CubyHum: A Fully Operational Query by Humming System

Steffen Pauws
Philips Research Eindhoven
Prof. Holstlaan 4 (WY21)
Eindhoven, the Netherlands
+31 40 27 45415
steffen.pauws@philips.com

ABSTRACT

'Query by humming' is an interaction concept in which the identity of a song has to be revealed fast and orderly from a given sung input using a large database of known melodies. In short, it tries to detect the pitches in a sung melody and compares these pitches with symbolic representations of the known melodies. Melodies that are similar to the sung pitches are retrieved. Approximate pattern matching in the melody comparison process compensates for the errors in the sung melody by using classical dynamic programming. A filtering method is used to save computation in the dynamic programming framework. This paper presents the algorithms for pitch detection, note onset detection, quantization, melody encoding and approximate pattern matching as they have been implemented in the CubyHum software system. Since human reproduction of melodies is imperfect, findings from an experimental singing study were a crucial input to the development of the algorithms. Future research should pay special attention to the reliable detection of note onsets in any preferred singing style. In addition, research on index methods and fast bit-parallelism algorithms for approximate pattern matching need to be further pursued to decrease computational requirements when dealing with large melody databases.

1. INTRODUCTION

Typically, people listen to music separately from gaining knowledge about the name of the performer, the composer or the title of the song. Because song titles and melodies are not learnt associatively, recalling a song title from a given melody or vice versa is notoriously difficult [14]. Obviously, it is hard to find music without knowing it by heart. The interaction concept of 'query by humming' makes it possible to retrieve a song when the user ponders a catchy tune without being able to name the song. It allows the user to sing any melodic passage of a song, while the system seeks the song containing that melody fast and orderly [6][11].

The current implementation of 'query by humming' in the CubyHum software system is a linked combination of speech signal processing, music processing and approximate pattern matching guided by empirical findings from singing experiments.

Its algorithmic organization is illustrated in Figure 1 and forms the guide to this paper. 'Query by humming' requires symbolic representations of the song melodies consisting of a sequence of musical notes (e.g., their pitch names) and the time onset and offset of each note.

First, the pitch is estimated from the singing by a technique called sub-harmonic summation (SHS). In short time frames, SHS computes the sum of harmonically compressed spectra. In principle, the maximum sum result is chosen as the pitch estimate in that time frame.

Second, musical events and timing information are detected in the singing such as note onsets, gliding tones and inter-onset-intervals. Standard signal processing techniques using short-time energy, pitch level shifts and amplitude envelopes are used for finding note onsets.

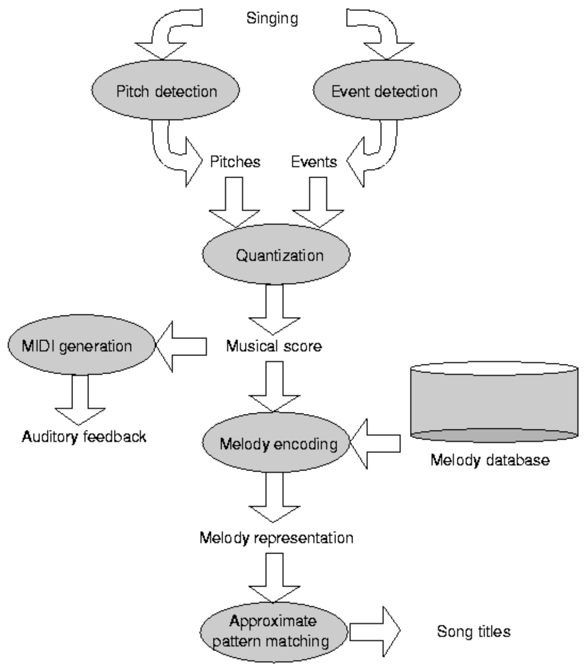

Figure 1. The algorithmic framework of 'query by humming' as implemented in CubyHum.

Both the pitch and timing information are combined and quantized into musical notes and durations. Note quantization is based on the tuning and scale standards in Western music. This quantization transforms the singing input into its formal musical notation. From this musical score, a melody can be synthesized for auditory feedback by using standard MIDI[1] technologies.

Since people are inaccurate in remembering and reproducing a melody, a melody representation and comparison process are devised that are largely invariant to key, tempo and ornamentation (i.e., adding and leaving out notes). First, not the absolute musical pitches, but the intervals between notes are used, as they are insensitive to key. In particular, nine interval classes are defined that represent different interval sizes. Large intervals (i.e., larger that 6 semitones) are not further distinguished since they are hard to sing and occur rarely in musical melodies from all over the

[1] MIDI stands for Musical Instrument Digital Interface, which is a standard format to exchange music performance data between music synthesizers and sequencers.

world. Likewise, duration ratios are used between successive notes, as they are insensitive to tempo.

Approximate pattern matching techniques allow the detection of patterns in tunes that are transformed and distorted in various ways. For that, we have defined a distance between any two variable-length melodies that depends on interval sizes and duration ratios. In general, it comes down to the search of an optimal alignment between melodies by minimizing the number of changes that are needed to transform one melody into another. This problem can be easily solved in a dynamic programming framework. However, classical dynamic programming is impractical in terms of running time performance for large melody databases. Therefore, a filter mechanism has been implemented that quickly discards passages in the melody database that cannot contain an approximate match.

2. THE ART OF SINGING

It is obvious that people have imperfect memories for melodies or may lack any formal singing practice. Unfortunately, the literature does not provide clear-cut insights into the singing performance of the general public, the long-term memory and recollection of melodies and how these issues relate to experienced singers and real-world song material. This knowledge is indispensable for the development of 'query by humming' algorithms. Hereunder, we summarize some of our findings of an experiment that examined the effects of singing experience, song familiarity and recent song exposure on singing popular rock song melodies[2] from memory. These findings were a crucial input to the development of the algorithms for 'query by humming'.

People sing any part of the melody. A repetitive melodic passage in a song *may* represent the 'hook-line' of a song that 'gets stuck in people's head'.

People sing at the wrong key. In our study, people chose a random pitch to start their singing. Only for their most favorite songs, people are thought to have a latent ability of absolute pitch [8].

People sing at a reasonably correct global tempo. In our study, people knew or had a feeling, by previous hearings, what the correct tempo would be and were able to approach this tempo reasonably accurately.

People sing too many or too few notes. Human memory is imperfect to recall all pitches in the right order. In our study, people sang just the line they remembered. They also added all kinds of ornaments (e.g., grace notes, filler notes, or thinner notes) to beautify their singing or to ease the muscular motor processes involved in singing.

