

Retrieving musical information based on rhythm and pitch correlations

Will Archer Arentz Magnus Lie Hetland
willa@idi.ntnu.no magnus@hetland.org

Bjørn Olstad
bjorn.olstad@fast.no

Abstract

Music Information Retrieval and Query-by-Humming systems has recently been given much attention. One of the reasons for this is the diversity of attractive applications that can be developed with these techniques. Imagine having a tune on the tip of your tongue, but you cannot really place it. What is the name of the song and who is the artist? The proposed method is designed to help in pursuing such request. Both pitch and rhythmic information is utilized to determine the most closely matched song for any given theme. This also includes polyphonic songs, which may not contain the wanted theme in the main melody, but rather in one of it's accompaniments.

Due to human's poor ability to accurately reproduce a piece of music, whether it is caused by poor memory or poor music skills, such methods must be robust in a way that makes people's perception of similarity highly correlated with the distances calculated by the algorithm. The proposed dynamic programming algorithm finds 93% of the wanted hits among the top-10 results when the timing is distorted with a standard deviation of 0.2 seconds and the pitch is distorted with a standard deviation of 0.3 notes, using 7 note queries and a 1564 song MIDI database.

1 Introduction

Typically, people do not memorize everything about a song the first time they hear it. Information such as title, composer and performer, are often learned at a much later stage of a person's relationship with a song. Furthermore, as the human brain often forgets this information, while the melody remains fresh in mind, it

seems obvious that an application capable of retrieving music from humming or whistling can be very useful.

The recent development in music information retrieval (MIR) has brought forward several promising query-by-humming (QBH) systems that attempt to develop applications where the hummed or whistled theme of a song can be search through a large database, thus returning the closest matching song. The input interface to such a system may for example be a digital piano or simply a microphone, where the user is requested to whistle or sing the query into the microphone, before a signal processing system turns the audio into musical notes. The processed sequence of notes often has several errors in comparison to the original piece of music. A major reason for this is human's poor music-reproduction-ability. For example, the theme may sound similar, even if a few notes are missing, the key is wrong, the tempo is too slow and a long tone was reproduced as 3 shorter tones. A QBH system should calculate the distance based on perceptible differences. The method proposed in this article has attempted to take such information into account, and it can therefore provide a very robust and accurate search, even for a very short query.

1.1 Prior work

Much work has recently been done in the area of music information retrieval and query-by-humming systems. Some of it is summarized in Pickens' survey of feature selection techniques in MIR (Pickens, 2001). Obviously, the most basic approach is to base the search on a monophonic sequence of notes, with their accompanying pitch and duration, simplifying the problem to one dimension. Alternatively, the pitch can be extracted and the duration is ignored, or vice versa. Both pitch and duration may be used in the final system, but in most work these features are treated separately. Exceptions to this can be found in (Lemstrom, Laine, & Perttu, 1999; Chen et al., 2000). Furthermore there is a question of using absolute or relative measures, where relative is the most popular, because changes in tempo or transposition across keys do not significantly alter the music information expressed (Ghies, Logan, Chamberlin, & Smith, 1995; Lindsay, 1994; Lemstrom et al., 1999; Blackburn & Roure, 1998).

The use of dynamic programming (DP) to calculate the distance between a set of query notes and all the notes within all the songs in the database in a sliding window manner, has recently been attempted (Song, Bae, & Yoon, 2002; Pauws, 2002). Both projects have limited the research to only include the pitch feature, thus ignoring the information in the duration feature. However, they both claim to have fully operational query-by-humming systems. Also, an approach using dynamic time warping on frame-based queries has been attempted (Mazzoni & Dannenberg, 2001). This paper shows how to include the timing feature as well

as the pitch and thus filling the gap in prior work.

This paper is organized as follows: Section 2 explains the proposed algorithm, followed by Section 3, which goes through the experimental results. Finally, Section 4 presents some concluding remarks, and points out some possible future research.

2 The Algorithm

A tune or motif can be represented as a sequence of pitch values, $p(z_i)$, and a corresponding timestamp function $t(z_i)$ which gives the starting point for each note. Two tunes $x = x_1, \dots, x_n$ and $y = y_1, \dots, y_n$ are said to match if $p(x_i) = p(y_i)$ for $i = 1 \dots n$. To achieve key invariance a match can be redefined as $p(x_i) - p(x_{i-1}) = p(y_i) - p(y_{i-1})$ for $i = 2 \dots n$. In other words, the semitones of x must equal the semitones of y .

When comparing a query motif $q = q_1, \dots, q_m$ with a tune $s = s_1, \dots, s_n$ from a music database, we would like to find a subsequence in s that matches q . This means that all notes in the query must be accounted for, but the matching notes from s need not be contiguous.

