A Subsequence Matching with Gaps-Range-Tolerances Framework: A Query-By-Humming Application

Alexios Kotsifakos #1, Panagiotis Papapetrou *2, Jaakko Hollmén *3, and Dimitrios Gunopulos #4

Department of Informatics and Telecommunications, University of Athens, Greece
[1]ak@di.uoa.gr, [4]dg@di.uoa.gr
Department of Information and Computer Science, Aalto University, Finland
[2]panagiotis.papapetrou@aalto.fi [3]jaakko.hollmen@aalto.fi

ABSTRACT

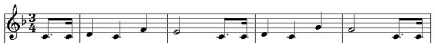
We propose a novel subsequence matching framework that allows for gaps in both the query and target sequences, variable matching tolerance levels efficiently tuned for each query and target sequence, and also constrains the maximum match length. Using this framework, a space and time efficient dynamic programming method is developed: given a short query sequence and a large database, our method identifies the subsequence of the database that best matches the query, and further bounds the number of consecutive gaps in both sequences. In addition, it allows the user to constrain the minimum number of matching elements between a query and a database sequence. We show that the proposed method is highly applicable to music retrieval. Music pieces are represented by 2-dimensional time series, where each dimension holds information about the pitch and duration of each note, respectively. At runtime, the query song is transformed to the same 2-dimensional representation. We present an extensive experimental evaluation using synthetic and hummed queries on a large music database. Our method outperforms, in terms of accuracy, several DP-based subsequence matching methods—with the same time complexity—and a probabilistic model-based method.

1. INTRODUCTION

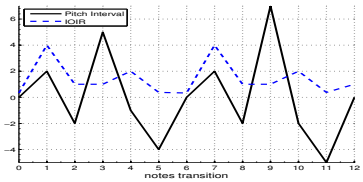
Finding the best matching subsequence to a query has been attracting the attention of both database and data mining communities for the last few decades. The problem of *subsequence matching* is defined as follows: given a query sequence and a database of sequences, identify the subsequence in the database that best matches the query. Achieving efficient subsequence matching is an important problem in domains where the target sequences are much longer than the queries, and where the best subsequence match for a query can start and end at any position in the database.

A large number of Dynamic Programming (DP) [1] based distance or similarity measures perform similarity search in several application domains including time series, categorical sequences, multimedia data, etc. Nonetheless, there are still many application domains, such as music retrieval, where these methods are not directly applicable or have very poor retrieval accuracy (see Section 5), since in many cases several properties and characteristics of the specific domain are ignored. In this paper, we focus on time series subsequence matching and approach the problem from the music retrieval perspective: suppose you hear a song but you cannot recall its name; one solution is to hum a short part of the song and perform a search on a large music repository to find the song you are looking for or even songs with similar melody. The main task of a *Query-By-Humming* (QBH) system is, given a hummed query song, to search a music database for the K most similar songs. This directly maps to subsequence matching as the hummed query is typically a very small part of the target sequence.

Let us now see how time series subsequence matching can be applied to QBH. Every piece of music is a sequence of notes characterized by a *key*, that defines the standard pattern of allowed intervals that the sequence of notes should conform with, and a *tempo*, that regulates the speed of the music piece. Each *note* consists of two parts: the *pitch* and the *duration*. A *pitch interval* is the distance between two pitches. In western music the smallest pitch interval is called *semitone*, a *tone* comprises two consecutive semitones, and the interval of 12 semitones is called *octave*. Another important term is *transposition*, i.e., the action of shifting a melody of a piece written in a specific key to another key. Finally, there is a discrimination between *monophonic* and *polyphonic* music; in the latter case it is possible for two or more notes to sound simultaneously, as opposed to the former case. Here, we consider monophonic music, as in QBH we deal with melodies hummed by users.



(a) Part of the music score.



(b) Representation using pitch intervals and IOIR.

Figure 1: Example of the music score and its 2-dimensional time series representation. IOIR is the duration ratio of two consecutive notes.

Pitch and duration are two distinctive factors for a music piece and they should both be used for efficient music representation [29]. We could have two or more songs that share similar note frequencies (i.e., pitch values) but their melodies vary due different individual pitch durations. Hence, if, for instance, only pitch is used to represent a music song, there is a high risk of erroneously matching two songs. Several existing approaches are hampered by the fact that they only consider pitch in their representation, ignoring

Proceedings of the VLDB Endowment, Vol. 4, No. 11

note durations [21]. In this work, we take into account both pitch and duration. Melodies are defined as 2-dimensional time series of notes of arbitrary length, where one dimension represents pitch and the other duration (see Figure 1 for an example).

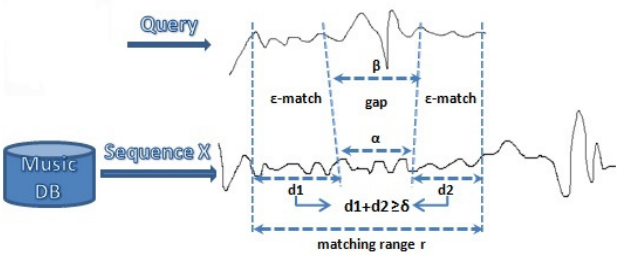


Figure 2: SMBGT: error-tolerant matching is denoted as ϵ-match.

In order to guarantee robust and meaningful subsequence matching for a potentially very noisy domain, like QBH, several parameters should be considered. First, when humming a song, slight or more serious errors may occur due to instant or temporary key or tempo loss, respectively. Thus, the matching method should be *error-tolerant*, otherwise there may be false negatives during the retrieval. In addition, the method should allow skipping a number of elements in both query and target sequences. This may, however, produce very long matching subsequences with large gaps between the matching elements. To solve this problem, we should constrain the length of the matching subsequence, e.g., to be at most r, where r is tuned appropriately for the application domain. Also, to ensure that the matching subsequences will include as many matching elements as possible, an additional constraint should be imposed to the minimum number of matching elements. However, constraining the number of matching elements may decrease the number of candidate matches. Specifically, if we have prior knowledge about the singing skills of the person who produced the hummed queries, we can tighten or loosen this constraint for strong or weak hummers, respectively. Moreover, we can bound the number of allowed consecutive gaps in both query and target sequences, which provides a setting that also controls the expansion of the matched subsequences during the DP computation.

Our first contribution is a subsequence matching framework that allows for a constrained number of **gaps** on both query and target sequences, **variable tolerance levels** in the matching (tolerances are functions of the query and target elements), and a matching **range** that constrains the maximum match length. In addition, a similarity measure, **SMBGT**, is proposed, which given a query Q and a target sequence X (with $|Q| << |X|$) finds the subsequence of X that best matches Q. Gaps are allowed in the alignment and can be bounded on both query and target sequences (by β and α respectively). Moreover, the maximum match length in X as well as the minimum number of matching elements are constrained (by r and δ respectively). An example of **SMBGT** is shown in Figure 2.

Our second contribution is an extensive **comparative evaluation** on QBH of several DP-based methods and a probabilistic model-based method, where their retrieval accuracy and runtime is studied on real (hummed by humans) and synthetic queries. Similar experimentation is performed for the proposed methods showing their superiority in terms of accuracy against these DP and model-based methods for several parameter settings.

2. RELATED WORK

Several DP methods exist for *whole sequence matching* including Dynamic Time Warping (DTW) [15] and variants (e.g., cDTW [26], EDR [4], ERP [5]) that are robust to misalignments and time warps, and some (e.g., DTW, cDTW) can achieve high retrieval accuracy in applications such as time series mining and classification [14]. Other DP-based methods allow for gaps in the alignment, e.g., LCSS [19]. All these methods, however, are designed for whole sequence matching. SPRING [27] is a DP-based method that finds the subsequences of evolving numerical streams that are closest to a query in constant space, and time linear to the database size (for short queries). Some of these DP methods can be applied to QBH. Similar DP-based methods exist for string matching [17, 28], though they are not in the scope of this paper.

Zhu et al. [33] developed an efficient lower-bound for DTW. This method, however, is designed for whole sequence matching and is not directly applicable to subsequence matching, which is our focus. The same problem remains in several other DP-based methods [21]. Such methods could be retrofitted by performing a sliding window search over the database. Nonetheless, such approach would be computationally expensive requiring one DP computation per window. Two methods for subsequence matching [11, 13] implicitly account for local adjustments of tempo, though they are not transposition invariant. SPRING seems promising for QBH and thus it is further studied in this paper. Moreover, the Edit distance [17] has been used for music retrieval with several variations [16]. Here, we study its most recent version [31]. All aforementioned DP methods, however, fail to handle noise imposed by users. Several LCSS-based approaches [2, 3, 29] can tolerate humming noise, though the fact that no bounds are imposed to the allowed gaps may result in a large number of false positives when $|Q| << |X|$. The approach of Han et al. [10] is based on uniform segmentation and sliding windows, which requires the user to manually select the length of the segments and is therefore not sensitive to the actual behavior of the data and can efficiently handle only *near exact* matching. Another approach has been proposed [7, 12] that deals with *whole query matching* and a bounded number of gaps only in the target sequence. Besides, none of these approaches accounts for note duration, i.e., they are proposed for 1-dimensional sequences. Moreover, some approaches embed transposition invariance as a cost function in the DP computation [6, 16], though, with not attractive runtime.

