Int. J. Mach. Learn. & Cyber. (2011) 2:147–156
DOI 10.1007/s13042-011-0025-0

ORIGINAL ARTICLE

Online music tracking with global alignment

Antonio Camarena-Ibarrola · Edgar Chávez

Received: 22 April 2011 / Accepted: 10 June 2011 / Published online: 1 July 2011

Abstract To make audio monitoring, the state of the art in this area makes use of local alignment algorithms between the objective audio and musical interpretation.The inductive hypothesis of a local alignment tool is that the alignment is correct to the current position of an error this is drag and accumulate to subsequent errors which do not recover unless elaborate heuristics are used. Our approach uses a local non-alignment scheme based on the audio search the entire purpose of short segments of audio taken from musical performance to get the k nearest audio segments (the proximity is determined using audio tracks based on entropy signs).The current audio segment of the play is paired with the nearest (in time) between the k previously selected audio segments of the target audio.To our knowledge, this is the first algorithm able to start up from an arbitrary point in the audio end, for example, if the musical performance had already begun when the monitoring system just went on.We complemented the overall strategy through a simple heuristic of ignoring the candidates when they are all too far in time with respect to the last position reported by the system.We have tested our method with 62 musical pieces, some of which are pop and classical music mostly.For every song we have two interpretations, we use one as the audio object and the other as the interpretation which will be monitored.We obtained excellent results.

Keywords Follow-up · Music · Entropy · Index · Near

1 Grounds

Real-time tracking of musical performances consists in setting the position on the target audio segment most recently captured audio of a musical performance as this is being played. Now explain briefly three applications of real-time tracking of musical performances: (a) The virtual music teacher, (b) auto-accompaniment of soloists, and (c) automatically added special effects in live performance. For the virtual master of music, the sound objective is well-executed piece of music by a professional musician. Musical performance which will be followed is that produced by the student. A music teacher teaches only one student at a time, as the student plays his instrument, the teacher makes comments pointing out errors or endorse the interpretation. A virtual teacher should do the same, use the audio signal generated by the student, and processed in real time and provide appropriate instructions to the student. The advantage of a virtual teacher that can generate many instances of the same and therefore simultaneously serve many students. For the automatic accompaniment of a singer or a solo audio music performance, the target audio would be made possibly in a recording studio, while the music that will be followed up is the live performance of the musician or singer. The system plays music accompaniment while generating activities (e.g., introducing delays in background music), these systems are true

An earlier version of this work was presented at the 9th Mexican International Conference on Artificial Intelligence (MICAI'2010) [8].

A. Camarena-Ibarrola (✉)
División de Estudios de Posgrado, Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, Morelia, Michoacán 58000, Mexico
e-mail: camarena@umich.mx

E. Chávez
Facultad de Ciencias, Físico-Matemáticas, Universidad Michoacana de San Nicolás de Hidalgo, Morelia, Michoacán 58000, Mexico
e-mail: elchavez@umich.mx

virtual orchestras. Regarding the automatic addition of special effects, the target audio is the audio recorded during a practice session of the event while the musical performance that will be monitored is generated in the live event. The system can for example turn on or off lights, play sound effects or even throw fireworks at specific moments in the audio marked target. The above examples are more complex because they involve the polyphonic music and even sounds that can be confused with noise. Monophonic music is more simple for monitoring purposes as described in related work.