Formally, we define an *alignment* of a query q and a tune s as a strictly increasing sequence of indices $i = i_1, \dots, i_m$. A matching alignment is an alignment i such that s_{i_1}, \dots, s_{i_m} matches q . How this match is found will be explained later in this section. Figure 1 shows a query q aligned with a tune s . Notice that when the alignment skips a note in the song, this is equivalent to the consolidation of two notes since only note on sets are considered.

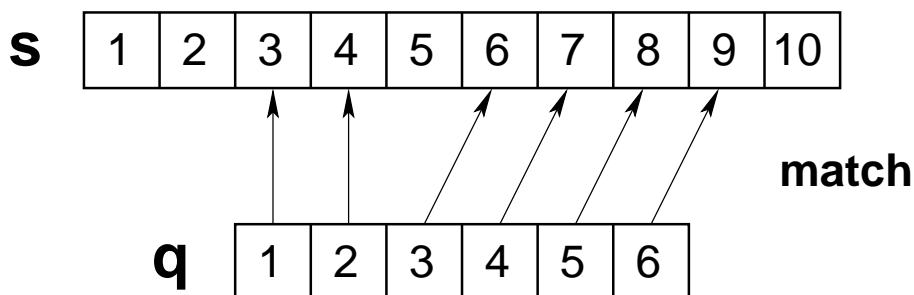


Figure 1: Match of query q and tune s with alignment $i = 3, 4, 6, 7, 8, 9$

Given the notion of a matching alignment, we can now define a dissimilarity

measure between a query q and a tune s for a specific alignment i , as follows:

$$d(q, s, i) = \sum_{j=2}^m w(q_{j-1}, q_j, s_{i_{j-1}}, s_{i_j})^2, \quad (1)$$

where $w(q_a, q_b, s_c, s_d)$ represents the cost of pairing up the note pair (q_a, q_b) in the query with the note pair (s_c, s_d) in the tune. A temporal cost function w can be defined as

$$w(q_a, q_b, s_c, s_d) = \alpha \{t(s_d) - t(s_c)\} - \{t(q_b) - t(q_a)\}, \quad (2)$$

where α is a scaling factor, used to account for tempo differences between the two tunes, and $t(s_i)$ is the timestamp for the given note $s_i \in s$. The dissimilarity between a query q and a tune s can then be defined as

$$d(q, s) = \min_i \{d(q, s, i)\}. \quad (3)$$

For a given alignment i , $d(q, s, i)$ is minimized by choosing α as

$$\alpha = \frac{\sum_{j=2}^m (t(s_{i_j}) - t(s_{i_{j-1}}))(t(q_j) - t(q_{j-1}))}{\sum_{i=2}^m (t(s_{i_j}) - t(s_{i_{j-1}}))^2}. \quad (4)$$

Since the optimal value for α is given as a function of the alignment i , the optimization task in (3) becomes a matter of finding an optimal alignment.

Assuming, for now, that the value of α is known, the optimization may be expressed recursively as follows:

$$d(q_{1:a}, s_{1:b}) = \min_c \{d(q_{1:a-1}, s_{1:c}) + w(q_{a-1}, q_a, s_c, s_b)^2\} \quad (5)$$

where $q_{1:a}$ and $s_{1:b}$ are prefixes of q and s , of length a and b , respectively.

This equation may be solved iteratively, by dynamic programming (Bellman, 1957). The basic solution simply consists of constructing a two-dimensional array E of size $m \times n$ for storing the partial solutions, and iterating over the two prefix-lengths. See (Sankoff & Kruskal, 1999) for examples of the same technique applied to the problem of computing the Levenshtein distance (edit distance) and Euclidean distance under dynamic time warping. An application of the basic Levenshtein distance algorithm to timestamped data, similar to ours, can be found in (Mannila & Ronkainen, 1997).

There are two issues that must be addressed before such a dynamic programming solution can be implemented: We need to find the value of α , and we need to determine the allowable values for c in (5).

Preferably, α should be computed according to (4), but not all the required values are available during the stepwise computation of the dynamic programming

algorithm. We approximate the α associated with a given c when calculating $d(q_{1:a}, s_{1:b})$ with a recursive filter, as follows:

$$\hat{\alpha}(a, b) = \begin{cases} 1, & a = 1 \\ \hat{\alpha}(a - 1, c), & a > 1, t(s_a) - t(s_c) < \varepsilon \\ \beta \cdot \hat{\alpha}(a - 1, c) + (1 - \beta) \cdot \frac{t(q_a) - t(q_{a-1})}{t(s_b) - t(s_c)}, & a > 1, t(s_a) - t(s_c) \geq \varepsilon \end{cases} \quad (6)$$

The β value should be relatively high to avoid that the matching algorithm degenerates, accepting any rhythm variations due to note-to-note variations in $\hat{\alpha}$. We used $\beta = 0.85$ in our experiments. Implementing this in a DP framework would mean introducing a second two-dimensional array α of dimension $m \times n$. In this way the scaling factor may be iteratively updated according to (6) to reflect the scaling factor that should be applied when calculating a partial solution.