Furthermore, n-gram-based methods for music retrieval [8, 29] fail to handle noisy queries efficiently as they are designed for near exact matching. Also, several probabilistic methods (HMM-based) have been developed for speech recognition and music retrieval [20, 23, 25, 31]. However, they are computationally expensive due to the required training, and creating models to represent all styles of music in a large database is a very tough task.

Regarding the representation schemes developed for music retrieval, a common trend is to represent notes by encoding only their pitch [21]. The combination of pitch and duration though improves music retrieval as it holds more information for each note. Finally, in this paper, we study monophonic music rather than polyphonic [30, 32], as it directly applies to QBH.

3. PROBLEM SETTING

3.1 Representing Musical Pieces

There are two common ways of expressing pitch: (a) *absolute pitch*, where the frequency of the note is used—in MIDI this value is an integer between 1 and 127 with 0 representing *pause*—and (b) *pitch interval*, which is the frequency difference between two adjacent notes. Three common ways to encode duration [22] are: (a) *Inter-Onset-Interval* (IOI) defined as the difference in time onsets of two adjacent notes, (b) *IOI Ratio* (IOIR), defined as the ratio of IOIs of two adjacent notes, with the IOIR of the last note equal to 1, and (c) *Log IOI Ratio* (LogIOIR), being the logarithm of IOIR.

For the 2-dimensional time series representation, we considered two combinations of encoding schemes: ⟨pitch interval, IOIR⟩ and ⟨pitch interval, LogIOIR⟩. With these combinations we deal with note transitions, saving much computational time as we do not have to check for possible transpositions of a melody, nor do we have to scale in time when compared to other melodies. An example of the ⟨pitch interval, IOIR⟩ representation of the song of Figure 1(a) is shown in Figure 1(b). Also, pitch intervals were quantized in [-11,11] by applying modulo 12 [29]. This quantization corresponds to two octaves, a reasonable range in which human pitch can fluctuate while singing. Regarding time, LogIOIR was quantized to the closest integer or the closest value in $[-2, 2]$ [22].

3.2 Definitions

Let $X = \{x_1, \ldots, x_n\}$ be a time series that represents a music piece, where $|X|$ denotes the size of X. Each $x_j = \langle x_j^p, x_j^r \rangle \in X$ is a pair of real values, where x_j^p and x_j^r correspond to pitch and duration information respectively, and are represented using any of the schemes described in Section 3.1. A music database is a set of time series $DB = \{X_1, \ldots, X_N\}$, where N is the number of music pieces in DB. A *subsequence* of X, denoted as $X[ts : te] = \{x_{ts}, \ldots, x_{te}\}$, is a set of elements from X appearing in the same order as in X. The first element of the subsequence is x_{ts} and the last is x_{te}. Note that $X[ts : te]$ is not necessarily continuous, i.e., gaps are allowed to occur by skipping elements of X. Let $Q = \{q_1, \ldots, q_m\}$ be another time series with the same representation as X. Consider the following definitions:

DEFINITION 1. ***(Variable error-tolerant match)*** *We say that $q_i \in Q$ and $x_j \in X$* match with variable ϵ-tolerance*, and denote it as $q_i \approx_\epsilon^f x_j$, if, we use absolute or relative tolerance, and for a set of constraints $\epsilon^f = \{\epsilon_p^f, \epsilon_r^f\}$:*

$$\epsilon_p^f(i) = f_p(q_i^p) \text{ and } \epsilon_r^f(i,j) = f_r(q_i^r, x_j^r), \quad (1)$$

where f_p is a function of q_i^p and f_r a set of constraints on q_i^r, x_j^r.

Note that in Definition 1 $\epsilon_p^f, \epsilon_r^f$ can also be two constants. In most application domains sequences are numerical, thus the probability for outliers increases, which is the case for QBH where users are prone to instant humming errors [23]. Hence, it is imperative to allow for flexible error-tolerant matches. In Section 5.1.3 we discuss an instantiation of this definition for the QBH application.

DEFINITION 2. ***(Common bounded-gapped subsequence)*** *Consider two subsequences $Q[ts_1{:}te_1]$ and $X[ts_2{:}te_2]$ of equal length. Let $\mathcal{G}_Q$ and $\mathcal{G}_X$ denote the indices of those elements in Q and X, respectively, that are included in the corresponding subsequences. If $q_{\pi_i} \approx_\epsilon x_{\gamma_i}$, $\forall \pi_i \in \mathcal{G}_Q$, $\forall \gamma_i \in \mathcal{G}_X$, $i = 1, \ldots, |\mathcal{G}_Q|$, and*

$$\pi_{i+1} - \pi_i < \beta,\ \gamma_{i+1} - \gamma_i < \alpha, \quad (2)$$

then, pair $\{Q[ts_1{:}te_1], X[ts_2{:}te_2]\}$ defines a common bounded-gapped subsequence *of Q and X. The longest such subsequence satisfying $te_2 - ts_2 \leq r$ is called $SMBGT(Q,X)$.*

Example: Let $Q = \{6, 3, 10, 5, 3, 2, 9\}$, $X = \{1, 1, 3, 4, 6, 9, 2, 3, 1\}$. Consider subsequence Q[2:6] with $\mathcal{G}_Q = \{2, 4, 5, 6\}$, which corresponds to sequence $\{3, 5, 3, 2\}$, and X[3:8] with $\mathcal{G}_X = \{3, 4, 7, 8\}$, which corresponds to $\{3, 4, 2, 3\}$. Also, assume the following parameter setting: $\epsilon = 1$ (absolute tolerance), $\alpha = 2$, $\beta = 1$, and $r = 6$. Clearly, the two subsequences are of the same length, at most two ($\alpha = 2$) consecutive gaps occur in X—between the second and third elements in X[3:8]—and at most one ($\beta = 1$) consecutive gap occurs in Q—between the first and second elements in Q[2:6]. Range constraint $r = 6$ clearly holds for X[3:8], while all matching elements in the two subsequences differ by at most 1 ($\epsilon = 1$). Thus, pair $\{Q[2:6], X[3:8]\}$, is $SMBGT(Q,X)$.

3.3 Problem Formulation

Problem (Subsequence Matching): Given a database DB with N sequences of arbitrary lengths, a query sequence Q, and positive integers δ and r, find set $\mathcal{S} = \{X_i[ts : te] | X_i \in DB\}$ of the top-K subsequences with:

$$|SMBGT(Q, X_i[ts : te])| \geq \delta. \quad (3)$$

It should be mentioned that each database sequence contributes with only one subsequence $X_i[ts : te]$ to $\mathcal{S}$. Note the additional constraint $te - ts \leq r$ which is by definition included in SMBGT.

4. SUBSEQUENCE MATCHING WITH GAPS-RANGE-TOLERANCES

We present a novel subsequence matching framework that is motivated by QBH. One of the novelties of our framework is that it considers variable error-tolerant matches without employing a probabilistic model [11, 23] (Section 5.1.3). In addition, it allows gaps in both the query and target sequences during their alignment, constrains the maximum match range in the target sequence, and bounds the minimum number of matched elements. To the best of our knowledge, this is the first subsequence matching approach that considers all the above aspects and, as shown in the experiments, outperforms—in terms of accuracy—existing methods by over an order of magnitude on hummed queries.

4.1 SMBGT: Subsequence Matching with Bounded Gaps and Tolerances

To solve the problem presented in Section 3.3 we propose *SMBGT*, a novel method for subsequence matching. SMBGT bounds the number of consecutive gaps allowed in *both* X and Q by two positive integers α and β, respectively, allows for variable tolerance levels in the matching, constrains the matching range, and bounds the minimum number of matching elements. The intuition behind allowing gaps in both sequences is to deal with serious humming errors that are likely to occur due to temporary key/tempo loss or significant instant note loss (more than the acceptable tolerance). Thus, we should be able to skip these elements. We will refer to a special case of SMBGT where α and β are set to infinity as SMGT, i.e., no constraints are imposed on the length of the allowed gaps.