People sing the wrong intervals or confuse some with others. From our study, people sang about 59% of the intervals correctly, though there were differences due to singing experience, song familiarity and recent song exposure. Interval confusion seems to be symmetric; interchanging an interval with another was found to be equally likely as the other way around. A large interval (thirds and larger) tends to be more easily interchanged for another.

People sing the contour reasonably accurately. In our study, people largely knew when to go up and when to go down in pitch when singing; they did that correctly in 80% of the times.

People with singing experience sing better on some aspects than people without singing experience do. In our study, the non-experienced and experienced singers did not differ in singing the contour of a melody accurately. However, experienced singers reproduced proportionally more correct intervals and sang at a better timing.

People sing familiar melodies better than less familiar ones In our study, less familiar melodies were reproduced with fewer notes and had proportionally fewer correct intervals than familiar melodies. Also, both experienced and non-experienced singers improved their singing of intervals when they had heard the melody very recently.

3. PITCH DETECTION

Pitch is a percept that is defined as the characteristic of a sound that gives the sensation of being high or being low. For a complex tone (as the human voice), the pitch corresponds mainly with the fundamental frequency of the signal. However, the correlation between sound frequency and pitch is not perfect, since pitch perception is influenced by the intensity, the duration, the surrounding sounds and the harmonics of the sound. Accurate pitch detection algorithms for normal speech have been developed, under the assumption that the voice has been well recorded under controlled conditions.

Typical pitch in normal conversational speech is in the range of 110 Hz in the male, 220 Hz in the female and 300 Hz in the child and may vary within one octave. When considering all voices and registers, singing extends from 80 Hz (the E_2 of a bass singer) to 1400 Hz (the C_6 of a soprano singer) and may vary over two-and-a-half octaves. We expect the pitch range for humming to lie in the first one-and-a-half octave of the normal singing range. On the other hand, whistling extends from 700 Hz to 2800 Hz; the lowest whistling tone of person comes near to the person's highest reachable sung note.

We use the *sub-harmonic summation* (SHS) method [7] to estimate the pitch in the singing. This method stems from the theory that each spectral component (i.e., spectral peak) contributes to the perception of a pitch that corresponds to the frequency of the component. Also, elements that have a lower harmonic relationship with this component (i.e., that have an integral factor in frequency) contribute to the perception of a pitch, taking into account that higher components contribute less than lower components do. All these contributions add up in a *sub*-harmonic summation. The maximum of this sum result is the estimate of the pitch. Even when a fundamental frequency (i.e., the first harmonic) is missing in the signal, while other harmonics are present, this mechanism creates a (virtual) pitch of about that typically produced by that fundamental. Virtual pitch is common in human perception. The algorithm has been implemented as outlined in the original paper [7].

The algorithm can be explained by a single expression,

$$H(s) = \sum_{n=1}^{N} h^{n-1} W(s + \log_2 n) \left| S(s + \log_2 n) \right| \qquad (1)$$

where $s={}^2\log f$ denotes the logarithmic frequency, $H(s)$ represents the sub-harmonic sum spectrum, N (e.g., 15) denotes the number of harmonics, n is the compression rank, h (e.g., 0.84) denotes the decreasing factor, $W(s)$ is an arc-tangent function representing the transfer function of the auditory sensitivity filter, and $\left| S(s + \log_2 n) \right|$ denotes the compressed amplitude spectrum representation. When a spectrum is compressed, its frequency axis is squeezed by an integral factor, the compression rank *n*, and consequently its peaks come closer to each other. A pitch estimate is then the value of $f = 2^s$ for which $H(s)$ is the maximum.

[2] Twelve songs of the Beatles were used as experimental data.

Robust pitch estimation is difficult when relying on a single frame; octave errors or other erratic pitch estimates for 'creaky' and 'hoarse' voices are hard to circumvent without taking the necessary precautions. Since the sub-harmonic sum spectrum provides an array of pitch estimate candidates, a post-processing procedure is used to smooth the pitch contour. For that, we use a dynamic programming framework in which sudden pitch jumps larger than 50 Hz are prohibited by interchanging a deviant absolute maximum by one of the relative maximums in the sub-harmonic sum spectrum.

4. EVENT DETECTION

Once the continuous pitches have been identified, the time locations at which a note starts (the onset time) and ends (the offset time) have to be found. To date, no algorithm has been developed that reliably detects the wide range of possible note onsets in performance data from different singing styles. For the current purpose, note onsets are defined as vowel onsets. Consequently, the singing is assumed to consist of short, relatively isolated syllables, preferably comprising a lengthened unvoiced fricative and a long mid vowel (e.g., /fa/-/fa/-/fa/). Note onsets are then characterized by an abrupt rise in energy over a broad frequency range. The sustained note has a relatively steady spectral shape representing the formants of the vowel used. Though note offsets can be identified to some extent by the fall of energy in especially the higher frequencies, they are less clearly defined. This is due to the exponential decay of the amplitude of a note making a note already inaudible while it is still physically present.

We have linked several standard signal processing techniques to detect the rise and fall of energy in different frequencies to segment the signal into note onset and offset times. First, the short-term energy method is used to detect silent parts in the singing. Subsequently, each non-silent part is provided to the other three methods in succession. We assume a digital signal $x(n)$, for $n = 0,\ldots,N-1$ that is pre-emphasized by the filter $1-0.95z^{-1}$ to produce a 32 dB boost in the spectral magnitude, blocked into frames.

4.1 Short-term energy method

The *short-term energy* method is a straightforward method to detect note onsets and offsets and to distinguish singing from silence. For that, the signal is blocked into non-overlapping frames of 10 ms.

The short-term energy E_k in frame *k* is estimated by

$$E_k = \sum_{n=1}^{N} x(n)^2 \tag{2}$$

The short-time energy is normalized by a maximum short-term energy found in a running window of the signal. Adaptive threshold values are used to determine note onsets and note offsets, in which the note onset threshold (e.g., 0.02) is defined to be higher than the note offset threshold (e.g., 0.01). This is done to avoid an on-off oscillation of a marginal signal (e.g., a weak fricative). First, the procedure looks for the first note onset to be detected. If the short-time energy exceeds the onset threshold, a note onset is detected. If, after a note onset has been detected, the short-time energy falls below the offset threshold, a note offset is detected.