The optimization parameter c will take on the values $b-1, b-2, \dots$. The c is simply the index of the note in the tune s that was matched by the previous note in the query q ; if we restricted c to $b-1$, no extra notes would be allowed between the notes in the query. As a way of pruning the search for an optimal alignment, we may restrict the values of c , either by setting an absolute limit on the number of extra notes allowed in the tune between two query notes, or by placing an upper limit on the ratio

$$\frac{\hat{\alpha}(t(s_b) - t(s_c))}{t(q_a) - t(q_{a-1})},$$

as well as placing upper and lower limits on $\hat{\alpha}$ (for example, 2 and 0.5, respectively). To compute c , a heuristic cost function is introduced. As a part of this, the pitch error is calculated as

$$\varepsilon_{pitch}(a, b) = |p(s_c) - p(s_b) - p(q_{a-1}) + p(q_a)|, \quad (7)$$

where the pitch values are MIDI note values multiplied by ten. This increase in pitch resolution allows for more accurate results from the pitch tracker, giving a more exact match for the cases where the pitch is slightly closer to the wrong, than the right note.

The proposed cost function for pitch error,

$$C_p(\varepsilon_p) = \begin{cases} \Omega_1 \cdot \varepsilon_p & \varepsilon_p \leq 10 \\ \Omega_1 + \frac{\Omega_2 - \Omega_1}{10}(\varepsilon_p - 10) & \varepsilon_p > 10 \end{cases}, \quad (8)$$

takes the pitch error as argument and calculates a cost accordingly. The constants Ω_1 and Ω_2 are the costs for a difference of one and two semitones', respectively. The values $\Omega_1 = 360000$ and $\Omega_2 = 1000000$ proved to be reasonable values. Although all constants were originally set by trial and error, they were later verified or optimized by genetic algorithms.

Given the pitch error cost, follows the temporal error cost. Defining the alignment cost as

$$C_t(a, b) = \alpha(a - 1, c) \cdot \{t(s_b) - t(s_c)\} - \{t(q_a) - t(q_{a-1})\}, \quad (9)$$

the following function for “previous song note cost” is suggested:

$$E(a, b) = \min_c \{E(a - 1, c) + C_t^2(a, b) + C_p(\varepsilon_p) + (b - c - 1) \cdot C_{\text{additionalnote}}\}. \quad (10)$$

Although several cost functions and parameter values were investigated, using random queries with different levels of added noise, (10) proved to be most promising. The constant $C_{\text{additionalnote}}$ determines the penalty added for skipping a note. A good choice is $C_{\text{additionalnote}} = 160000$.

Pseudocode that illustrates the main ideas of the algorithm can be seen in figure 2 on the following page. The algorithm can be divided into three sections; initialization, DP-table computation, and reading and returning best match. In the first section every value in the first row of the DP-table is set to 0, thus allowing for transposed queries. The second section loops over all notes in the query (q_i) and all notes in the song (s_j). The values k_{\min} and k_{\max} indicates the range of notes that can be skipped in the song. This range is limited by the constant maxskip . As k loops over the candidates for previous song note, the cost is computed according to (10) and the lowest inserted in the DP-table. Finally, in the third section the lowest value on the last row indicates the best match and is thus returned.

Note that ε denotes a small value, for example 5 milliseconds. This is to adjust for inaccuracies occurring during manual entering of chords. The running time for the algorithm is $O(mn)$ for each song in the database.

3 Experimental results

The described algorithm was tested by interfacing it with a pitch tracker, which converted a hummed or whistled query to notes. Although a pitch tracker was developed for the proposed query-by-humming system, only the matching algorithm is described and discussed in this paper. Moreover, although the pitch tracker and matching algorithm were capable of producing and alternatives with different probabilities for each choice of pitch in the query, this capability was not used in any of the experiments.