4.1.1 Computation

Consider an "alignment array" a of size $(|Q| + 1) * (|X| + 1)$, where Q, X are the compared sequences. $\forall i \in \{1, \ldots, |Q|\}$ and $\forall j \in \{1, \ldots, |X|\}$, the recursive computation for SMGT is:

$$a_{0,j} = 0 \text{ and } a_{i,0} = 0, \quad (4)$$

$$a_{i,j} = \begin{cases} a_{i-1,j-1} + 1 & \text{, if } q_i \approx_\epsilon^f x_j \\ \max\{a_{i-1,j}, a_{i,j-1}\} & \text{, otherwise.} \end{cases} \quad (5)$$

An additional matrix s keeps for each cell $a_{i,j}$ the start point of its best alignment, in $s_{i,j}$, and is updated according to the transitions.

For both SMGT and SMBGT, the computation of a is performed in an online fashion and the space complexity is $O(|Q|)$ as they do not need to store the whole matrix a. Instead, two 1-dimensional arrays are used, $prev$, cur, to track the scores ($prev.value$ and $cur.value$) and start points ($prev.start$ and $cur.start$) of two consecutive columns, $j - 1$, j, of a, since having the values of column $j - 1$ of a suffices to compute column j. The goal is to be

able to match *any* subsequence of Q with *any* subsequence in the database. According to the above recursion, whenever a match occurs the score on the alignment path is increased by 1, otherwise the maximum score of the two adjacent (left, top) cells is inherited with no extra transition penalty. In case of a tie, we choose the transition that corresponds to the subsequence with the most recent start point since this subsequence includes a smaller number of gaps. In SMBGT, the recursion for a is modified to include constraints α and β. Thus, when a mismatch occurs at position (i, j), $a_{i,j}$ stores the largest number of matched elements that can be propagated vertically or horizontally, while not violating α and β. This is checked by an additional step, called $propagation$ (Section 4.1.2).

Algorithm 1: Function $Update()$ for $SMGT$ and $SMBGT$.

Input: query Q, target X, column index j, array cur, and δ.
Output: current best match $best$.

begin
  // return the value and start point of the cell with the maximum value in cur.
  $\{l_{max}, l_{start}\} = max\{cur\}$;
  $l_{len} = j - l_{start} + 1$;
  $b_{len} = best_{end} - best_{start} + 1$;
  if $best == null \wedge l_{max} \geq \delta$ **then**
    $best_{value} = l_{max}$; $best_{start} = l_{start}$; $best_{end} = j$;
  end
  else if $l_{max} > best_{value} \vee (l_{max} == best_{value} \wedge b_{len} > l_{len})$ **then**
    $best_{value} = l_{max}$; $best_{start} = l_{start}$; $best_{end} = j$;
  end
end

Algorithm 2: Function $Reset()$ for $SMGT$.

Input: query Q, target X, column index j, array cur, tolerance ϵ^f, and match range r.
Output: updated column cur.

begin
  for $i \leftarrow 1$ **to** $|Q|$ **do**
    if $j - cur_i.start + 1 == r$ **then**
      if $q_i \approx_\epsilon^f x_j$ **then** $cur_i.value = 1$;
      else //determine the appropriate transition and return the value and start point.
        $\{cur_i.value, cur_i.start\} = check(cur, prev)$;
    end
  end
end

The maximum length of the database subsequence with the longest common bounded-gapped subsequence is constrained by r, and the minimum matching score by δ (Section 3.3). Notice that during the computation, $best = (best_{value}, best_{start}, best_{end})$ keeps track of the current best solution, with $best_{value}$ being the value of the best match, and $best_{start}$, $best_{end}$ the start and end points of that match, respectively. $best$ is updated as shown in Algorithm 1 taking into account δ. At the end of the computation $best$ corresponds to the location of SMBGT(Q, X) in X. Finally, given K, SMBGT reports the K database sequences, where the longest common bounded-gapped subsequences occur. To keep track of these subsequences, a priority queue S is maintained and updated accordingly using function $Update_{queue}()$. When a new candidate subsequence is found, S is updated if it contains less than K elements or if the new candidate match has a higher score than any of the K subsequences in S. The main steps of SMBGT are shown in Algorithm 3.

4.1.2 Propagation

Two additional arrays, A_{start} and B_{start}, are used to determine the direction of the propagation (left or top). Thus, for each cell (i, j), A_{start} and B_{start} store the latest match positions in X and Q, respectively. Two versions of these arrays (A^{prev}_{start}, A^{cur}_{start} and B^{prev}_{start}, B^{cur}_{start}) are used corresponding to $prev$ and cur, respectively. Suppose that the value for cell (i, j) (i.e., cur_i) is being computed and $propagation()$ is triggered due to a mismatch between q_i and x_j. This function will check whether the value of an adjacent (left or top) cell can be inherited. If $j - A^{prev}_{start}(i) \leq \alpha$ then left propagation is allowed. Similarly, if $i - B^{cur}_{start}(i-1) \leq \beta$, top propagation is allowed. We always choose the propagation that inherits the highest value in matrix a. In case both $prev_i$ (i.e., $a_{i,j-1}$) and cur_{i-1} (i.e., $a_{i-1,j}$) can be propagated, we choose $\max\{prev_i.value, cur_{i-1}.value\}$. If no propagation is possible, $cur_i.value = 0$ and $cur_i.start = 0$, so that another match can start at this point. In case of a tie, we choose the transition that leads to the subsequence with the most recent start point as this subsequence includes a smaller number of gaps.

Algorithm 3: $SMBGT$.

Input: query Q, target X, gap constraints α and β, tolerance ϵ^f, match range r, and parameter K.
Output: priority queue S.

begin
  $S = null$;
  for $t \leftarrow 1$ **to** $|DB|$ **do**
    $best_{value} = 0$; $best_{start} = 0$;
    for $j \leftarrow 1$ **to** $|X_t|$ **do**
      for $i \leftarrow 1$ **to** $|Q|$ **do**
        if $q_i \approx_\epsilon^f x_j$ **then**
          $cur_i.value = prev_{i-1}.value + 1$;
          $cur_i.start = prev_{i-1}.start$;
        end
        else
          $cur_i = propagation(i, j, A_{start}, B_{start})$;
        end
      end
      $best = Update(j, cur)$;
      $cur = Reset_B(j, cur, A_{start}, B_{start})$;
    end
    $Update_{queue}(S, best, K)$;
  end
end

4.1.3 Eliminating Large Matches

Regarding SMGT, due to constraint r it is necessary to perform an additional step (called $Reset()$) in order to avoid expanding matching subsequences whose length is r and thus are not going to be included in the final set of top-K matches. After computing cur and updating $best$, we scan cur to detect those cells that correspond to subsequence matches with length equal to r. This elimination is performed by function $check()$ which returns the new value and corresponding start point for each cell. Hence, for each cell (i, j), if $q_i \approx_\epsilon^f x_j$, then cur_i (that corresponds to that cell) is set to 1. Otherwise, it is checked whether inheriting the value of the left ($prev_{j-1}$) or top (cur_{i-1}) cell may lead to a violation of r. The value of the left cell can be inherited if the subsequence length that corresponds to that cell is less than $r - 1$. The intuition is that if the corresponding length is equal to $r-1$ this subsequence would have already been reported as a candidate match on cur and thus we should not expand it further. Moreover, the value of the top cell can never be equal to r, as the elimination is performed on cur from top to bottom. Thus, the value of the top cell can always be inherited. If both values can be inherited, we select the transition with the highest value and in case of a tie the transition that leads to the subsequence with the most recent start point. If no transition is possible, $check()$ returns 0 as the cell and start point value. The main steps of function $Reset()$ can be seen in Algorithm 2. In SM-

BGT the reset function that has to be triggered (called $Reset_B()$) is similar to $Reset()$ with an additional propagation check in case of a mismatch between q_i and x_j. If no propagation is possible the value and start point of the cell are set to 0, otherwise they are updated accordingly. In particular, in the case of a top propagation, the value of the top cell is inherited by the current cell. In the case of a left propagation, it should be ensured that it may not lead to a subsequence that violates r (the length of the subsequence that corresponds to the left cell is less than $r-1$). Finally, if both propagations are possible, we perform the one leading to the subsequence with the highest score and in case of a tie we perform the one that leads to the subsequence with the most recent start point.