2 Related work

Monitoring systems in real time music is characterized by two features: (a) The features extracted from the audio signal, and (b) the technique used to follow the interpretation regarding the audio target. The features are extracted from the audio signal which replaced for the next stage, monitoring, which is conducted by an online alignment tool, the end result is a sequence that locates each of the musical moment on the audio target. In [18], the dynamic pace of the signal is used to characterize monophonic music. For polyphonic music, in [5] were used as features the energy, changing energy, zero crossings, the fundamental frequency and fundamental frequency switching. In [12, 13], the overall tone, chroma values, the flux and spectrum cepstral up the collection of features that the authors used for real time monitoring all these features are extracted from the signal without involving loss of time reference, the audio is replaced by a sequence of features. Both audio and musical performance objective are compared as strings of vectors or symbols, each symbol represents vector or the same amount of time or frame. The selection of an appropriate alignment tool complements a real time system monitoring music. The alignment algorithm establishes the correlation between the characteristics of both signals. The classical approach to alignment of two sequences is called the *dynamic time warping* (DTW) [17]. DTW is to determine the optimal bending feature set as one of the sequences should be shortened or lengthened to reduce the differences between the two sequences to a minimum. DTW was originally designed to align two sequences that are both known*a-priori*. In our case, only one sequence is known (the audio target) while the musical performance will get to know as they captured it, off-line alignment is not useful. Recently, in [6, 7] a variation of DTW called*on-line timewarping* was proposed as an adaptation to enable online tracking of musical performances, surges of tone and increases in energy band frequency are used as features. Doubling of time online trying to predict the optimal path bending by partial filling of the cost matrix in fixed-size window (the window size is a parameter of the algorithm), if the minimum cost is at the right end window, then fill the array with the column comes next, if instead the minimum is at the bottom of the window, then fill the array of costs on to the next line. The algorithm progresses in both line and column where the minimum cost is in the pseudo-diagonal. The real diagonal is not known in reality as the length of one of the series (musical performance which makes it up) is unknown. In practice, the algorithm of on-line time warping tends to deviate from the optimal path because the error is cumulative, this algorithm can completely lose track of the interpretation which seeks to track.

Natural alternative alignment are hidden Markov models (HMW) as in [5]. A HMM is a doubly stochastic process is not observable (it's hidden), and can be observed only through another set of stochastic processes that produce the sequence of observed symbols [14]. Determine the parameters of a HMM given a sequence of observations is a problem known as*training*,and is met by the Baum–Welch algorithm [1]. Using a HMM (e.g. for monitoring purposes) is the called evaluation problem and is solved both by the *forward* and the *backward* procedures [15]. Finally, the *Viterbi* algorithm is used to discover the hidden part of the model, which defines the interconnection between the states of HMM. Some critical decisions have to be resolved by the designers of the HMM, for example the number of states. A recent alternative for tracking of musical performances is the use of particulate filters and particle swarm optimization [18], this is an interesting approach in which a cycle prediction/correction is used to discover a collection of both audio rhythms objective as in music performance. It makes a number of statistical assumptions (like independence) of the variables that represent the rhythms. The end result is an approximate matching of these. The cycle of prediction / correction makes the use of this technique in a real-time application.

The rest of the paper is organized as follows. In the next section we describe our novel technique to track real-time musical performances, provide details of both the feature extraction module (determination of the audio track) as the index we use instead of the traditional approach is to use an alignment tool. In Sect. 4 we describe the collection to testing, the experiments and results. In Sect. 5 we write the conclusions we reached based on the results of the experiments and suggest some future extensions to this work.

3 Our proposal

State-of-art approaches use only local information to keep track of interpretations, ie they assume no tracking error has occurred up to the point, then read the next segment of

audio and decide whether to move or position within the target audio. If a tracking error has actually occurred, have no way to correct and end up lost in the monitoring process and would not even realize it because they gradually lost. When using traditional techniques for monitoring alignment of interpretations, errors are cumulative, a misalignment in time t_0 will not be corrected at any $t > t_0$. The inductive hypothesis for both the dynamic bending time to HMWs is that the alignment is correct for all previous time frames and use this information to align the current time frame. One of the reasons for using local alignment in real time applications is that a sequential search over all audio target is too expensive. We think it is more natural not to assume anything about the past in the process of identifying commonalities, this allows the recovery of previous mistakes most natural way. We face two challenges in this task, the first is to make a robust feature extraction, and the second challenge is the search through the target audio without implying a high cost in time and then allows us to track real-time rendering.

To identify a song using a short piece for this in [10] were used for this purpose pieces as short as 3 s by using an audio fingerprint based on the second derivative of energy.