To achieve a more accurate measurement of the system’s actual performance, a new evaluation-scheme was set up. The idea was therefore to extract a theme from a random place in a song with some set length l , and then modify this theme before using it as a query on the database. The database used in the experiments contained 1564 monophonic songs. These songs were single channel MIDI files

```

match( $q, s$ )
  for  $j \in 1 \dots n$ 
     $E[1, j] := 0$ 
     $\alpha[1, j] := 1$ 
  for  $i \in 2 \dots m$ 
    for  $j \in i \dots n$ 
       $k_{max} := j - 1$ 
       $k_{min} := k_{max} - maxskip$ 
       $E_{best} := \infty$ 
       $\alpha[i, j] := 1$ 
       $\delta := t(q_i) - t(q_{i-1})$ 
       $rightnote := p(s_j) + p(q_{i-1}) - p(q_i)$ 
       $k_{best} := -1$ 
      for  $k \in k_{min} \dots k_{max}$ 
         $\varepsilon_{pitch} := |p(s_k) - rightnote|$ 
         $C_{\varepsilon_{pitch}} := PitchErrorCost(\varepsilon_{pitch})$ 
         $\Delta := \alpha[i - 1, k] \cdot (t(s_j) - t(s_k)) - \delta$ 
         $E_{cur} := E[i - 1, k] + \Delta^2 + C_{\varepsilon_{pitch}} + (j - k - 1) \cdot C_{additionalnote}$ 
        if  $E_{cur} < E_{best}$ 
           $k_{best} := k$ 
           $E_{best} := E_{cur}$ 
       $E[i, j] := E_{best}$ 
      if  $t(s_j) - t(s_{k_{best}}) < \varepsilon$ 
         $\alpha[i, j] := \alpha[i - 1, k_{best}]$ 
      else
         $\alpha[i, j] := \beta \cdot \alpha[i - 1, k_{best}] +$ 
           $(1 - \beta) \cdot (\delta / t(s_j) - t(s_{k_{best}}))$ 
       $best := \infty$ 
    for  $j \in m \dots n$ 
      if  $E[m, j] < best$ 
         $best := E[m, j]$ 
  return  $best$ 

```

Figure 2: Pseudocode for the matching algorithm.

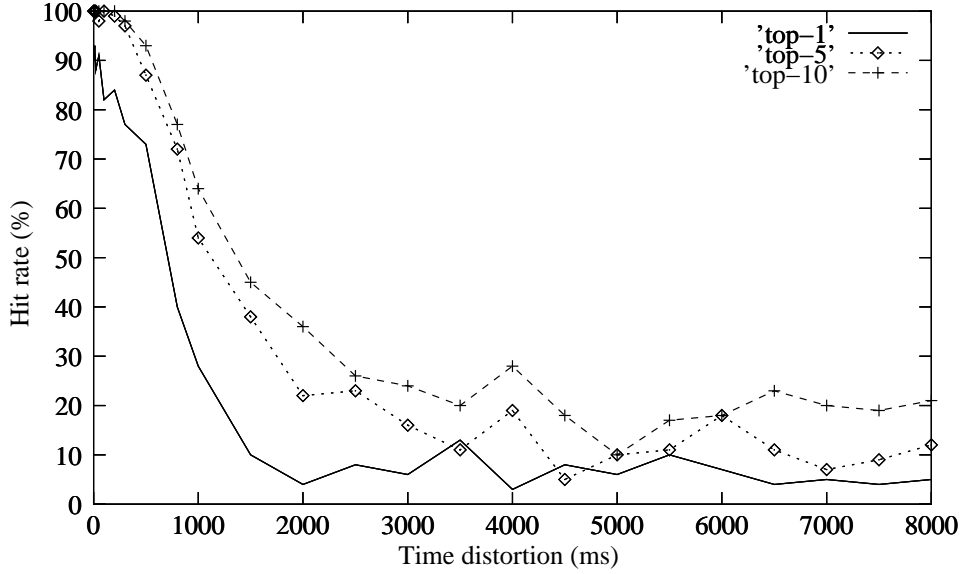


Figure 3: The algorithm’s hit rate as a function of time distortion(σ_{time})

(Loy, 1985), which were used as ring tones on mobile phones. The MIDI files were found by searching on the Internet. The proposed algorithm can handle polyphonic songs as well, but for simplicity only monophonic songs were used for the experiments in this paper, as the handling of polyphonic files is basically traversing each channel sequentially, as if they were separate files. In the database used in the experiments, several recordings were present for some of the songs.

When simulating the errors of humans as they try to recreate a musical piece, it is important to understand that the ability to accurately reproduce the right pitch at the right time varies greatly. The timing is especially important in this context, as this is where most of the errors occur. By measuring the errors of a few musicians trying to perfectly recreate a piece of music, we discover that the errors can be approximated by the normal distribution:

$$n(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Thus, we can sample new values into a query $q = q_1, \dots, q_l$, from the exact motif $m = m_1, \dots, m_l$ extracted from the song.