4.2 Example of SMBGT

Consider the following example: let $Q = \{0, -4, 1, 2, -2\}$ and $X = \{0, 0, -4, 3, 0, 2, -3, 1\}$. We want to find SMBGT$(Q, X)$, with $\alpha = 2$, $\beta = 1$, $\delta = 3$, and $r = |Q| = 5$. For simplicity, we consider only the pitch dimension and do not impose any matching tolerance. We show four matrices: (1) a which is the alignment array used for the DP computation, (2) s which contains for each cell of a the start point of the best matching subsequence that ends on that cell, (3) A_{start}, and (4) B_{start}, which are the additional matrices used by function $propagation()$. Following Equation 4 and Algorithm 3 (Appendix), the first 6 columns of all four matrices are shown in Figure 3 (a). At this phase, column 6 contains cells that will trigger function $Reset_B()$. Consider row 5 of column 6. The start point of the subsequence that corresponds to that row is at position 2, which gives a subsequence of length $6-2+1 = 5 = r$. Thus, this cell should be reset. Since $q_i = x_j = 2$ (match), the new value of that cell should be 1. Let us check the next row of column 6. The length of the corresponding subsequence is now $6-2+1 = 5 = r$, however in this case $q_i \neq x_j$, thus we should check whether any propagation is possible. Regarding the left propagation, $j - A^{prev}_{start}(i) = 6-0 > \alpha$; hence left propagation is not allowed. Also, $i - B^{cur}_{start}(i-1) = 5-4 = 1 = \beta$, hence top propagation is allowed, and the new value of cell $(6, 6)$ is set to 1. Notice that s, A_{start}, and B_{start} are updated accordingly. The new matrices are now reset and are shown in Figure 3 (b). Clearly, $best_{value} = 3$, $best_{start} = 2$, and $best_{end} = 6$.
The time complexity of both SMGT and SMBGT is $O(|Q||X|)$. Also, none of the two measures is metric (see Appendix A.1).

5. EXPERIMENTS

We performed an extensive comparative evaluation of several DP-based methods and HMMs with SMGT and SMBGT on QBH by studying their accuracy and runtime on hummed and synthetic queries, showing the superiority of the proposed methods.

5.1 Experimental Setup

5.1.1 Data

We created a music database of 5643 freely available on the web MIDI files that cover various music genres. We also generated six synthetic query sets (100 queries per set) of lengths between 13 and 137: Q_0, $Q_{.10}$, $Q_{.20}$, $Q_{.30}$, $Q_{.40}$, and $Q_{.50}$. Q_0 contained exact segments of the database, while $Q_{.10} - Q_{.50}$ were generated by adding noise to each query in Q_0. For all queries we randomly modified 10, 20, 30, 40, and 50% of their corresponding time series in both dimensions. Noise was added to existing query elements without insertions or deletions. Moreover, in all noisy query sets we allowed at most 3 consecutive elements to be replaced with noisy values. Also, we used a set of 100 hummed queries of lengths between 14 and 76. More details can be found in Appendix A.4.

5.1.2 Evaluation

We studied five DP-based methods that can be applied to music retrieval: SPRING [27], Edit distance-based [31], two DTW-based [11, 13] (denoted DTW_s, DTW_c), and a gapped-based approach [12] (denoted Il. et al.). The first four are designed for subsequence matching whereas the fifth performs *whole query* matching. In our experiments, Edit has been slightly modified to deal with LogIOIR and quantizations, while for Il. et al. a more elastic version has been used, suitable for subsequence matching and supports both constant and variable tolerance. Both Il. et al. and SPRING were modified to allow for varying r. These methods are sketched in Appendix A.2. Moreover, since probabilistic methods have been applied to music retrieval, for completeness, we also compare the DP methods with an HMM-based approach where each database sequence is modeled by an HMM (see Appendix A.3 for details). Our evaluation strategy was organized as follows: we first tested the

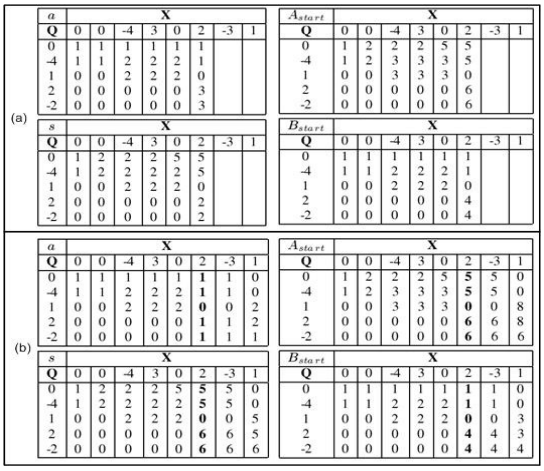


Figure 3: DP matrices for SMBGT (a) before and (b) after reset.

robustness of all methods with respect to noise using the synthetic query sets; those methods that achieved a reasonably high recall ($> 90\%$) even for high noise levels (50%) were further tested on hummed queries where the noise level can be much higher. Specifically, we evaluated the performance of Il. et al., SPRING, Edit distance, DTW$_s$, DTW$_c$, SMGT, and SMBGT on synthetic and real queries, in terms of *recall*, *mean reciprocal rank (MRR)* [9], and *average rank (AR)*. The top-K answers were returned. Recall is the percentage of queries for which the correct answer is among the top-K results. MRR is the mean inverse rank of all queries in their top-K results. If the right answer is not included in the results, then the inverse rank is 0. The rank of a query is the number of matches (i.e., database sequences) with similarity/distance value at least as high/low as that of the correct match (including the correct match). The average rank of a query set is the average rank of all queries in the set. All three measures are essential for the evaluation of a QBH method, as the first one (recall) shows how successful the method is in finding the correct answer among the top-K, whereas MRR and average rank indicate if there is room for improvement in terms of recall when decreasing K. For all methods we tried all variations and parameter settings; in the experiments we report those variations that achieved the best performance. Experiments were run on an AMD Opteron 8220 SE processor at 2.8GHz, and implemented in C++.

5.1.3 Variable Tolerances - Instantiation

For QBH, a reasonable definition for ϵ^f_p is the following:

$$\epsilon^f_p(i) = \lceil q^p_i * t \rceil, \text{ with } t = 0.2,\ 0.25,\ 0.5. \quad (6)$$

For ϵ_p^f, both absolute and relative tolerances were studied. Relative tolerance has been extended as follows:

$$x_p^r/(1+\epsilon_p^f) \leq q_i^p \leq x_p^r * (1+\epsilon_p^f), q_i^p, x_p^r \geq 0, \quad (7)$$

$$x_p^r/(1+\epsilon_p^f) \geq q_i^p \geq x_p^r * (1+\epsilon_p^f), q_i^p, x_p^r < 0. \quad (8)$$

This was necessary for this application, since by definition we should distinguish between positive and negative values of pitch intervals.

To define an appropriate form for $\epsilon_r^f(i,j)$ we should differentiate between IOIR and LogIOIR. After having people hum several pieces of different kinds of music, we observed a tendency of making duration ratios smaller, even half of their actual values. This is reasonable, as users care more about singing melodies than being exact in tempo. Also, we should account for cases of queries at slower tempos. Thus, for IOIR, we define:

$$\epsilon_r^f(i,j) = \{x_j^r \leq 2 * q_i^r,\ x_j^r - q_i^r \geq -0.5\}. \quad (9)$$

Negative values may occur in the case of LogIOIR, thus:

$$\epsilon_r^f(i,j) = \begin{cases} \{0 \leq log_2(x_j^r/q_i^r) \leq 1\}, log_2 x_j^r \geq 0. \\ \{|log_2(x_j^r/q_i^r)| \leq 1\}, log_2 x_j^r < 0. \end{cases} \quad (10)$$

As the logarithmic values get smaller, the difference in ratios gets smaller as well. Notice that the two forms of $\epsilon_r^f(i,j)$ shown in Equation 9 and Equation 10 are appropriate for the QBH application studied in this paper.

Nonetheless, the proposed variable tolerance framework is generic and can be used for other application domains, even with more dimensions, after defining an appropriate form for ϵ^f.

5.2 Experimental Results on Synthetic Queries

5.2.1 Parameters

For the methods that consider r in their computation (SPRING, Il et al., SMGT, and SMBGT) we set $r = |Q|$, as due to the query sets' construction, the desirable match will not exceed that value. Taking into account the noise levels of the query sets, in SMBGT and SMGT δ was set to 0.9, 0.7, 0.6, 0.5, 0.35, and 0.3 times the length of each query in the six query sets, $Q_0 - Q_{.50}$, respectively. We selected these gradually decreasing δ values, so as to be elastic enough as noise increases, and avoid false dismissals. We experimented with $\alpha = \beta = 3$, as we know that the maximum number of gaps in all query sets is 3, thus allowing SMBGT to capture these gaps. Also, we tried both tolerance schemes, i.e., constant and variable (Equation 6). For all synthetic query sets SMBGT achieved its best accuracy for variable absolute tolerance with $t = 0.2$, SMGT for variable absolute tolerance with $t = 0.5$, and Il. et al. for constant relative tolerance with $\epsilon_p = 1$ and $\epsilon_r = 4$. Regarding the HMM method, we conducted extensive experimental evaluation testing the effect of all of its parameters. The number of states M varied from 1 to 10, and the best accuracy was achieved for $M = 5$; training for $M > 10$ were prohibitive in terms of training runtime and memory consumption.