4.2 Surf method

The *Surf* onset detection algorithm has been adopted from the techniques of Schloss [17]. The signal is passed through a first-

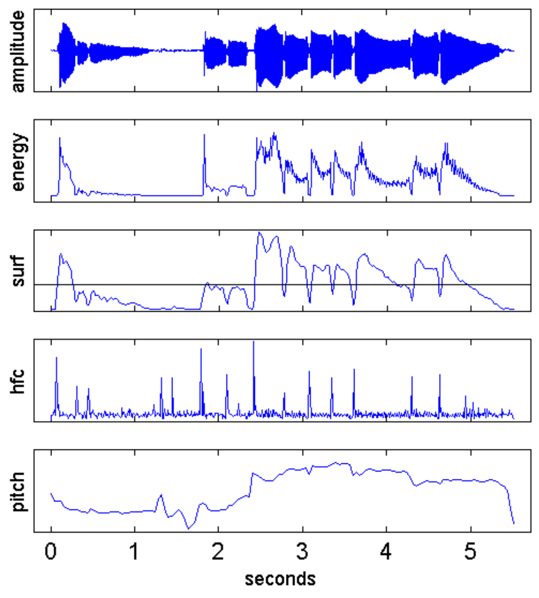

Figure 2. The waveform produced by a male person singing the first melody line of the Beatles' song 'Yesterday' of 12 notes by using the syllables 'na-na'. The short-term energy method indicates regions of silence and singing for isolated notes; it detects 9 note onsets. The Surf method indicates the same 9 note onsets by looking at the positive zero-crossings in the surf contour. The high-frequency content method can detect all 12 note onsets by peak picking; there are some spurious peaks that can result in 'false alarms'. The pitch method can detect gliding note onsets by looking how the pitch contour fluctuates over time.

order high pass filter and blocked into frames of 20 ms, with a frame shift of 10 ms. The frames are used to compute a smoothed amplitude envelope of the signal that represents the higher frequencies. This envelope $y(k)$ is made out of the sequence of average absolute values of the signal within each frame *k*. The slope of the envelope is found by a polynomial fitting procedure [15]. For that, the envelope sequence is fitted with a second-order polynomial over a *M*-point segment of the sequence. We denote a small finite envelope sequence around sample τ by $\{y(\tau+t)\}_{t=-M}^{M}$, where M specifies the width of the polynomial interpolation. This small segment is fitted by a second-order polynomial $a+bt+ct^2$ by minimizing some fitting error.

A polynomial approximation of the slope at τ is then given by

$$\frac{\partial y(\tau+t)}{\partial t} \cong b = \sum_{t=-M}^{M} \frac{t \cdot y(\tau+t)}{\sum_{t=-M}^{M} t^2} \tag{3}$$

We compute a 5-point approximation; M is fixed at 2. At the edges of the envelope sequence, no slope computation is done.

Since a note onset is characterized by an abrupt rise of higher frequencies of the signal, we looked at positive zero-crossings of the slope contour to find these onset times.

4.3 High-frequency content method

The *high-frequency content* method has been adopted from Masri and Bateman [9]. It aims at revealing both changes in overall energy and the energy concentration at higher frequencies. The signal is blocked and Hanning-windowed into frames $k = 0,\ldots,K-1$ of 20 ms, with a frame shift of 10 ms.

A *M*-point FFT is used to produce a short-time DFT $X_k(m)$. The short-time energy E_k in frame *k* is computed as the sum of the squared magnitude of each FFT bin,

$$E_k = \sum_{m=m_0}^{M/2+1} \left|X_k(m)\right|^2 \tag{4}$$

where m_0 denotes the lowest FFT bin that is taken into account. Only the FFT bins are considered that fall within a frequency band of 400 Hz and higher.

The short-time higher frequency content H_k in frame *k* is a weighted version of E_k, linearly biased to the higher frequencies,

$$H_k = \sum_{m=m_0}^{M/2+1} m \cdot \left|X_k(m)\right|^2 \tag{5}$$

A detection function is computed that combines each pair of consecutive frames; it is the product of the rise in high frequency energy between two frames and the current normalized high frequency content,

$$D_k = \frac{H_k}{H_{k-1}} \cdot \frac{H_k}{E_k} \tag{6}$$

The first ratio represents the rise in high frequency content. The second ratio represents the normalized high frequency content for the current frame *k*. Their product (i.e., D_k) peaks prominently at abrupt increase in high frequency energy content. If it surpasses a given threshold, we say that the detection function has found a note onset.

4.4 Pitch method

The *pitch* method is a way to segment gliding notes into note onset and offset times. Essentially, it groups and averages the pitches as found in 40 ms frames in different windows on a frame-by-frame basis. For this, we use a growing window of frames for which it has been concluded that they contain similar pitches. The growing window maintains the pitch of a current note. A second window of fixed size is placed at the end of the growing window.

The median pitch is computed for both windows. If the median pitch of the fixed-size window falls within 100 cents (a semitone) of the median pitch of the growing window, the growing window is extended with one frame and, consequently, the window of fixed size is moved one frame; the computation starts over again.

If, however, the median pitch of the fixed-size window equals or differs more than 100 cents (a semitone) from the median pitch of the growing window, it is concluded that the position of the fixed-size window notifies the start of a new note.

The offset time of the current note and the onset time of the new note are determined by the starting frame of the window of fixed size. Now, the window of fixed size is the growing window and a new window of fixed size is placed at the end of it; the computation starts over again.

The minimal length of any window is determined at 3 pitch frames (120 ms), which also determines the minimal duration of a gliding note. Finding note onset and offset times on the basis of a changing pitch has the advantage that the user has some freedom in singing: notes do not need to be sung in an isolated manner but can be 'thread together'. However, a gliding note (or glissando) is segmented into a step-wise sequence of ascending or descending notes of 120 ms each.

5. QUANTIZATION

Quantization means the division of the pitch (tone) and time continuum into discrete steps. These steps are necessary to decide exactly how much the pitch has changed or what temporal units have elapsed. Time quantization is not further discussed, since quantized duration is not used in the melody comparison. Instead, the inter-onset-interval (IOI) is used, which is the time difference between two adjacent note onsets.

To arrive at a discrete musical pitch, a pitch center of a continuous pitch contour within a time interval is required. This pitch center should be the pitch perceived by the listener and correspond to the target pitch the singer intended to produce. Although there is ample evidence that the discrete pitch perceived in a pitch contour is that of the mean [2], the use of a mean is sensitive to octave errors or other deviant pitch estimates (e.g., due to vibrato, pitch overshooting or undershooting). We use the median pitch instead. If the contour is error free, the median and mean pitch in a time interval are reasonably close.