We conducted experiments in which we can identify a song with pieces of only 1-s using the described in [3], the feature extracted from the audio signal is multi-band spectral entropy, not only recognized the song but the precise spot where the piece of a second is located within the song, although this piece has a high noise level. This is just what we need to characterize a musical performance for monitoring purposes which is the first part of our method. For the second part, find the piece of audio throughout the audio goal requires time proportional to the size of the target audio. This time can be significantly improved by using an index to perform searches. The general idea is to build a metric index with all possible pieces of a second long audio taken from the target, the index can be searched with sublinear complexity which allows us to apply it to real-time tracking. The combination of these two ideas was a powerful tool with the ability to recover from any error of alignment is very fast and even tolerates noise both on the audio track and the musical performance. For convenience of the reader, described below in separate sections feature extraction and indexing technique.

3.1 Multi-band spectral entropy signature

In [3], the removal of a very robust audio fingerprint based on the evolution over time of entropy spectral frequency bands was used for the purpose of music retrieval by content. The same audio track was successfully used for automatic monitoring of radio stations in [4] with excellent results. We have now adapted this audio signature based on multi-band entropy (MBSES) for monitoring online musical performances, the resulting MBSES is determined as follows:

1. The signal is processed in time frames of 185 ms, the frame size ensures time support for the calculation of entropy. The frames are overlapped by three quarters (75%), then you will get a feature vector every 46 ms or so.
2. Apply the Hann window to the time frame and then determines the discrete Fourier transform.
3. The resulting spectrum, the entropy is determined (according to the formulation of Shannon) for each of the first 24 critical bands according to the Bark scale is computed for each one of the first 24 critical bands According To the Bark scale (frequencies between 20 and 15,500 Hz) using the Eq. 1 also denoted as σ_{xx} and σ_{yy} are the variances of the real and imaginary part respectively, σ_{xy} is the covariance between them.

$$H = \ln(2\pi e) + \frac{1}{2}\ln(\sigma_{xx}\sigma_{yy} - \sigma_{xy}^2) \tag{1}$$

4. For each band, determine if the entropy is increasing or no (compared to the same band of the previous time frame). Equation 2 sets the corresponding bit is determined for the band b and frame n for the signature using $H_b(n)$ and $H_b(n-1)$ (The spectral entropy of band b for frames n and $n-1$ respectively). Only 3 bytes are necessary to save the signature for every 46 ms of audio.

$$F(n,b) = \begin{cases} 1 & \text{if } [H_b(n) - H_b(n-1)] > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

A diagram of the process to determine the MBSES of an audio signal is shown in Fig. 1.

3.1.1 *Measuring deviations of the audio*

The MBSES of a audio audio is a binary matrix, it can be shown graphically as a black and white image. Figure 2a shows graphically the binary matrix of 24 × 24 for a piece of a second long take of a song. Figure 2b shows the MBSES of a noise contaminated version of the same piece of audio where the signal to noise ratio (SNR) is 3 dB. To evaluate how different are two pieces of audio we use the Hamming distance, the distance is determined by counting the number of bits that are different between two specific binary matrices. Figure 2c shows the exclusive-OR operation (XOR) between the matrices shown in Fig. 2a, b. Hamming distance between these matrices is simply the number of bits on in the matrix resulting from the XOR