The observation distribution of the states we experimented with was Gaussian (which is common). Finally, the pseudocounts tested for the unobserved data were 0.001, 0.01, and 0.1 to 5 with step 0.2. The best accuracy was achieved for 0.1. For more details see Appendix A.5.3.

We tested several values for K, ranging from 5 to 150, for all synthetic query sets. In QBH, however, high values of K may not be practical as users may not be willing to look at a large number of candidate songs to identify the targeted one; we chose a reasonable value ($K = 20$) to report the accuracy, regardless of noise level.

5.2.2 Representation

We observed that, as the noise level increases, the representations achieving the highest recall per DP-based method are a subset of the representations of lower noise levels. The best representations for synthetic queries were ⟨mod12, IOIR⟩ and ⟨pitch interval, IOIR⟩ while the latter is also the representation leading to highest accuracies for hummed queries (Section 5.3). This shows that the simpler the representations, the more promising they seem to be in QBH. The representation used for the HMM method—leading to the smallest possible alphabet size—was ⟨mod12, LogIOIR in $[-2, 2]$⟩. For more details see Appendix A.5.3.

5.2.3 Accuracy

Regarding the 100 exact queries (Q_0), all DP-based methods achieved 100% recall, with MRR and average rank 1 for top-5, except for DTW$_s$, which did not exceed a recall of 96% even for $K = 250$. The HMM method, achieved 96% recall for top-5, with MRR 0.95 and average rank 1.22, and recall 100% for top-150. SMBGT, SMGT, and Edit distance performed best, and behaved similarly for all query sets. Even for $Q_{.50}$ their recall is 97, 96, and 97%, respectively. Figure 4(left) shows an overview of these findings. For more detailed results see Tables 1, 2, 3, 4, and 5 in Appendix A.5.1. The reason for the high recall of Edit distance is that if two elements do not match, it will increase the total matching score by at most 1, while for the remaining intact elements this score will not be affected. On the contrary, the recall of all other competitor methods degrades with noise. SPRING and the HMM method behave similarly for $Q_{.10}$, $Q_{.20}$, and $Q_{.30}$ presenting a recall of more than 91%, but further increasing the noise level results in a smooth degradation for SPRING and a sharp one for the HMM method. For $Q_{.50}$ their recall is only 75 and 56%, respectively. The accuracy of DTW$_c$ and Il. et al. degrades very sharply when adding noise, achieving for $Q_{.40}$ 27 and 53% recall, and for $Q_{.50}$ 7 and 32%, respectively. The latter method presents such behaviour, as it will sooner stop its computation when not being able to find a match for a query element. SMBGT significantly outperforms Il. et al. with $\alpha = \beta$ due to its additional ability to skip query elements. DTW$_s$ performs worst for all noise levels (0% recall for $Q_{.50}$), and this behaviour, along with that of DTW$_c$, is justified by the fact that they implicitly embed time by allowing it to adjust locally, and they are unable to skip non-matching elements, as they force them to align. For both of them, the value of parameter c (Equation 17) achieving their best recall was 2. SPRING explicitly accounts for duration information in its computation, thus it can tolerate higher noise levels as opposed to DTW$_s$ and DTW$_c$. Referring to MRR and average rank, the same conclusions hold, with SMBGT, SMGT and Edit distance remaining close to 1 for all noise levels, which is expected as the other competitors identify fewer correct answers in the top-K.

5.2.4 Time

We observed that the average execution time per query length for all methods increased linearly to the query length. DTW$_s$ was the fastest of all DP-based methods, however, achieving the worst accuracy. DTW$_c$, SMBGT, SMGT, and Edit distance showed negligible differences to each other, while SPRING and DTW$_s$ were faster than the aforementioned methods, since they have a simpler computation scheme compared to the other methods. Il. et al. was the most computationally expensive method, while the HMM was the fastest one. Nevertheless, the training time for the HMMs was close to 14 hours due to their high complexity, which is prohibitive, not to mention the significant degradation in accuracy for high noise levels. For more details refer to Figure 6.

5.3 Experimental Results on Hummed Queries

The methods that showed to be noise-tolerant even for high noise levels of 50%, i.e., SMBGT, SMGT, and Edit distance, were further evaluated for the hummed query set, since possibly none of the elements of the correct song will be intact, due to humming errors and noise introduced by the recording procedure (Appendix A.4.3).

5.3.1 Tolerance

In Figure 4(middle and right) (and also Table 7 in Appendix A.5.1), we see the results for absolute tolerance for SMBGT and SMGT, with $t = 0.2$ and $t = 0.25$ (Equation 6), respectively, which achieved the best accuracy. For any tolerance scheme, small constant and variable t made our methods perform better than for greater values, with variable t being better. Moreover, absolute tolerance always outperformed relative tolerance in all experiments. For example, for $t = 0.2$ and $K = 10$, relative tolerance was 33% worse than absolute. Regarding Edit distance, no tolerance can be defined.

5.3.2 Tuning Parameters

We experimented with parameters r, α, and β, for $\delta = 0$. First, we studied the effect of r in SMGT where no constraint is imposed on the number of consecutive gaps. Increasing r, starting from $r = |Q|$, we observe that the recall of SMGT increases, until $r = 1.2 * |Q|$, after which the recall degrades. Interestingly, setting $r = \infty$ leads to a recall of 0%. This is not surprising since increasing r without any additional constraint in the number of gaps results in a larger number of erroneous candidate matches (see Table 6 for detailed results). We also studied the influence of α and β for the extreme case of $r = \infty$ for SMBGT. Testing all pairs of values in $[2, 8]$, the recall was significantly improved and the best recall was achieved for $\alpha = \beta = 4$ (51% for $K = 50$), as shown in Figure 4(middle). Then, in order to capture the impact of r combined with α and β, we gradually decreased the value of r and tested all pairs of $\alpha, \beta \in [2, 8]$. The best accuracy was achieved for $\alpha = 5$, $\beta = 6$, and $r = 1.2 * |Q|$, verifying the need for skipping elements in both target and query. In Figure 4(middle) we also show how accuracy is improved when varying r from 1.2 to 2, for $\alpha = 5$ and $\beta = 6$.

5.3.3 Accuracy

Figure 4(right) shows that our methods are at least 30 times higher in recall than Edit distance for $K = 50$, while SMBGT achieves 10% higher recall than SMGT for $K = 5$, and 15% for $K = 10$ and $K = 20$. The recall of Edit distance is 0% even for $K = 10$. The values reported in Figure 4(right) are achieved for $r = 1.2$, $\alpha = 5$ and $\beta = 6$ (for SMBGT), and $\delta = 0$ for all K. Regarding MRR, SMBGT outperforms Edit distance by more than two orders of magnitude for $K = 50$, while SMGT achieves worse MRR than SMBGT, as these measures are influenced by the recall (see also Table 7). The values of MRR for $K < 50$ are very close to those of $K = 50$, and hence we do not report them. Increasing $K > 50$ may improve the recall of all methods, though, trading K for higher recall will increase retrieval cost as more database sequences will be reported. Our goal, in QBH, is to achieve high recall by reporting as few candidates as possible. Hence, for $K = 5$-50 the proposed methods clearly offer higher recall. Finally, we tested the impact of δ on recall for the best combination of parameters for SMBGT. Increasing δ, even for $\delta = 0.5 * |Q|$ and $K = 50$ the recall does not decrease, while further increasing it makes recall worse. For example, for $\delta = 0.6 * |Q|$ it degrades to 28%, and for $\delta = 0.7 * |Q|$ to 9%, when $K = 5$. This behavior indicates that the recall on the hummed queries for which the correct song appeared in the top-K was not influenced by requesting more elements of the targeted sequences to match to theirs. In other words, these hummed queries were very similar to the targeted songs, leading us to the conclusion that if, in QBH, the users are singing well (both in pitch and time), δ can be set, for example, to 0.5, resulting in fewer false positive candidates.