The time of the query element q_i is set by

$$t(q_i) = [\sqrt{-2\ln(U_2)}\cos(2\pi U_1)]\sigma_{time} + t(m_i) \quad (11)$$

where U_1 and U_2 are uniformly distributed random numbers, and σ is set to some appropriate standard deviation. The greater the standard deviation, the greater

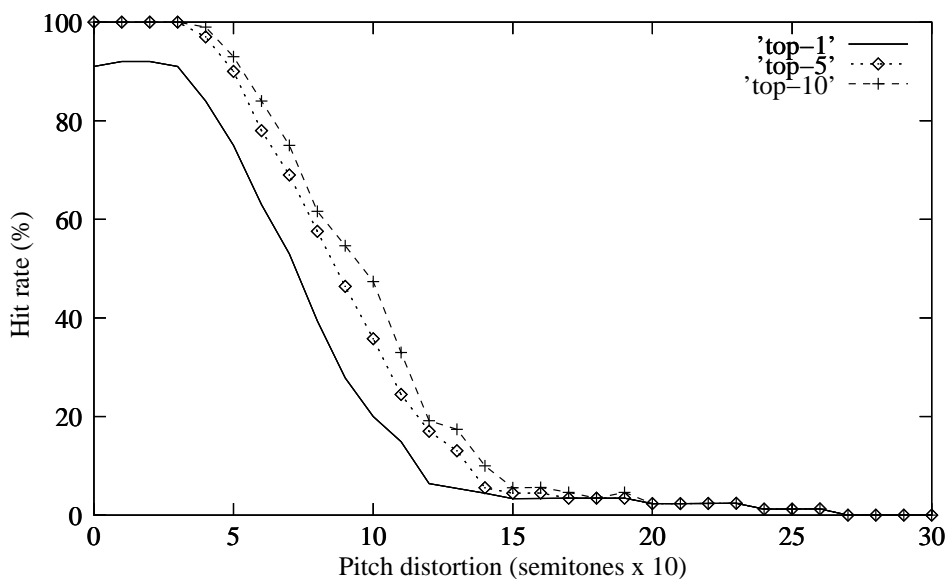


Figure 4: The algorithm’s hit rate as a function of pitch distortion(σ_{pitch})

the error introduced into the query. Figure 3 shows how the increasing standard deviation decreases accuracy in the result. For each chosen standard deviation, one hundred queries were run, and the results measured as being the top match, among the top 5 or among the top 10 result hits. Each query had a size of 10 notes, which will usually suffice to get a good result. Throughout the experiments it was decided to use 100 queries to determine the hit rate. That means the percent of the queries that found its match among the top- n (usually top-10) results from the 100 queries run through the search algorithm.

Furthermore, since some songs are manually typed into a computer or mobile phone rather than played on an instrument, several songs may have the same sequence of 10 notes and thereby leaving additional potential for errors. And, more importantly, some songs had more than one entry in the database, but with different names. This makes it impossible for the algorithm to know which one we are searching for. Thus, the top-1 measure is not very reliable, and the top-5 and especially the top-10 should be the focus of our attention. It can be seen from figure 3 that even with a standard deviation of 8000 milliseconds, that is 8 seconds, the requested result is among the top-10 in 21% of the queries.

Figure 4 shows how the algorithm performs as the pitch is distorted using the Box-Muller method as described in (11). Also here the query consisted of 10 notes and 100 queries were tested for each choice of σ_{pitch} . The values describing pitch changes (x-axis) are the number of semitones between two notes, multiplied

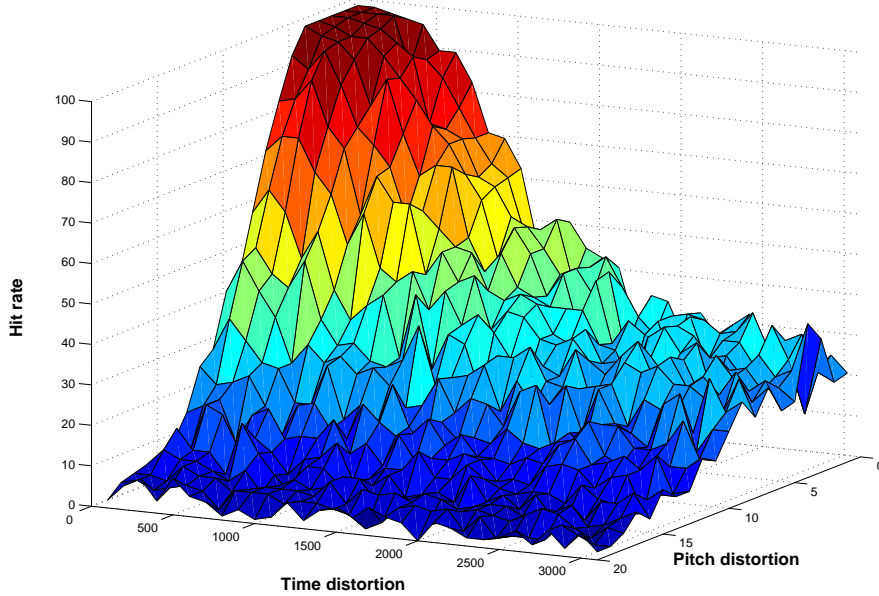


Figure 5: The algorithm’s hit rate as a function of time and pitch distortion

by 10. That is, if a C equals 600, then a C# would be 610. It can be seen that the penalty for making a pitch mistake is higher than the penalty for making a severe rhythmic mistake. This is due to the fact that humans perceive pitch mistakes as significantly more serious than timing mistakes (Shmulevich, Yli-Harja, Coyle, Povel, & Lemstrom, 1999). Our algorithm should therefore act on this principle.