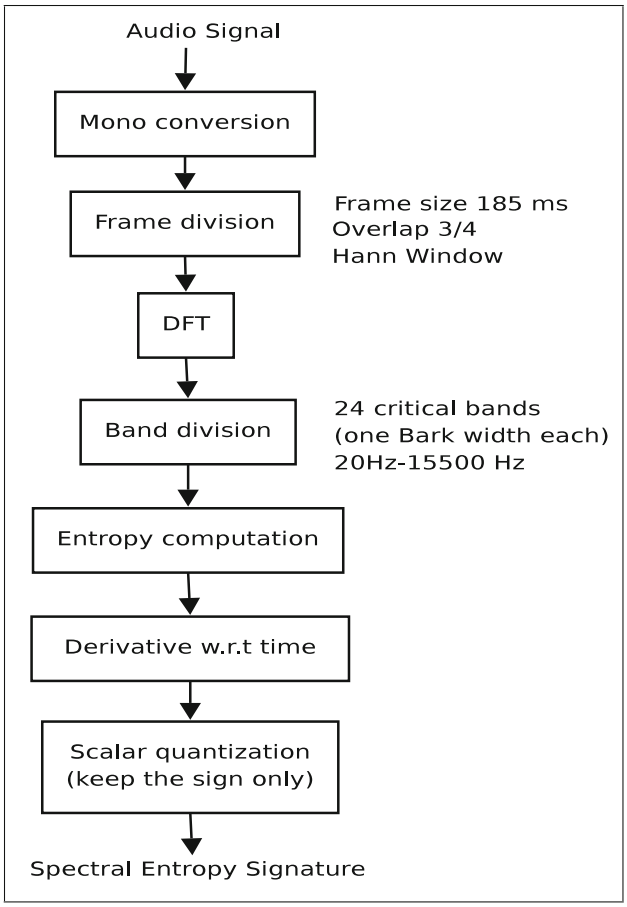

Fig. 1 Determination of the multiband spectral entropy signature (MBSES)

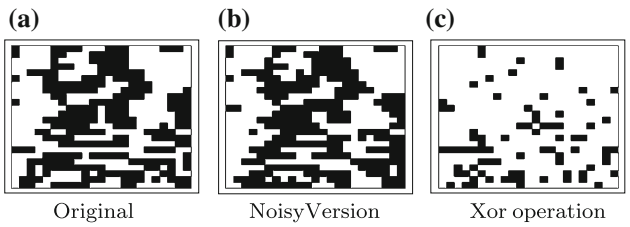

Fig. 2 *Left* MBSES from a piece of a second duration of a song. *Center* MBSES from the same piece but contaminated with noise (SNR = 3 dB). *Right* the XOR operation between the two binary matrices is related to the Hamming distance between them

operation, ie the number of black dots in Fig. 2c. Normalized Hamming distance ranges from zero to one, to normalize the distance, divide by the maximum distance in our case 576.

3.2 Tracking musical performances

After determining the MBSES of each of the overlapping time frames, the song is represented as a binary matrix with

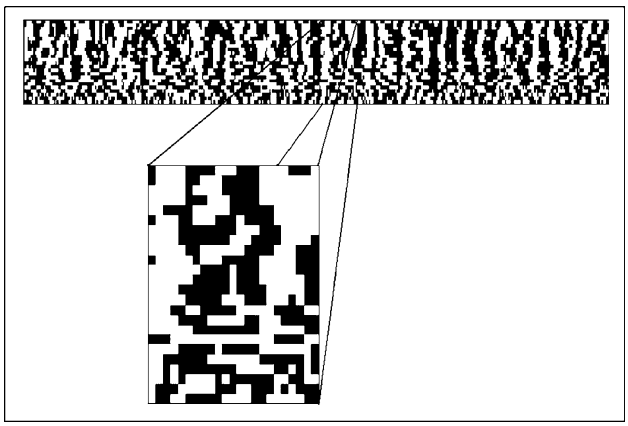

Fig. 3 The MBSES of a piece of a song depicted graphically

24 rows (24 Bark bands) and the number of columns depends on the length of the song. In Fig. 3 the MBSES of a excerpt of a song about 16 s long is shown, in the same figure the binary submatrix corresponding to about 1 s of audio is magnified.

If we wish to pursue a musical performance we could extract every possible sub-array of 24 by 24 from the MBSES interpretation (referred to as audio target) of the song. Each sub-matrix corresponds roughly to 1 s of audio, and these sub-matrices may be labeled by its position within the audio target. Such a collection of matrices form training data basis for a recognition system chips audio. The data-base stores the audio track of each piece and its location within the song. The system recognition of pieces reports the location of the test piece if indeed they identify it as belonging to the audio objective. It is possible, according to our experiments that some pieces of the song being followed are very similar to most pieces at the audio end. This is very common for pieces of the chorus part (repetitive pieces of a song). Based on this observation, our tracking system musical performances should not simply use the piece identified as the nearest neighbor because of the risk of confusion with another piece of the same song which forces the tracking system to jump from one place to another. Instead, we look to the k nearest neighbors and just read the piece of musical performance which will be monitored and choose the one whose location is the nearest after the last position reported by the monitoring system. A sequent search through all the audio is the non-functional goal for lengthy songs because the search time would be greater than the maximum time allowed in a real time system, than that needed for another solution. Fortunately, there are a wide variety of indexes to locate closer k neighbors and other types of searches (like range search) sublinear complexity with time. We describe one of these indices.