5.4 Lessons Learned

Concluding our experimentation we learned the following: (1) due to their inherent lack of flexibility most of the existing subsequence matching methods cannot handle high noise levels as shown in the synthetic experiments, hence they completely fail in hummed queries as opposed to SMBGT and SMGT. Among all competitor methods Edit showed to be the most promising, however, for hummed queries it was at least 30 times lower in recall, for $K = 50$, than SMGT and SMBGT. For smaller K, e.g., $K = 10$, SMBGT achieved up to 15% higher recall than SMGT, which shows that imposing bounds α and β is indeed useful, (2) variable ϵ-tolerance always achieved better accuracy than constant ϵ-tolerance, (3) α and β were tuned by trying all possible combinations in a small range reasonable to QBH and reporting the one with the best accuracy. In applications where this range is prohibitively large, cross validation could be used, (4) δ may be tuned according to the singing skills of the hummers and r should not exceed the query length by more than a factor of 1.2, (5) simple representations favor DP-based methods, (6) all DP-based methods have similar retrieval time complexity (with small variations) while HMMs are faster, though the required training is computationally expensive.

6. CONCLUSIONS

Motivated by QBH we proposed a subsequence matching framework, which allows for gaps in both query and target sequences, variable tolerance levels in the matching of elements, and constrains the maximum match length and the minimum number of matched elements. Our framework was shown to outperform several DP-based subsequence matching methods and a model-based probabilistic method in terms of accuracy, after extensive experimental evaluation on a large music database using hummed and synthetic queries. Directions for future work include testing the performance of the proposed methods in other application domains.

Acknowledgements. This work was partially supported by the ALGODAN Centre of Excellence and the SemSorGrid4Env and MODAP European Commission projects.

7. REFERENCES

[1] R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 1954.

[2] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *SPIRE*, pages 39–48, 2000.

[3] B. Bollobás, G. Das, D. Gunopulos, and H. Mannila. Time-series similarity problems and well-separated geometric sets. In *Symposium on Computational Geometry*, pages 454–456, 1997.

[4] L. Chen and R. Ng. On the marriage of l_p-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[5] L. Chen and M. T. zsu. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[6] T. Crawford, C. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:73–100, 1998.

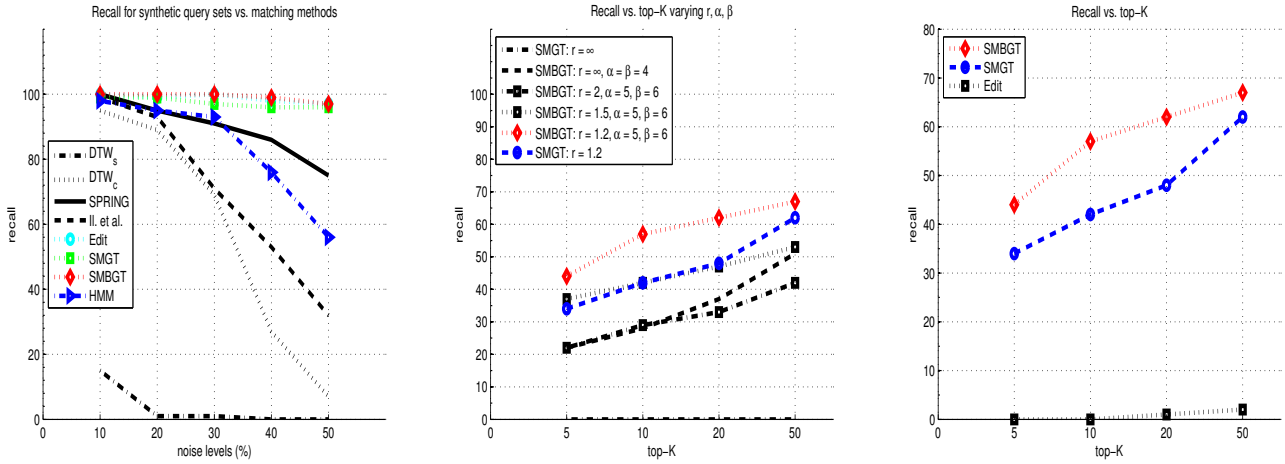


Figure 4: Recall of the proposed methods for: (left) synthetic queries vs. DP and HMM methods for $K = 20$; (middle) hummed queries varying $\alpha, \beta, r,$ and K; (right) hummed queries vs. Edit varying K.

[7] M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis, and K. Tsichlas. Approximate string matching with gaps. *Nordic Journal of Computing*, 9(1):54–65, 2002.

[8] R. Dannenberg, W. Birmingham, B. Pardo, N. Hu, C. Meek, and G. Tzanetakis. A comparative evaluation of search techniques for query-by-humming using the MUSART testbed. *Journal of the American Society for Information Science and Technology*, 58(5):687–701, 2007.

[9] R. Dannenberg, W. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo. The Musart Testbed for Query-by-Humming Evaluation. *Computer Music Journal*, 28(2):34–48, 2004.

[10] T. Han, S.-K. Ko, and J. Kang. Efficient subsequence matching using the longest common subsequence with a dual match index. In *Machine Learning and Data Mining in Pattern Recognition*, pages 585–600. 2007.

[11] N. Hu, R. Dannenberg, and A. Lewis. A probabilistic model of melodic similarity. In *ICMC*, pages 509–515, 2002.

[12] C. Iliopoulos and M. Kurokawa. String matching with gaps for musical melodic recognition. In *PSC*, pages 55–64, 2002.

[13] J. Jang and M. Gao. A query-by-singing system based on dynamic programming. In *International Workshop on Intelligent Systems Resolutions*, pages 85–89, 2000.

[14] E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[15] J. B. Kruskall and M. Liberman. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps*. Addison-Wesley, 1983.

[16] K. Lemström and E. Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *AISB*, pages 53–60, 2000.

[17] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics*, 10(8):707–710, 1966.

[18] S. Levinson, L. Rabiner, and M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983.

[19] D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, 1978.

[20] C. Meek and W. Birmingham. A comprehensive trainable error model for sung music queries. *Journal of Artificial Intelligence Research*, 22(1):57–91, 2004.

[21] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990.

[22] B. Pardo and W. Birmingham. Encoding timing information for musical query matching. In *ISMIR*, pages 267–268, 2002.

[23] B. Pardo, J. Shifrin, and W. Birmingham. Name that tune: A pilot study in finding a melody from a sung query. *Journal of the American Society for Information Science and Technology*, 55(4):283–300, 2004.

[24] A. Pikrakis, S. Theodoridis, and D. Kamarotos. Classification of musical patterns using variable duration hidden Markov models. *Transactions on Audio, Speech, and Language Processing*, 14(5):1795–1807, 2006.

[25] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[26] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Transactions on ASSP*, 26:43–49, 1978.

[27] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. In *ICDE*, pages 1046–1055, 2007.

[28] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[29] A. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In *ACM Multimedia (Part 1)*, page 66, 1999.

[30] E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content-based music retrieval. In *ISMIR*, pages 193–199, 2003.

[31] E. Unal, E. Chew, P. Georgiou, and S. Narayanan. Challenging uncertainty in query by humming systems: a fingerprinting approach. *Transactions on Audio Speech and Language Processing*, 16(2):359–371, 2008.

[32] G. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *ISMIR*, pages 13–17, 2002.

[33] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD*, pages 181–192, 2003.

APPENDIX

A. APPENDIX

We provide some additional material and further details on our work. The content of this section is organized as follows: in Appendix A.1 we prove that SMBGT is non-metric (this holds for SMGT as well); in Appendix A.2 and A.3 we present the DP-based competitors and the probabilistic approach, respectively, that have been benchmarked and tested against our two proposed methods; in Appendix A.4 we provide additional description of the data used in our experimentation; finally, in Appendix A.5 we include some additional experimental results on accuracy, runtime, and representation.

A.1 SMBGT

THEOREM 1. *SMBGT is not metric.*

PROOF. We prove this theorem by showing that SMBGT does not follow the triangle inequality. Consider the following three 1-dimensional time series: $X_1 = \{3, 2\}, X_2 = \{4\}, X_3 = \{2, 2, 1\}$. Assume ϵ-tolerance fixed to 1, $r = \infty$, $\delta = 0$, and $\alpha = \beta = 1$. Then $SMBGT(X_1, X_2) = 1$, $SMBGT(X_2, X_3) = 0$, and $SMBGT(X_1, X_3) = 2$. However, $SMBGT(X_1, X_2) + SMBGT(X_2, X_3) < SMBGT(X_1, X_3)$. □

A.2 DP-based Methods

DP-based methods typically use an "alignment array" a of size $(|Q| + 1) * (|X| + 1)$, where Q, X are the compared sequences. For $1 \leq i \leq |Q|$ and $1 \leq j \leq |X|$, each cell $a_{i,j}$ represents either the minimum cost for aligning subsequences of Q and X ending at i and j, respectively, or the maximum number of their matched elements, depending on whether we use a distance or similarity measure.

