requirement is hardly of any concern.

7. Conclusions

In this paper, we have presented a search engine that can be used to search for songs on a multimedia database by humming the tunes. Through simulation we have proved the effectiveness of the algorithm in searching for and recognizing the right set of songs. We have shown the behaviour of the algorithm for varying database sizes and varying range of input tunes or hums. The Dual Filtering Approach and Tag generation concepts are hereby proven to be effective.

We intend to refine the algorithm further and develop this into a full fledged commercial application that can be easily installed on to multimedia servers to make the "query by humming" functionality available on a large scale.

8. Application:

The search engine presented here is an innovation and has the potential to be a popular commercial application. The functionality provided by this search engine has not been implemented elsewhere and hence will be the first of its kind in the multimedia applications market. The following are the aspects that make this search engine unique:

- ✔ Effective search algorithms: The Dual Filtering Approach
- ✔ Exhaustive search: The concept of Tags

9. Limitation:

The following are the limitations of this algorithm:

- Length of the hummed input should be a minimum of 15s and a maximum of 30s for proper results
- Not noise tolerant, that is, too much noise in the recorded hum causes unexpected results.
- Searching of songs not present in the database may also yield results, this is because the algorithm always retrieves the song similar to the hummed tune.

10. References and Bibliography:

[McNab-1] Rodger J. McNab , Lloyd A. Smith, Ian H. Witten, Clare L. Henderson and Sally Jo Cunningham, "Towards the Digital Music Library: Tune Retrieval from Acoustic Input", Department of Computer Science, School of Education, University of Waikato, Hamilton, New Zealand

[Chai-2] Wei Chai, "Melody Retrieval On The Web", Program in Media Arts and Sciences, School of Architecture and Planning, MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[Yang-3] Cheng Yang, "Music Database Retrieval Based on Spectral Similarity", Department of Computer Science, Stanford University