The median pitch between a note onset and offset is used to quantize the musical pitch value for each note using the equally tempered musical scale tuned at $A_4 = 440Hz$ (A-440). Musical pitch is represented in categories along scales in terms of semitones and cents. These categories are relative measures based on frequency ratios. Knowing that an octave is a frequency ratio of 2:1, the semitone is one-twelfth of an octave ($\sqrt[12]{2} \approx 1.05946$) and a cent is 1/100 of a semitone. Now, notes on the equally tempered scale relative to A-440 occur at multiples of 100 cents; they can be expressed as a distance in cents from 8.176 Hz and have a total order. For instance, the middle C ($C_4 = 261.63Hz$) is 6000 cents (or there about). To calculate the discrete musical pitch *p* of a median pitch *f*, we use the frequency ratio $R = f/8.176$ and

$$p = \lfloor 12 \cdot \log_2 R + 0.5 \rfloor \tag{7}$$

The musical pitch is then represented by an integer value; its corresponding pitch label is indexed in an array. By allowing an integer range between 0 and 127, we have essentially the MIDI convention to encode musical pitch.

Singing is inevitably contained with deviations and changes from the universal tuning standard at A-440. For instance, people tend to fall or rise in pitch (and key, consequently) due to large interval sizes, fatigue and inaccuracies in muscular control. To compensate for this, the tuning standard is adapted to the singing in due course. For the first note sung, it is assumed that singing starts at the universal tuning standard. For each next note, the (closest) musical pitch is calculated and the cent difference between the pitch sung and the musical pitch is computed. Using an inverse variant of Equation 7, the reference pitch (which is at 8.176 Hz, at first) is adapted, which changes all musical pitches relative to this reference pitch.

6. MELODY REPRESENTATION

A melody representation that is invariant to key is an interval representation, that is, the sequence of distances between two succeeding notes expressed in semitones. In particular, a melody sequence $S = s_1 s_2 \ldots s_N$ comprising absolute pitches is transformed in a sequence $S' = \cdot(s_2 - s_1)(s_3 - s_2)\ldots(s_N - s_{N-1})$, where the dot '$\cdot$' represents a special start element since no interval is associated with the first note in a melody.

In order to transcend from mode and to use a diatonic scale structure, intervals are grouped together. This has to be done without knowledge about the tonic since we do not know the key of the melody. Except for the unison, all intervals are categorized in groups of two intervals of 1 to 2 semitones. In addition, all intervals larger than 6 semitones are grouped in a single category for ascending and descending intervals. It is well-known that intervals larger than 5 semitones and greater are rare (only 10%) in musical melodies from all over the world [5]. In addition, large intervals are difficult to sing accurately.

As shown in Table 1, the interval categories are represented by the integers $-4,-3,\ldots 3,4$. The special start element '$\cdot$' is maintained. The resulting melody representation is a 9-step contour and resembles the Dowling model of how melodies are assumed to be stored in human memory [4]. Temporal information is kept by storing the real value of the inter-onset-interval (IOI) of each note to each corresponding 9-step contour element.

Invariance to global tempo is established by calculating the ratios of two IOIs, but this is done in the melody comparison.

Table 1. The 9-step melody representation (st = semitone).

interval name	interval size	integer code
desc. dim. fifth and greater	< -6 st	-4
desc. perfect/augm. fourth	-5 or –6 st	-3
desc. minor/major third	-3 or –4 st	-2
desc. minor/major second	-1 or –2 st	-1
unison	0 st	0
asc. minor/major second	1 or 2 st	1
asc. minor/major third	3 or 4 st	2
asc. perfect/augm. fourth	5 or 6 st	3
asc. dim. fifth and greater	> 6 st	4

7. MELODY COMPARISON

The art of melody comparison is finding approximate similarities between finite sequences of elements drawn from a finite alphabet, though the sequences have different lengths. In our case, one sequence is a relatively small pattern and a second sequence that has a longer length. The former sequence represent a transcription of what has been sung; the latter sequence represent a melody from the database. This sequence can also be interpreted as a concatenation of all melodies in the database. The pivot is defining an appropriate similarity metric (or distance measure) for melodic sequences that (1) assigns different costs to different local dissimilarities, (2) meets some invariance principles and (3) is 'psychologically plausible'; it should provide an orderly representation of the melodies that fits human expectation and music theory.

7.1 Notation

We adopt the following notation for comparing melodic sequences: let $Q = q_1 q_2 \ldots q_M \in \Sigma^*$ be a query pattern sequence of length $|Q| = M$ and $S = s_1 s_2 \ldots s_N \in \Sigma^*$ a sequence of length $|S| = N$. Σ is a finite alphabet of pitch intervals. Here, we use the 9-step alphabet $\Sigma_{9-step} = \{-4,-3,\ldots 3,4\}$.

We denote s_j as the *j*-th element of *S* for an integer $j \in \{1,\ldots,N\}$. We denote $S_{i\ldots j} = s_i \ldots s_j$ as a subsequence (or factor) of *S*, which is the empty subsequence ε if $i > j$. The *prefixes* of *S* are the $N+1$ subsequences $S_{1\ldots j} = s_1 \ldots s_j$ for $0 \le j \le N$. Likewise, the suffixes of *S* are $S_{j\ldots N} = s_j \ldots s_N$ for $1 \le j \le N+1$. In addition, we define a tabular function L_S, which is specific to the sequence *S*, that provides the IOI for a given *j*-th element of *S*. In particular, $L_S(s_j) := t_j$, where t_j is the IOI for s_j.

7.2 Typical problem instances

The performance of an approximate pattern matching algorithm depends on the length of the query pattern sequence *M*, the size of the longer sequence *N*, the size of the melody database, the size of the alphabet $|\Sigma| = \sigma$, the number of differences allowed *k* and consequently the error level $\alpha = k/M$.

Practical problem instances for our melody comparison can be described by the following parameters.

- The query pattern sequence *Q* has a typical length *M* of a dozen elements. For instance, singing the first phrase of the Beatles' song 'Yesterday' amounts to singing 12 notes (or 11 intervals).
- The melodic sequence *S* has a typical length *N* of a few hundred elements. The vocal melody of a popular rock song has about 300 notes.
- The melody database can be as small as a few hundreds (for small-scale applications) and thousands and thousands (for full-scale applications).
- The alphabet Σ_{9-step} has 9 elements. These elements are integers representing interval categories having total order.
- In our singing experiment, we found that the percentage of errors allowed is in the range of 20-40%.