3.3 Metric indexes

A metric index organizes a set of data using a distance function to answer questions quickly (data distance function over form which is known as "space metric"). Queries that concern us are the ones we delivered to the k nearest neighbors among all possible pieces obtained from the audio target. We selected the Burkhard–Keller tree (BK-tree) because of its simplicity since the intrinsic metric space dimension that concerns us is not great. Figure 4 shows a BK-tree constructed from MBSES hypothetical as shown in the top of Fig. 4. Fourteen binary sub-matrices are extracted and added to the BK-tree, each corresponding to an audio piece referred to as a, b, …, n. For space reservation, in Fig. 4 a, b, …, n are 6×6 binary matrices. The matrices actually extracted are 24×24 as explained previously. The first sub-matrix we read (the parent) becomes the root of the BK-tree, the second (matrix b) compared with the first using the Hamming distance which turns out to be 21 (21 bits are different) so it becomes a son of the root labeled under the League 19–24 since $19 \leq 21 \leq 24$, reads the third matrix (matrix c) which is inserted immediately, then compared with the root, its Hamming distance turns out to be 17, so that becomes another child of the root under the League 13–18 ($13 \leq 17 \leq 18$). To insert the d matrix, this is first compared to the root, as the distance falls in the range of 19–24 and there exists a node (matrix b), then array d must be compared with the matrix b. Matrix b is added as a child of the parent b under the Leage 7–12 (Hamming (b, d) = 12). The remaining matrices are inserted in the BK-tree in the same way.

The BK-tree for the example of Fig. 4 has a ring width $s = 6$, this is the range of distances, grouped by the same league under a particular node. To understand this, observe Fig. 5, The BK-tree root in Fig. 4 (sub-matrix a) is located in the center of the metric space. Six rings are displayed around sub-matrix "a", each with a $s = 6$ width. Any sub-matrix must fall within one of these rings as the maximum Hamming distance between two 6×6 binary matrices is 36. Any sub-matrix is a descendant of the same root child located within the same ring. Then the submatrices "e" and "m" fall within the same ring, as well as sub-matrices "c", "g", "k", "i", "h", and "j" form a sub-tree and remain within the same ring. Any node that is a child of the root is in turn the root of another BK-subtree, and are themselves the center of another set of rings, for example the sub-matrices "n", "d" and "f", which fall within the same ring centered in sub-matrix "b". Since these sub-matrices along with the sub-matrix "l" form a subtree, then all fall within the same ring centered in the sub-matrix "a". To find the nearest neighbor, the method explores the rings whose distance from the center are dwindling like the distance between the center and the consultation [9].