A.2.1 Edit distance-based

The most recent variation of the Edit distance [31] between two sequences Q and X, with slight extensions to deal with LogIOIR and quantizations, is computed as follows:

$$a_{0,j} = 0 \text{ and } a_{i,0} = i, \quad (11)$$
$$a_{i,j} = \min\{a_{i-1,j} + 1, a_{i,j-1} + 1, a_{i-1,j-1} + w(q_i, x_j)\},$$

where $w(q_i, x_j)$ is defined as:

$$w(q_i, x_j) = \frac{1}{2} * \left\{ \left| \frac{q_{i_p} - x_{j_p}}{PitchRange} \right| \right\} + \frac{1}{2} * DurationCost \quad (12)$$

$$DurationCost = \begin{cases} |1 - \frac{\min\{q_{i_r}, x_{j_r}\}}{\max\{q_{i_r}, x_{j_r}\}}|, \text{ for IOIR.} \\ |\frac{q_{i_r} - x_{j_r}}{DurationRange}|, \text{ for LogIOIR.} \end{cases} \quad (13)$$

$PitchRange$ and $DurationRange$ correspond, respectively, to the maximum pitch interval and LogIOIR range in DB. After a is computed, this method reports $\min_j\{a_{|Q|,j}\}$, i.e., the minimum cost of aligning Q with the subsequence of X ending at position j.

A.2.2 SPRING

SPRING [27] uses an additional matrix s which keeps for each cell $a_{i,j}$ the start point of its current best alignment. The recursive computation of $a_{i,j}$ is:

$$a_{0,j} = 0 \text{ and } a_{i,0} = \infty, \quad (14)$$
$$a_{i,j} = w(q_i, x_j) + d_{best}, \quad (15)$$
$$d_{best} = \min\{a_{i-1,j}, a_{i,j-1}, a_{i-1,j-1}\}, \quad (16)$$

with $w(q_i, x_j)$ being the L_p norm of q_i and x_j. The same initialization is used for $s_{i,j}$ and at each iteration the start point of the element that was used to produce d_{best} is propagated. Finally, after a is computed, SPRING reports $\min_j\{a_{|Q|,j}\}$.

A.2.3 DTW tempo scaling

The next two methods were developed to measure the melodic similarity of two sequences Q and X without using tempo information directly, though allowing to locally adjust the tempo at certain positions. We refer to them as *simple* (DTW_s) [13] and complex (DTW_c) [11]. Both methods share the same initial condition, shown in Equation 17, where the simple scheme has been slightly modified to conform with the complex scheme. The recursions for the simple and complex scaling schemes are shown in Equation 18 and Equation 19, respectively.

$$a_{0,j} = 0 \text{ and } a_{i,0} = a_{i-1,0} + c, \quad (17)$$
$$a_{i,j} = w(q_{i_p}, x_{j_p}) + \min\{a_{i-1,j-1}, a_{i-1,j-2}, a_{i-2,j-1}\}, \quad (18)$$
$$a_{i,j} = \min\left\{ \begin{matrix} a_{i-1,j-1} \\ a_{i-2,j-1} + w(q_{i-1_p}, x_{j_p}) \\ a_{i-1,j-2} + w(q_{i_p}, x_{j-1_p}) \end{matrix} \right\} + w(q_{i_p}, x_{j_p}). \quad (19)$$

Note that c is a user-defined positive integer and $w(q_{i_p}, x_{j_p}) = |q_{i_p} - x_{j_p}|$. Finally, both schemes report $\min_j\{a_{|Q|,j}\}$.

A.2.4 Iliopoulos et al.

The method by Iliopoulos et al. [12] performs whole query matching, by demanding all points of Q to match within a constant ϵ-tolerance in a subsequence of X, and allows for a limited number of gaps only in the target sequence. A DP computation is performed, where every match is rewarded with a score of 1, whereas, whenever a mismatch occurs between q_i and x_j, it checks whether the best matching value found so far for q_i, can be propagated without exceeding α gaps in X.

In our experiments, we have developed a more elastic version of this method suitable for subsequence matching. First, a single database scan identifies all possible start points of the candidate matches. These start points correspond to the first query element according to the tolerance scheme. Then, for each candidate, the method performs the DP computation described above. In addition, the whole query matching requirement is eliminated. When a non-matched query element q_i is met, instead of ignoring the query subsequence $Q[1 : i]$ matched so far, we consider $Q[1 : i]$ as a candidate match. The rationale behind this is that it would be a too tight constraint to demand all query elements to match in X. In a realistic situation, as in QBH, humming errors might occur that, in the original setting, would immediately eliminate candidate matches with even a mismatch due to a single falsely hummed note.

A.3 Probabilistic Method

An HMM [18, 25] is a doubly stochastic process that contains a finite set of states. Each state emits/observes one symbol based on a probability distribution, and transitions between states are regulated by the so-called *transition probabilities*. More formally, an HMM is defined by: (1) M distinct states, (2) L distinct symbols that can be observed at each state, i.e., the discrete alphabet, (3) set $T = \{t_{ij}\}$ of transition probabilities, where $t_{ij} = P[s_t = j | s_{t-1} = i], 1 \leq i, j \leq M$, where s_t is the state at time t. This implies that the current state depends only on the predecessor one (first order Markov chain assumption), (4) set $E = \{e_j(k)\}$ of the probabilities of observation symbols at state j, where $e_j(k) = P[o_t = k | s_t = j]$, where o_t is the observed symbol at time t,

and (5) set $\Pi = \{\pi_j\}$ of prior probabilities, where $\pi_j = P[s_1 = j], 1 \leq j \leq M$.

When an HMM is trained from a set of sequences it reflects the probabilistic relations of symbols within the sequences. Given a database of sequences, if we had a probabilistic model for each individual sequence or group of homogeneous sequences, we could transform the query matching problem to a probability calculation of each model having generated a sequence (query Q). In other words, we would be looking for the model which maximizes the log-likelihood of the query sequence.

Taking advantage of the training phase of HMMs is not that trivial when the database contains a large number of heterogeneous sequences. In QBH the database may contain a large number of songs covering a wide range of music styles, as happens with our data. This would impose high heterogeneity in the database and there would be no implication about any kind of correlation between the sequences. Consequently, forming groups of similar sequences which can then be represented by HMMs (after training), or even combining HMMs by constructing mixture models, would lead to unexpected and meaningless results. If, in contrast, the database consists of homogeneous sequences (e.g., music patterns) then HMMs could be highly applicable for, e.g., retrieval or classification [24]. Thus, it is obvious that the most reasonable and fairest approach would be to model each database sequence with one HMM [23], which is in fact the approach we adopted in this paper. Notice that accounting for more sequences per HMM would achieve at most the performance of this approach, as in the former case each HMM would try to model the behavior of several uncorrelated database sequences, while in the latter the HMM is trained to model the structure of each database sequence.

A.4 Data

A.4.1 Database

Our database covers various music genres such as blues, rock, pop, classical, jazz, themes from movies and tv series. For each MIDI (comprising 16 channels) and channel, we extracted the highest pitch at every time click (*all-channels extraction* [29])[1]. Then, we converted tuples ⟨pitch, click⟩ to ⟨pitch interval, IOIR⟩, resulting in 40891 time series. This pre-processing procedure was done offline and only once, guaranteeing that there is no chance of missing a melody existing in any channel of a song.

A.4.2 Synthetic Queries

The pitch interval of the noisy elements was changed by $\pm k \in [3, 8]$ (integer), as we wanted the noise to range within one octave. This simulates the error performed by a human when singing a song by memory as well as the intrinsic noise that may be added by any audio processing tool. An erroneous interval of 3 to 8 semitones, i.e., 1.5 to 4 tones, is very reasonable.

Regarding the IOIR dimension, each q_i^r was modified by $\pm k \in [2, 4]$ (real), so that several reasonable variations of duration ratios could be simulated, and also be outside the bounds described by Equation 9, avoiding any bias against our proposed methods. In case of a negative value in IOIR, it is reset to a very small real positive value, as duration ratios should be positive. Moreover, in all noisy query sets we allowed at most 3 consecutive elements to be replaced with noisy values. This is because in QBH we do not expect to have many consecutive matching errors, or else it would be hard to identify the correct target for short queries.

[1]In the extraction process we excluded channel 10, since it is used for drums and cannot offer any musical information in QBH.