By adding distortion both to the pitch and to the time, the hit rate response of the algorithm can be seen from figure 5. Here a “hit” is defined as included in the top-10 matches returned from the algorithm. For each point in the graph 100 queries consisting of 10 notes were run through the algorithm.

However, the hit rate also depends on the number of notes in the query. In figure 6 it can be seen how the algorithm’s ability to find the right results improves as the query length increases. In this figure’s bottom graph, distortion was introduced in both the time ($\sigma_{time} = 500$) and the pitch ($\sigma_{pitch} = 5$) data.

Table 1 shows how the length of the query affects the hit rate, in some typical distortion scenarios. It should be noted that even with as few as 5 notes in the query, good results can be achieved with medium distortion.

Although these experiments, using fabricated queries with statistically sam-

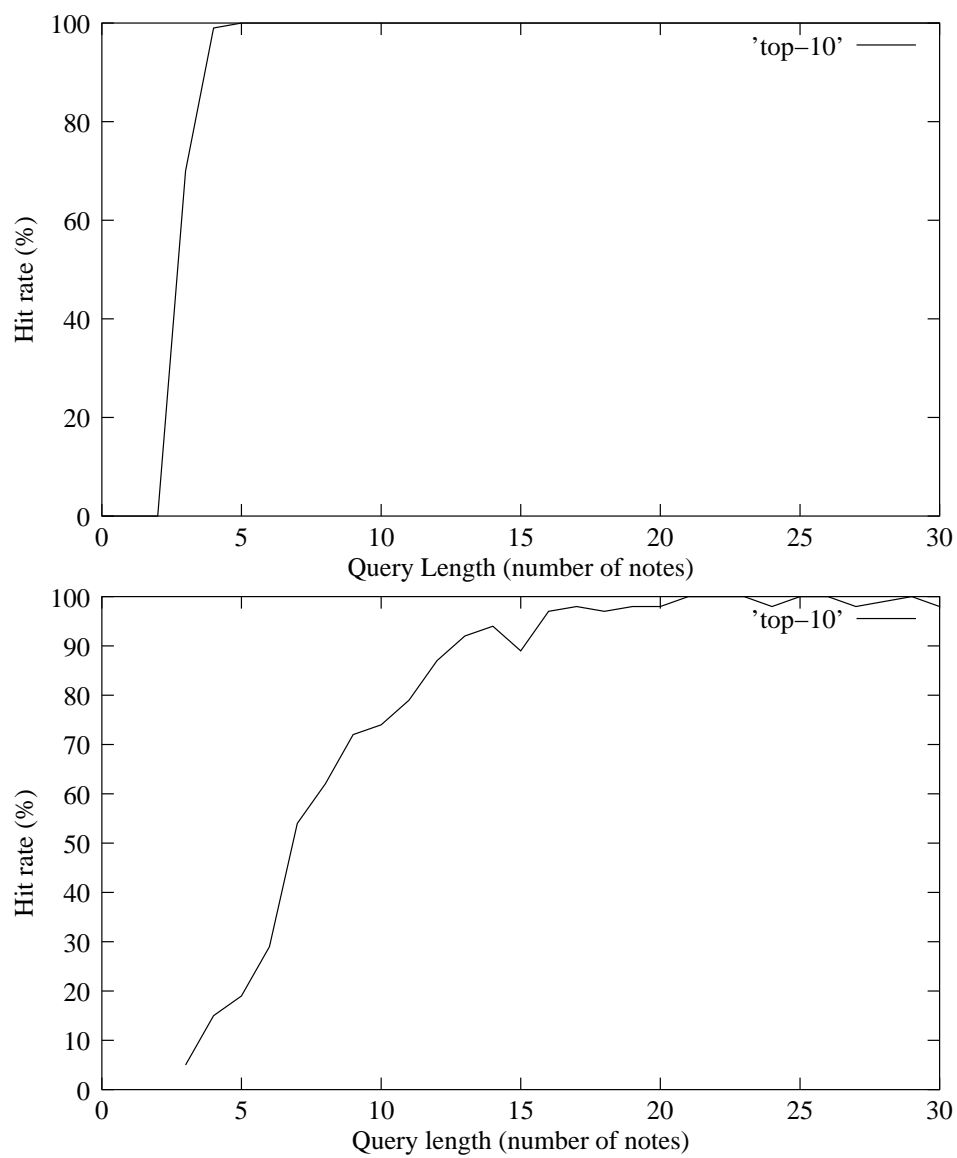


Figure 6: The algorithm's hit rate as a function of query length (top: $\sigma_{time} = 0, \sigma_{pitch} = 0$, bottom: $\sigma_{time} = 500, \sigma_{pitch} = 5$)