7.3 Edit distance

The traditional way to compare two sequences is to allow particular differences (or errors) of elements to occur in the sequences while computing their distance, denoted as $ed : \Sigma^* \times \Sigma^* \to \Re_0^+$. Thus, $ed(Q,S)$ represents the distance between Q and S. The type of differences can be deletions, insertions and replacements of single elements that are necessary to transform one sequence into the other. A cost (or penalty) is associated with each transformation (or difference). A cost may be a constant (e.g., a unit cost for each transformation) or any value function that computes the difference between two elements in its context. By choosing appropriate costs, one can select those approximate matches that make sense in a particular domain and reject other which do not. When we restrict the costs to be unit, the match will be based on the unit-cost edit distance, that is, the minimal number of deletions, insertions and replacements to transform the sequence *Q* into the sequence *S* [20].

The unit-cost edit distance model can be used in two different ways.

1. **Minimal distance problem.** Finding an approximate match between *Q* and *S* that has minimal edit distance.
2. ***k*-difference problem.** Finding an approximate match (or all approximate matches) in *S* that has (or have) at most $k \in \aleph_o^+$ different elements with *Q* (i.e., at most an edit distance of *k* with *Q*).

The computation of the edit distance can be easily solved by using classical dynamical programming for sequences of the same length [20] and for sequences of different lengths [17].

7.4 Local melody differences

The edit distance model works fine for textual sequences and for melody representations that abstract from tonal and timing structures such as a contour representation. For melodies decoded in Σ_{9-step} and with timing information, we have to account for other types of differences and their effects.

Hereunder, we enumerate the most important local differences between melodies. The ones that have to do with musical pitch are shown in Figure 3. Some of these differences (or human errors in melody reproduction) have already been discussed in Section 2.

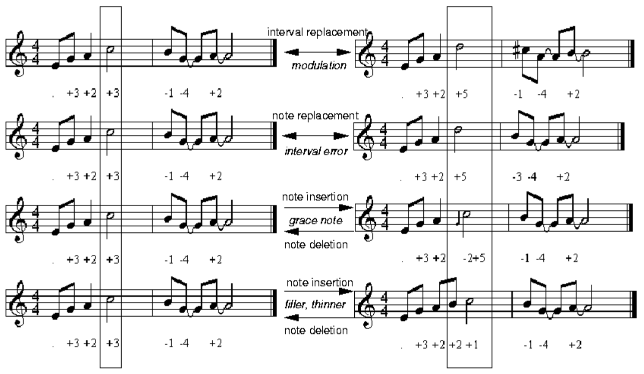

Figure 3. Typical local differences between melodies. Underneath each musical staff, the interval sizes in semitones are shown, not the interval categories from Σ_{9-step} .

1. *Melodic sequences of variable length*: the singing of any part of a melody.
2. *Amount of mistuning*: the singing of an interval a little too sharp or flat should not be as bad as singing it far too sharply or flat.
3. *Modulation by interval replacement*: the singing of a wrong interval may result in a key modulation of the whole succeeding melody line.
4. *Note replacement*: replacing one note for another note has implications for the interval representation of the melody. The interval associated with the replaced note changes by a certain number of semitones, which is compensated by the very next interval.
5. *Note insertion*: the singing of an additional note (a filler or grace note) has implications for the interval representation of the melody. The sum of the sizes of the two new intervals introduced by the inserted note equals the size of the original interval.
6. *Note deletion*: forgetting to sing or missing a particular note (a thinner note) has implications for the interval representation of the melody. The size of the new interval due to the deleted note equals the sum of the sizes of the two original intervals.
7. *Other note and interval insertions and deletions*: some insertions and deletions of small melodic fragments cannot be accounted by some of the above-mentioned schemes. For instance, short melody lines can be added or deleted.
8. *Duration error*: the lengthening or shortening of a note without changing global tempo.

In addition to these local melody differences, one might think about the concepts of *consolidation* and *fragmentation* as introduced by Mongeau and Sankoff [12]. A consolidation represents the replacement of several notes at the same pitch by a single note at that pitch. Likewise, a fragmentation represents the replacement of a single note by several ones at the same pitch. If these sequence differences were interpreted by a series of single insertions and deletions, it would cost more than counting them as a single transformation. However, these concepts occurred rarely in the singing data of the experiment and are computationally intensive. Therefore, we found them not in proportion to their added benefit.

7.5 Dynamic programming solution

Similar to the edit distance model, the classical dynamical programming approach to compute the melody distance $D(Q,S)$ between two melodic sequences $Q = q_1 q_2 \ldots q_M$ and $S = s_1 s_2 \ldots s_N$ is done by filling a matrix $(D_{0\ldots M,0\ldots N})$. The entry $D_{i,j}$ holds the minimal melody distance between the two prefixes $Q_{1\ldots i}$ and $S_{1\ldots j}$. The algorithm to construct the matrix is done by using the following recurrent formula

$$D_{i,j} = \min \begin{cases} D_{i-1,j} + 1 + K \cdot \left| \dfrac{L_Q(q_i)}{L_Q(q_{i-1})} \right| & \text{(interval deletion)} \\ D_{i-2,j-1} + 1 + K \cdot \left| \dfrac{L_Q(q_{i-1}) + L_Q(q_i)}{L_Q(q_{i-2})} - \dfrac{L_S(s_j)}{L_S(s_{j-1})} \right|, & \\ \quad \text{if } q_{i-1} + q_i = s_j,\ i > 2 \quad \text{(note deletion)} & \\ D_{i-1,j-1} + \dfrac{C}{\sigma}\left|q_i - s_j\right| + K \cdot \left| \dfrac{L_Q(q_i)}{L_Q(q_{i-1})} - \dfrac{L_S(s_j)}{L_S(s_{j-1})} \right| & \\ \quad \text{(modulation or no error)} & \\ D_{i-1,j-2} + 1 + K \cdot \left| \dfrac{L_Q(q_i)}{L_Q(q_{i-1})} - \dfrac{L_S(s_{j-1}) + L_S(s_j)}{L_S(s_{j-2})} \right|, & \\ \quad \text{if } q_i = s_{j-1} + s_j,\ j > 2 \quad \text{(note insertion)} & \\ D_{i,j-1} + 1 + K \cdot \left| \dfrac{L_S(s_j)}{L_S(s_{j-1})} \right| & \text{(interval insertion)} \end{cases} \quad (8)$$

where $\sigma = \left|\Sigma_{9-step}\right| = 9$ denotes the size of the alphabet, and C and K denote constants that have to be determined empirically. We use $C = 1$ and $K = 0.2$.