Table 1: Accuracy of all methods for query set $Q_{.10}$ and $K = 20$

	Accuracy			
Methods	Recall (%)	MRR	AR	Repr.
SMBGT	100	1	1	1-8
SMGT	99	0.99	1.2	1,2,4-6,8
DTW$_s$	15	0.1265	18.2	1,5
DTW$_c$	95	0.9198	2.23	1,5
Edit	100	1	1	1,2,6
SPRING	100	0.995	1.01	2,4,6,8
Il. et al.	99	0.9523	1.33	1,5
HMM	98	0.9054	1.72	3

Table 2: Accuracy of all methods for query set $Q_{.20}$ and $K = 20$

	Accuracy			
Methods	Recall (%)	MRR	AR	Repr.
SMBGT	100	1	1	1-8
SMGT	99	0.99	1.2	1,2,5,6
DTW$_s$	1	0.0005	20.98	5
DTW$_c$	89	0.8479	3.48	5
Edit	99	0.99	1.2	1
SPRING	95	0.9357	2.98	2,6,8
Il. et al.	93	0.8802	3.02	1,5
HMM	95	0.8953	2.14	3

Table 3: Accuracy of all methods for query set $Q_{.30}$ and $K = 20$

	Accuracy			
Methods	Recall (%)	MRR	AR	Repr.
SMBGT	100	1	1	1,2,5,6
SMGT	97	0.97	1.6	1,2,5,6
DTW$_s$	1	0.01	20.8	5
DTW$_c$	69	0.6304	7.49	5
Edit	100	0.9833	1.11	1
SPRING	91	0.8778	5.2	2,6,8
Il. et al.	71	0.5803	7.8	5
HMM	93	0.7974	3.09	3

Table 4: Accuracy of all methods for query set $Q_{.40}$ and $K = 20$

	Accuracy			
Methods	Recall (%)	MRR	AR	Repr.
SMBGT	99	0.9745	1.27	2,5,6
SMGT	96	0.9381	1.9	5
DTW$_s$	0	0	21	-
DTW$_c$	27	0.2514	15.89	5
Edit	98	0.9664	1.47	1
SPRING	86	0.7906	7.84	6,8
Il. et al.	53	0.3759	11.34	5
HMM	76	0.5538	7.57	3

Table 5: Accuracy of all methods for query set $Q_{.50}$ and $K = 20$

	Accuracy			
Methods	Recall (%)	MRR	AR	Repr.
SMBGT	97	0.9487	1.71	2,5,6
SMGT	96	0.9241	1.97	5
DTW$_s$	0	0	21	-
DTW$_c$	7	0.0427	19.92	5
Edit	97	0.9366	1.88	1
SPRING	75	0.638	14.51	6
Il. et al.	32	0.2072	15.44	5
HMM	56	0.369	11.53	3

A.4.3 Hummed Queries

To evaluate the methods in QBH, 4 males were asked to hum 25 songs (each). Two of them were musically trained with middle and low level studies in the piano and the guitar, while the third and fourth had no musical background. Melodies were hummed in a microphone and then converted to MIDI using the *Akoff music composer-version 2.0* [2], a well-known tool commonly used for

[2]http://www.akoff.com/music-composer.html.

evaluating QBH systems (e.g., by Zhu et al. [33]). All-channels extraction was applied to the queries to obtain the same representation with the database. The query set covered several genres of music, such as classical ("Für Elise"), blues ("Hideaway"), jazz ("Strangers in the Night"), rock 'n' roll ("Rock Around the Clock"), rock ("Fly Away"), country ("Hey Good Lookin"), and romantic songs ("What a Wonderful World").

Selecting the final set of hummed queries involved manual process. Apart from the mistakes that a user can make, any recording procedure may introduce noise in both pitch and duration. After listening to the MIDI of each hummed song, noise had been introduced, especially in pitch. Consequently, users had to hum each song several times before selecting the version with the least amount of noise, i.e., the one whose melody sounded as close to the target as possible. Furthermore, users were asked to avoid singing with lyrics and also to sing close to the microphone.

A.5 Experimental Results

A.5.1 Accuracy

The results of the performance evaluation of the proposed methods with respect to accuracy, for all synthetic query sets, are shown in Tables 1, 2, 3, 4, and 5.

Table 6: Recall of SMGT for various top-K and ranges for the hummed queries

	Recall (%)					
r	$\|Q\|$	$1.1\|Q\|$	$1.2\|Q\|$	$1.5\|Q\|$	$2\|Q\|$	∞
$K=5$	32	31	34	31	30	0
$K=10$	38	40	42	36	40	0
$K=20$	41	46	48	44	42	0
$K=50$	47	52	62	57	56	0

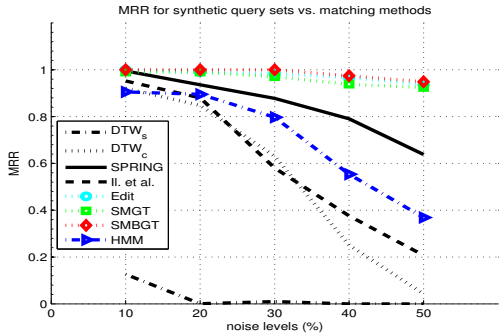


Figure 5: Accuracy of the proposed methods vs. DP and HMM methods for $K = 20$ in terms of MRR for synthetic queries.

Table 7: Recall and MRR of the proposed methods vs. Edit distance for the hummed queries ($K = 50$)

	Accuracy		
Methods	Recall (%)	MRR	Tol.
SMBGT	67	0.3661	abs. 0.2
SMGT	62	0.2956	abs. 0.25
Edit	2	0.0012	-

A.5.2 Runtime

In Figure 6 we show the average execution time for all methods per query length conforming with the complexities of the methods. For DP-based methods the time complexity is $O(|Q||X|)$ whereas for Il. et al. it is $O(|X| + |Q||X|^2)$. Training an HMM for a sequence X is $O(W|X|M^2)$, and computing the log-likelihood of a query Q being generated by an HMM is $O(|Q|M^2)$, where W is the number of iterations of the Baum-Welch [25] algorithm.

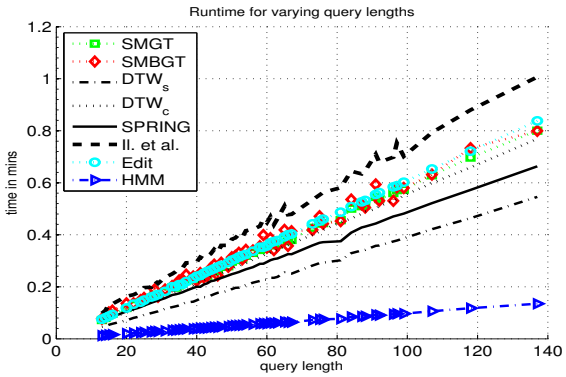


Figure 6: Average execution time for all methods per query length.

A.5.3 Representation

For synthetic queries we observe that, as the noise level increases, the representations (Table 8) achieving the highest recall per DP-based method are a subset of the representations of lower noise levels (Tables 1, 2, 3, 4, 5). An example can be seen for SPRING, where in $Q_{.10}$ the highest recall is achieved for 2, 4, 6, and 8, in $Q_{.20}$ and $Q_{.30}$ for 2, 6, 8, and in $Q_{.40}$ and $Q_{.50}$ for 6, 8 and 6, respectively. Moreover, representations 1 and 5 appear most in all synthetic query sets, and 5 is also the representation leading to highest accuracies for hummed queries. The latter shows that the simpler the representations, the more promising they seem to be in QBH. The representation used for the HMM method, which leads to the smallest possible alphabet size, is 3. Each pair of the cartesian product of the two dimensions pitch interval and ratio is encoded by one symbol, resulting in 115 discrete symbols. Selecting a small alphabet size implies that each state does not take into account too many symbols (so as to observe/emit one of them), and hence, it becomes more likely for a query to be generated by the targeted sequence, if there is not much noise in it. This is because the approach is probabilistic and every symbol is assigned a non-zero probability of being emitted at each state. Following a Gaussian distribution with mean $\mu = 58$ for representation 3 (Table 8), which is the mean of the symbols' distribution over the database, and varying the standard deviation σ, HMMs performed better than randomly selecting the probability values of emitting any symbol at each state.

Table 8: Code numbers for representations

Code number	Representation
1	$\langle$mod12, IOIR$\rangle$
2	$\langle$mod12, LogIOIR$\rangle$
3	$\langle$mod12, LogIOIR in $[-2, 2]\rangle$)
4	$\langle$mod12, LogIOIR quantized to closest integer$\rangle$
5	$\langle$pitch interval, IOIR$\rangle$
6	$\langle$pitch interval, LogIOIR$\rangle$
7	$\langle$pitch interval, LogIOIR in $[-2, 2]\rangle$)
8	$\langle$pitch interval, quantized to closest integer$\rangle$