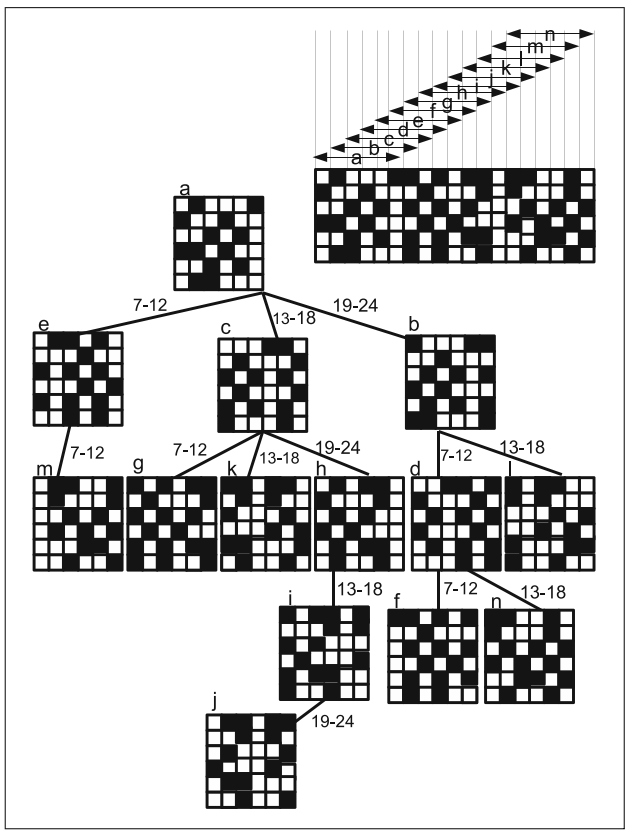

Fig. 4 One-second excerpts are indexed using a BK-tree

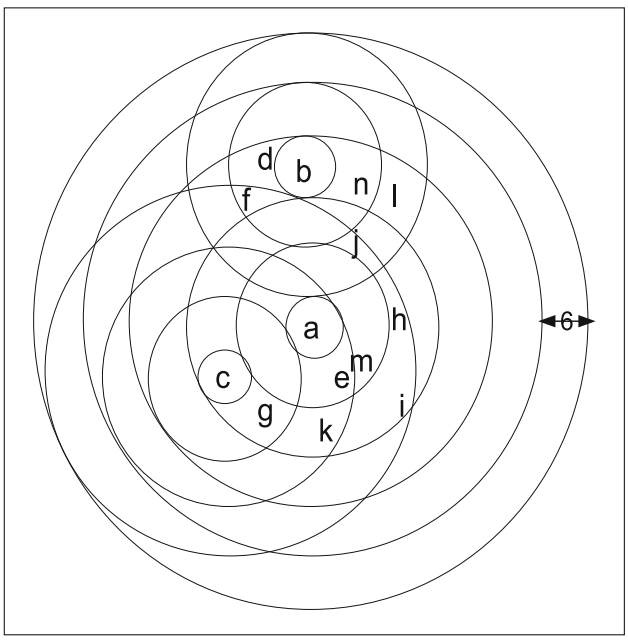

Fig. 5 Metric space related to the BK-tree shown in Fig. 4

3.3.1 Space requirements

In the implementation of the BK-Tree there is no need to store redundant information, instead of extracting a 24×24 binary sub-matrix for each 46.25 ms frame and add it to the BK-Tree. We simply store a pointer to the location of the 24×24 sub-matrix binary within the full target audio MBSES.

Then, the space required is to store the necessary Audio MBSES target and the corresponding pointers of the BK-Tree index. For example, for a 4 min song of 5,190 frames, the MBSES requires $5190 \times 3 = 15{,}570$ bytes, i.e. only 15.2 Kbytes for storing the audio MBSES and about 50 Kbytes for the BK-Tree. Our index also has a bucket for each node to save pointers. Our BK-tree ring has a width of 24 and a bucket with 12 entries. See [9] for details. Our estimate for memory requirements of our system is 280 bytes for each second which is really low.

4 Experiments

For each of the songs of a set of 62 we get two different musical performances. For example, for *Beethoven's Fifth Symphony in C minor performed by Berlin Philharmonic conducted by Karajan*, we get the version of the *Philharmonic Orchestra Directed by Kleiber Vienna*. This collection of songs and musical pieces is our test suite.

For each pair of musical performances from our collection, we take one as the audio target, and the other as the interpretation which will be followed up. We determine the MBSES of the audio target and build the BK-tree index with ring-width $s = 24$ using each 24 by 24 sub-matrix taken from this MBSES, remembering that a song is a 24 rows matrix and a number of columns depending on the duration. For example, the MBSES of a 4 min song is a matrix of 24 rows and about 5,200 columns (one column every 46 ms of audio). There were about 5,200 sub-matrices extracted (overlapping by 23 columns) and each of these sub-matrices were inserted into the BK-tree corresponding to the song in question. The monitoring process will proceed as follows: For each second audio the MBSES extract is received extract (which would be a binary matrix of 24 by 24) Then seek the k neighbors closest to the target using the audio for this index. Of the k neighbors select one that is closest at the time of the last position reported by the same monitoring system. Then read another second of audio and repeat the process until the end of the musical. We then actually used two separate distances, the Hamming distance to search the metric space similar to that as shown in Fig. 5 the subset of k candidates. Each candidate has an associated location in time so we then use the second distance, ie the difference between the last time position reported by the monitoring system and the time position of each candidate in the target audio, later selecting the closest candidate. When $k = 1$, the second distance is not used.