The following set of initial boundary conditions and special cases is used

$$\begin{aligned} D_{0,0} &= D_{1,0} = 0 \\ D_{1,0} &= 1 \\ D_{i,0} &= D_{i-1,0} + 1 + K \cdot \left| \frac{L_Q(q_i)}{L_Q(q_{i-1})} \right| \\ D_{i,1} &= D_{i-1,0} \\ D_{0,j} &= 0 \\ D_{1,j} &= D_{0,j-1} = 0 \end{aligned} \quad (9)$$

The rationale of the recurrent formulae is, first, that pitch intervals between melodies are penalized by their absolute difference, $\left|q_i - s_j\right|$. If the pitch intervals are equal, there is no interval cost. If they are not equal, we speak about an interval replacement that may result in a modulation (key-change) of one melody in comparison to the other. The interval cost is normalized by the size of the alphabet so that it will never reach a cost of 1 or higher. Additional to this interval cost, there is a durational cost expressed by the absolute difference of duration ratios. The constant *K* represents the relative contribution of duration differences versus that of interval differences. A note replacement is not explicitly accounted for, but it can be interpreted as two modulations in series since it involves two succeeding intervals.

Second, if $q_{i-1} + q_i = s_j$ or $q_i = s_{j-1} + s_j$, we speak of a note insertion or note deletion, respectively. Recall that a note insertions or deletions have special implications for the underlying intervals, expressed by the conditional summations. An interval cost of 1 is associated with these differences. The durational cost penalizes longer durations of inserted or deleted notes more than smaller durations; it thus favors grace notes for thinner and filler notes. In principle, the concepts fragmentation and consolidation can be worked out using the same scheme.

The two remaining differences are the insertions and deletions that cannot be accounted for by the other schemes. Their costs are 1 plus a varying durational cost. The duration cost is based on the motivation that the deletion of an interval can be seen as replacing a note with a nullified note of zero-length. Likewise, an interval insertion is similar to replacing a zero-length note with a note of a non-zero length.

The initial boundary conditions and special cases look rather complicated because (1) they express the possible start of *Q* at any position in *S*, (2) the fact that the used duration ratios do not exist at the very start of a sequence and (3) the fact that the sequences start with a special start symbol.

The filling of the matrix $(D_{0...M,0...N})$ starts at $D_{0,0}$ and ends at $D_{M,N}$ in either a column-wise top-to-bottom manner or a row-wise left-to-right manner. By keeping track of each local minimization decision in the matrix in a pointer structure, one can reveal the optimal alignment between *P* and a subsequence of *S*. The entry in the column $(D_{M,0...N})$ holding the minimal distance value refers to the end of an optimal alignment. By tracing back the pointers, one can recover all local minimization decisions in reverse order that resulted in this minimal value and, hence, the starting point of the optimal alignment. Likewise, one can find multiple optimal alignments, if there are several. Or, one can find the alignments (and positions) that have a distance that is lower than a pre-defined threshold.

Since we have to compute all entries of the matrix and the computation of each entry $D_{i,j}$ is a constant factor, the worst and average case time complexity is still $\mathrm{O}(M \cdot N)$. Note that this computation has to be done for each melody in the database. A significant reduction in practical computing time without loss of performance can be obtained by leaving out the recurrent expressions for note insertion and deletion.

In principle, if we compute the matrix column-wise or row-wise, only the current column (or row) and the previous two need to be stored; only $3 \cdot \min(M, N)$ cells are required. Since $M < N$, the space required is $\mathrm{O}(M)$.

7.6 An index method: Filtering

Chances are small that a query pattern *Q* has a high approximate melodic similarity with many melodic passages *S* in the database. Leaving out subsequences in *S* that cannot have a sufficiently high similarity with *Q* saves the computation of complete columns in the dynamic programming matrix used to evaluate Equations (8) and (9). Index methods quickly retrieve parts in *S* that might be highly similar to *Q*. When these parts in *S* are identified, they still need to be evaluated by using Equation (8) and (9) to ensure whether or not they really match with *Q*.

Current index methods are based on the *k*-difference problem between Q and S using the unit-cost edit distance model. One of these index methods is known as 'filtering': parts in *S* that meet a well-defined necessary (but not sufficient) 'filtration' condition with respect to *Q* and a pre-defined error level $\alpha = k / M$ are candidate for further evaluation; all other subsequences are discarded. It is conceivable that discarded parts in *S* can still have a high melodic similarity with *Q*, as the filtering is based on the edit distance. To alleviate this discrepancy, the error level has to be set appropriately.

The used filtering method is the Chang and Lawler's LET (Linear Expected Time) algorithm [3]. It discards a subsequence of S when it can be inferred that it does not contain an approximate match with Q. This can be done by observing that a region in *S* having a *k*-approximate match with a pattern *Q* of length *M* is at least of length *M - k* and is a concatenation of at most $k+1$ longest subsequences of *Q* with intervening (non-matching) elements. So, the 'filtration' condition says that any subsequence in *S* of $k+1$ concatenated longest subsequences of *Q* that is shorter than *M - k* can be discarded. The remaining subsequences are further evaluated using Equation (8) and (9).

The algorithm uses a suffix tree on *Q* to determine in linear time the longest subsequences of *Q* in *S*. A suffix tree on a sequence *Q* is a special data structure that forms an ordered representation of all suffixes of *Q*. A suffix tree can be built in linear $\mathrm{O}(M)$ time and needs $\mathrm{O}(M)$ space [10][18].

The algorithm works by traversing *S* in a linear fashion (from left to right) and maintains the longest subsequence of *Q* at each element in *S* using the suffix tree on *Q*. When this subsequence cannot be extended any further, it starts a new subsequence of *Q* at the *next* element. Note that there is an intervening element defined between any two longest subsequences of *Q* in *S*. These elements are called *markers* in *S*.

The result is a partitioning of *S* that consists of the longest subsequences of *Q* intervened by markers. Subsequences in *S* are discarded, if they are a concatenation of $k+1$ longest subsequences of *Q* (with *k* markers) of a length that is shorter than $M-k$.

An additional result of the partitioning of *S* with respect to *P* is the number of markers in *S*. This quantity is also known as the *maximal matches* distance between *S* and *P*. This distance has been proven to be a lower bound for the unit-cost edit distance [19].

Example. Let $Q = abcba$ ($M = 5$) and $S = adaaabdbadbbb$ ($N = 13$). The partition of *S* as a concatenation of longest subsequences of *Q* intervened by markers is *a-a-ab-ba-b-b*, since *a*, *ab*, *ba* and *b* are all (longest) subsequences of *Q* in *S*. The markers have been omitted at positions 2, 4, 7, 10 and 12 in *S*. The *maximal matches* distance between *P* and S equals 5. By allowing $k = 1$ difference, the regions *a-a* and *b-b* are discarded since they are of length 3 < *M – k* = 4. On the other hand, the regions *a-ab*, *ab-ba* and *ba-b* need to be further evaluated since their lengths are $M-k$.