Table 1: The hit rate (% in top-10 from tested 100 queries) as function of the query length and some typical distortions

Query length	3	5	7	10	15	20	25
$\sigma_{time} = 0$ $\sigma_{pitch} = 0$	76	100	100	100	100	100	100
$\sigma_{time} = 50$ $\sigma_{pitch} = 1$	43	93	100	100	100	100	100
$\sigma_{time} = 100$ $\sigma_{pitch} = 2$	22	89	100	100	100	100	100
$\sigma_{time} = 200$ $\sigma_{pitch} = 3$	15	63	93	98	100	100	100
$\sigma_{time} = 300$ $\sigma_{pitch} = 4$	16	37	74	94	100	100	100
$\sigma_{time} = 400$ $\sigma_{pitch} = 5$	5	29	53	78	96	100	100
$\sigma_{time} = 500$ $\sigma_{pitch} = 6$	4	22	47	59	96	95	97

pled errors can say a lot about the algorithm’s capabilities, experimenting with a real-world data set may be necessary to ensure robustness against typical human errors. Errors, such as gradual or sudden tempo changes, key changes, and skipping or adding notes were not tested for with the fabricated queries.

Even though the MAMI project (Lesaffre et al., 2003 (ISMIR 2003)) has a great collection of freely available audio queries, they do not include a corresponding MIDI database. The data set used by Mazzoni and Dannenberg (Mazzoni & Dannenberg, 2001) was therefore chosen, for comparing our approach with their frame-based dynamic time warping,

where the authors suggested to use sequences of 10ms frames - assigning one pitch value to each frame - as basis for their DTW search. While the database scores are easily converted to frames, the authors used a “pitch tracker” application to transform audio queries into sequences of frames. The data set was composed of 481 single-track songs and 136 queries, both in MIDI format. The audio recordings used to generate the MIDI queries were not available, we did not have any control over the note segmentation process. Although the queries also were available in the frame-format used in Mazzoni and Dannenberg’s experiments, these could not simply be converted to MIDI.

Mazzoni and Dannenberg’s use of frames instead of musical scores, introduce a potentially significant source of error in the process, because we don’t know

much about the algorithm converting a query to MIDI compared to the one for converting to frame-sequences.

Mazzoni did, however, experiment both with frame-based and score-based matching (Mazzoni, 2002). The reported results for these experiments were summarized as 77% of the queries resulting with a match among the top-10 search results for the frame-based approach, and 54% for the score-based. It is finally concluded that the frame based approach, although very slow, is superior to any score-based matching scheme for QBH systems. Although there are a number of issues not handled by the DTW algorithm (i.e. skipping notes, additional note, changing key), it seems reasonable that a frame based Dynamic Programming algorithm would give good results for QBH systems. This is especially true, because the fine details of the input are kept and used in a non-heuristic search.

Using the same score-based queries; we were able to get 67% among the top-10 with the presented method. Although well below Mazzoni's frame-based approach, several factors have to be considered when comparing the two methods.

Firstly, the score segmentation is believed to be the primary source for error. Furthermore, the DTW method was unable to robustly handle tempo and key changes, although these errors were not extensively tested for in the present data set. Finally, it should be noted that the DTW algorithm only matched against the beginning section of each song, as all queries were composed, starting at the beginning of the song. The proposed method did, however, match against all possible compositions of seven notes (excluding possible skipped notes) in each song, using only the first seven notes in the query. In other words, we searched through a much larger search space than the DTW method.

The current implementation of the search engine can handle more than thousand polyphonic songs in each collection, and still search through a collection in a fraction of a second. As the running time of the matching algorithm is linear, any growth in the database size will require an equal growth in CPU power to retain search speeds. In the proposed system, large databases were divided into collections, which were searched through individually with different CPUs/computers, before finally collecting, sorting and representing the results.

4 Conclusion

This paper has presented a method for accurately searching in music databases with short queries, allowing distortion in both the pitch and time domain. The dynamic programming algorithm finds 93% of the wanted hits among the top-10 results when the timing is distorted with a standard deviation of 0.2 seconds and the pitch is distorted with a standard deviation of 0.3 semitones ($\sigma_{pitch} = 3$), using only 7 note queries.