7.6.1 Heuristic adjustments

Filtering methods are judged on their correctness, their time complexity and their filtration efficiency.

1. **Correctness.** The LET algorithm has been proven to correctly solve the *k*-difference problem, that is, it does not miss any approximate matches in *S* in edit distance sense.

However, regions in *S* that are filtered out by LET can still be similar to *Q* in our melody distance sense.

2. **Time complexity.** The identification of candidate regions in *S* happens in linear time $O(N)$ by using a suffix tree on *Q*.

3. **Filtration efficiency.** The efficiency relates to the number of elements that can be discarded by the filter. Filtering works well on low error levels and bad or not at all on higher error levels; the filtration efficiency drops very quickly at a particular error level.

By using the *maximal matches* distance between a region in *S* and *Q* as a lower bound for their edit distance, we can further rule out regions in *S* on an heuristic basis by recognizing that *S* can have repetitive subsequences. Repetition in a melodic sequence is common; a melody can contain similar passages referring to the tune of the chorus or the individual phrases of a stanza.

We use two rule-out methods aiming at increasing the filter efficiency at a given error level. The heuristic is based on the observation that a region in *S* does not need to be evaluated again, if a similar one has already been evaluated.

1. **LET-H1**: all non-overlapping regions in *S* with equal *maximal matches* distances normalized by the length of the region to *Q* are maintained. From this set, only one region is subjected to further evaluation; all others are discarded.

2. **LET-H2**: only the region in *S* with the minimal *maximal matches* distance normalized by the length of the region to *Q* is chosen for further evaluation; all others are discarded.

It must be emphasized that these heuristic extensions make the filter no longer working correctly, since approximate matches in S are discarded on purpose. To find a balance in correctness (heuristic) level, filtration efficiency and melody comparison performance, we empirically set $\alpha = 0.25$ while using method LET-H1. As shown in Section 7.6.2, this provides us a 64% to 89% reduction in computing columns during dynamic programming for pattern sequences *Q* with a length of 12. In practice, the LET algorithm was found to be too permissive in providing still too many similar regions for further evaluation. In contrast, LET-H2 was found to be far too stringent by discarding relevant regions. Some regions in *S* that were discarded by LET-H2 turned out to have a high melodic similarity with *Q*.

7.6.2 Filtering experiment

In order to assess the filtration efficiency of the three filtering methods for typical problem instances, we conducted experiments with a varying error level α using a database with 510 popular melodies[3], each containing 285 notes on average. The filtering methods were LET, LET-H1 and LET-H2. All sequences were made out of our alphabet $\Sigma_{9\text{-step}}$ of 9 interval elements. The patterns *Q* were constructed with varying lengths (*M* = 10, 12, 14). They were either randomly chosen excerpts from the database (the melodic sequences) or randomly compiled from the alphabet (the random sequences).

A measure for *filtration efficiency* is the number of elements that are discarded divided by the total number of elements,

$$\mathit{filtration\ efficiency} = \frac{N - N_e}{N} \qquad (10)$$

where *N* denotes the total number of elements and N_e denotes the number of elements that need further evaluation. In order to decrease random variations, we have determined the averages of 250 independent runs with different patterns.

The results are shown in Table 2. The random sequences are more stringently filtered since they show little resemblance with the structure in popular melodies. It is clear that the filtration efficiency of LET has a steep drop at an error level α between 0.2 and 0.3. The use of the heuristics in LET-H1 and LET-H2 boosted the filter efficiency at each error level. An error level α of 0.25 is an appropriate parameter value when using one of the filter approaches.

Table 2. ***Filtration efficiency*** **simulated for different parameters of a problem instance ($|\Sigma_{9-step}| = 9$). Parameter combinations that resulted in zero** ***filtration efficiency*** **are not shown.**

M	*k*	α	*Melodic sequences*			*Random sequences*		
			LET	LET-H1	LET-H2	LET	LET-H1	LET-H2
10	1	0.10	0.99	1.00	1.00	1.00	1.00	1.00
	2	0.20	0.81	0.90	0.97	0.97	0.98	0.99
	3	0.30	0.22	0.37	0.76	0.35	0.49	0.85
	4	0.40	0.03	0.06	0.24	0.04	0.07	0.26
12	1	0.08	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.17	0.93	0.96	0.98	1.00	1.00	1.00
	3	0.25	0.53	0.64	0.89	0.81	0.89	0.96
	4	0.33	0.11	0.16	0.45	0.14	0.23	0.52
	5	0.42	0.02	0.04	0.13	0.02	0.03	0.12
14	1	0.07	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.14	0.98	0.99	0.99	1.00	1.00	1.00
	3	0.21	0.77	0.83	0.94	0.96	0.97	0.99
	4	0.28	0.28	0.38	0.70	0.45	0.58	0.86
	5	0.36	0.06	0.08	0.24	0.06	0.08	0.25

8. CONCLUSIONS

CubyHum is a software system in which 'query by humming' is realized by linking algorithms from various fields: speech signal processing, music processing and approximate pattern matching. Empirical findings from singing experiments were a crucial input to the development of these algorithms. In short, it tries to detect the pitches in a sung melody and compares these pitches with symbolic representations of melodies in a large database. Melodies that are similar to the sung pitches are retrieved. Approximate pattern matching in the melody comparison process compensates for the errors in the sung melody (e.g., sharp or flat notes, wrong tempo) by using classical dynamic programming. A filtering technique saves much of the computing necessities involved in dynamic programming.

CubyHum has been integrated in an in-house research demonstrator, the 'Easy Access' music jukebox, in which innovative user interface solutions supporting various user search strategies and intentions in music retrieval are demonstrated (see Figure 4). Besides 'query by humming', this Internet-connected jukebox incorporates

speaker identification for personalization purposes,

collaborative filtering for recommending new music,

navigation and playlist creation features by voice and pointing gestures, and

[3] The melody database contained 510 vocal monophonic melodies from songs of the Beatles (185), ABBA (73), the Rolling Stones (67), Madonna (38), David Bowie (34), U2 (33), Prince (23), Michael Jackson (20), Frank Sinatra (20) and the Police (17).

system feedback by text-to-speech synthesis and auditory cues.

Using this personalized jukebox, a user can simply name, sing and point at songs to listen to or to collect them in a playlist.