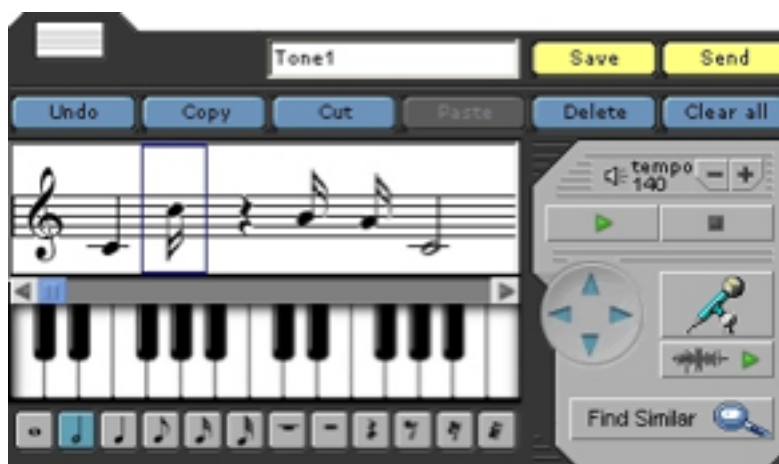


Figure 7: A screenshot from our query-by-humming interface

The algorithm has currently a linear running time, so future work includes re-searching how to index the search so that the speed can be increased. Also, the comparison with Mazzoni's work suggests that a frame-based version of the algorithm may prove interesting. Furthermore, a generalized framework for applying and adjusting the algorithm to other applications also deserves further investigation.

All in all the proposed method has shown to be very successful, and it has generated much commercial interest, as well as being incorporated into several applications such as a full query-by-humming system (see figure 7), a mobile ring tone search application and a real-time search and assist feature on MIDI keyboards.

References

- Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.
- Blackburn, S., & Roure, D. D. (1998). A tool for content-based navigation of music. In *Proceedings of ACM International Multimedia Conference (ACMMM)*. Retrieved April 07, 2005, from <http://eprints.ecs.soton.ac.uk/725/02/html>.
- Chen, A. L. P., Hsu, J.-L., Chang, M., Chen, J., Hsu, C.-H., & Hua, S. Y. S. (2000). Query by Music Segments: An Efficient Approach for Song Retrieval. In *Proceedings of IEEE International Conference on Multimedia and Expo*. Retrieved April 07, 2005, from <http://make.cs.nthu.edu.tw:8000/asp/query2.asp>.

- Ghias, A., Logan, J., Chamberlin, D., & Smith, B. C. (1995). Query by Humming: Musical Information Retrieval in an Audio Database. In *ACM Multimedia* (p. 231-236).
- Lemstrom, K., Laine, P., & Perttu, S. (1999). Using Relative Interval Slope in Music Information Retrieval. In *Proceedings of International Computer Music Conference* (p. 317-320).
- Lesaffre, M., Tanghe, K., Martens, G., Moelants, D., Leman, M., Baets, B. D., et al. (2003 (ISMIR 2003), October 26-30). The MAMI Query-By-Voice Experiment: Collecting and annotating vocal queries for music information retrieval. In *Proceedings of the International Conference on Music Information Retrieval*.
- Lindsay, A. (1994). *Using contour as a mid-level representation of melody*. Master Thesis, Massachusetts Institute of Technology, Retrieved April 07, 2005, from <http://citeseer.nj.nec.com/lindsay96using.html>.
- Loy, G. (1985). Musicians Make a Standard: The MIDI Phenomenon. In *Computer Music Journal* 9(4) (p. 8-26).
- Mannila, H., & Ronkainen, P. (1997). Similarity of Event Sequences (revised version). In *Proceedings of the Fourth International Workshop on Temporal Representation and Reasining, TIME 97* (p. 136-139).
- Mazzoni, D. (2002, August). *Tech Report: Melody Matching Using Time Warping* (Tech. Rep.). Harvey Mudd College.
- Mazzoni, D., & Dannenberg, R. B. (2001). Melody Matching Directly From Audio. In *Proceedings of ISMIR* (p. 17-18). Bloomington: Indiana University.
- Pauws, S. (2002). Cubyhum: A Fully Operational Query by Humming System. In *Proceedings of ISMIR* (p. 187-196).
- Pickens, J. (2001). A survey of feature selection techniques for music information retrieval. *Technical report, Center for Intelligent Information Retrieval, Department of Computer Science, University of Massachusetts*. Retrieved April 07, 2005, from <http://citeseer.ist.psu.edu/pickens01survey.html>.
- Sankoff, D., & Kruskal, J. (Eds.). (1999). *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison* (Reissue ed.). CSLI Publications.
- Shmulevich, I., Yli-Harja, O., Coyle, E., Povel, D., & Lemstrom, K. (1999, April). Perceptual issues in music pattern recognition — complexity of rhythm and key finding. In *Proceedings of AISB Symposium on Musical Creativity* (p. 64-69). Edinburgh, United Kingdom.
- Song, J., Bae, S. Y., & Yoon, K. (2002). Mid-Level Music Melody Representation of Polyphonic Audio for Query-by-Humming System. In *Proceedings of ISMIR* (p. 133-139).