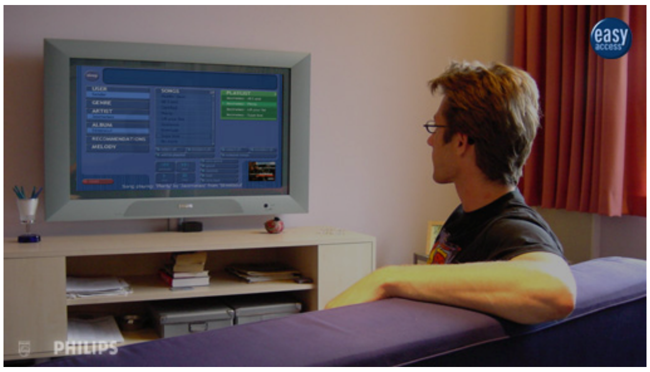

Figure 4. The 'Easy Access' music jukebox.

Some formal user studies and evaluations on 'query by humming' (and the Jukebox as a whole) have already been finalized. Their findings guide further algorithmic improvement and tell what usability issues for a 'query by humming' system are prevalent. As a conclusion, we would like to address the following recommendations for further research.

8.1 Pitch detection

The current pitch detection algorithm (sub-harmonic summation) has been developed for normal speech for which it works reliably. It has been made robust with respect to deviant pitch values by smoothing the pitch contour in a dynamic programming framework. Further research has to be pursued to detect pitch reliably for highly pitched tones, inharmonic sounds and severely degraded acoustical or channel conditions.

8.2 Event detection

The current event detection is based on standard signal processing algorithms. For best performances, users are recommended to sing the notes of their melody in an isolated manner using a non-sense syllable of an unvoiced fricative and a long vowel (e.g., '/fa/-/fa/-/fa/'). Although this isolated way of singing can be easily taught and learnt, it takes away any opportunities for expressive and free-style singing.

Further research has to be pursued to detect musical events robustly and reliably allowing users to sing in any preferred style. For instance, the finding of note onsets can be helped by robust vowel onset detection mechanisms and the use of parametric models that detect abrupt signal changes or employ the presence of stationary signal segments. Moreover, singing contains all kinds of expressive means that largely go unnoticed by the current event detection method. If, for instance, vibrato or accentuation can be reliably detected, these cues can be used to extend the melody comparison process.

8.3 Approximate pattern matching

Computing requirements are dominated by approximate pattern matching. It turned out that melody comparison by means of classical dynamic programming is impractical in terms of running time performances for large melody databases. Using the current filtering method, current response times take a few seconds for a database with only 510 melodies on a current platform. From a usability point of view, this has to be reduced to half a second for a database with many more melodies.

Research on fighting the inherent $O(M \cdot N)$ time complexity of dynamic programming is pivotal. There are essentially two approaches to tackle this challenge. Index methods for approximate pattern matching allow a search to jump swiftly to candidate approximate matches in the database; this field is new and rather immature. In addition, some recurrent expressions for finding approximate matches can be evaluated by a fast bit-parallel implementation of dynamic programming. These algorithms exploit the intrinsic parallelism of bit-vector operations. If the length of the pattern P is smaller than the size of the computer word, they can run in essentially linear time [1][13].

9. ACKNOWLEDGMENTS

Thanks go to Dik Hermes (Technische Universiteit Eindhoven) for telling me everything about pitch perception, Sander van de Wijdeven for helping me C++debugging 'CubyHum' and all members of the Philips 'Easy Access' research project team.

10. REFERENCES

[1] Baeza-Yates, R., and Navarro, G. (1999). Faster approximate string matching. *Algorithmica 23, 2*, 127-158.

[2] Brown, J.C., and Vaughn, K.V. (1996). Pitch center of stringed instrument vibrato tones, *Journal of the Acoustical Society of America, 100*, 1728-1735.

[3] Chang, W., and Lawler, E. (1994). Sublinear approximate string matching and biological applications. *Algorithmica, 12, 4/5*, 327-344.

[4] Dowling, W.L. (1978). Scale and Contour: Two components of a theory of memory for melodies. *Psychological Review, 85, 4*, 341-354.

[5] Dowling, W.J., and Harwood, D.L. (1986). *Music cognition*. New York: Academic Press.

[6] Ghias, A., Logan, J., Chamberlin, D., and Smith, B.C. (1995). Query by humming: Musical information retrieval in an audio database. *Proceedings of the ACM international Multimedia conference and exhibition, November 1995, San Francisco, California*. New York: ACM, 231-236.

[7] Hermes, D.J. (1988). Measurement of pitch by subharmonic summation. *Journal of Acoustical Society of America, 83, 1*, 257-264.

[8] Levitin, D.J. (1994). Absolute memory for musical pitch: Evidence from the production of learned melodies. *Perception & Psychophysics, 58*, 927-935.

[9] Masri, P., and Bateman, A. (1996). Improved modelling of attack transients in music analysis-resynthesis. *Proceedings of International Computer Music Conference (ICMC 96), Hong-Kong, Aug 1996*, International Computer Music Association, 100-103.

[10] McCreight, E. (1976). A space-economical suffix tree construction algorithm. *Journal of ACM, 23, 2*, 262-272.

[11] McNab, R.J., Smith, L.A., Witten, I.H., and Henderson, C.L. (2000). Tune retrieval in the multimedia library, *Multimedia Tools and Applications, 10*, 113-132.

[12] Mongeau, M., and Sankoff, D. (1990). Comparison of musical sequences. *Computers and the Humanities, 24*, 161-175.

[13] Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM, 46, 4*, 395-415.

[14] Peynirçioglu, Z.K., Tekcan, A.I., Wagner, J.L, Baxter, T.L., and Shaffer, S.D. (1998). Name or hum that tune: Feeling of knowing for music, *Memory & Cognition, 26, 6*, 1131-1137.

[15] Rabiner, L.R. and Juang, B. (1993). *Fundamentals of Speech Recognition*, Prentice-Hall Inc.

[16] Schloss, W. (1985). *On the Automatic Transcription of Percussive Music: From Acoustic Signal to High Level Analysis*, PhD Thesis, Department of Music, Report No. STAN-M-27, Stanford University, CCRMA.

[17] Sellers, P.H. (1980). The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms, 1*, 359-373.

[18] Ukkonen, E. (1995). Constructing suffix trees on-line in linear time. *Algoritmica, 14, 3*, 249-260.

[19] Ukkonen, E. (1992). Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science, 92, 1*, 191-211.

[20] Wagner, R.A. and Fischer, M.J (1974). The string-to-string correction problem, *Journal of the Association of Computing Machinery, 21, 1*, 168-173.