Algorithm 1 tracks flow Audio and reports the location of each segment of a second incoming audio on the audio end. The BK-tree of the audio goal should be built prior to using this algorithm.

Input: BK-Tree S of Score, audio-stream A to be tracked, number of neighbors K
Output: Stream of positions relative to the score
PP is a Binary Matrix of 24 by 24;
t is an array for K positions inside the score (integers);
$CurrentTrackingPosition \leftarrow 0$;
while *There are more one second segments of audio in input stream* **do**
  Read one second of audio from A, determine its MBSES and store in PP;
  Search the K-nearest neighbors of PP using Bk-tree S and store in t their locations inside the score;
  Search among the K-nearest neighbors the one whose location (stored in t) is closest in time to CurrentTrackingPosition and store its location in p;
  $CurrentTrackingPosition \leftarrow p$;
  Report $CurrentTrackingPosition$;
end

Algorithm 1 Algorithm to track an audio streamreporting every one-second segment position in relation to a score

Figure 6 depicts the tracking process of an interpretation of the Beatles song "All My Loving". The curves show for each second of interpretation where it was located within the target audio. Note the peaks in Fig. 6 for low k. These peaks are in fact tracking failures because they involve a jump from a position that is not close in time, and we call these *false locations* or *tracking failures*. Repeating monitoring for different k we see that such false locations tend to vanish as k increases. In the example shown in Fig. 6 tracking failures disappear for $k = 6$.

Peaks as shown in Fig. 6 are displayed when none of the k nearest neighbors are located near the last position reported by the monitoring system. As k increases, the probability that none of the k nearest neighbors occur near the current position decreases, which is why the peaks almost disappear for large k.

An extreme example of tracking failure can be seen in Fig. 7 corresponding to Nutcracker Suite by Tchaikovsky. At the very beginning of the performance, a succession of location errors move the tracking position from the beginning to near the end. However, as the nearest sought

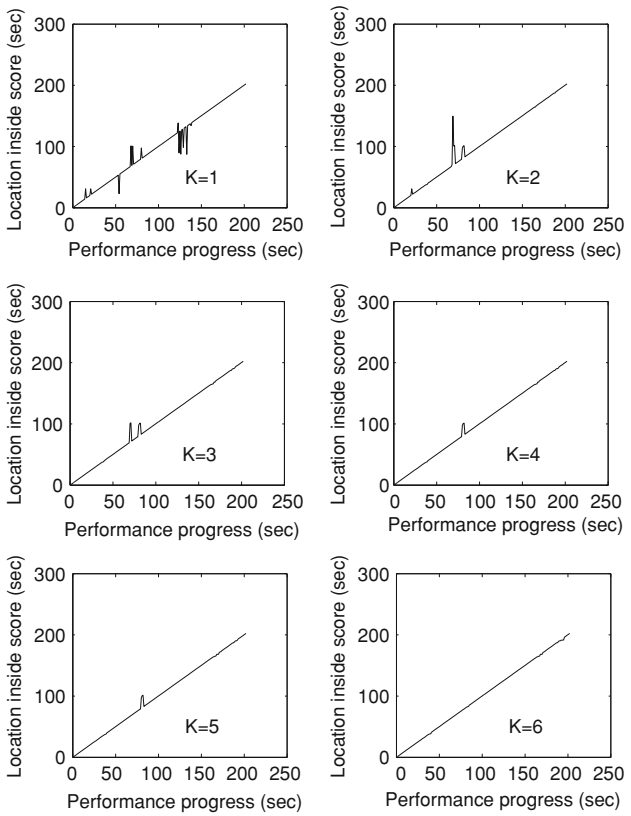

Fig. 6 Tracking of a performance of "All my loving" (The Beatles) for different K (Number of nearest neighbors considered). Some tracking failures can be observed for low K

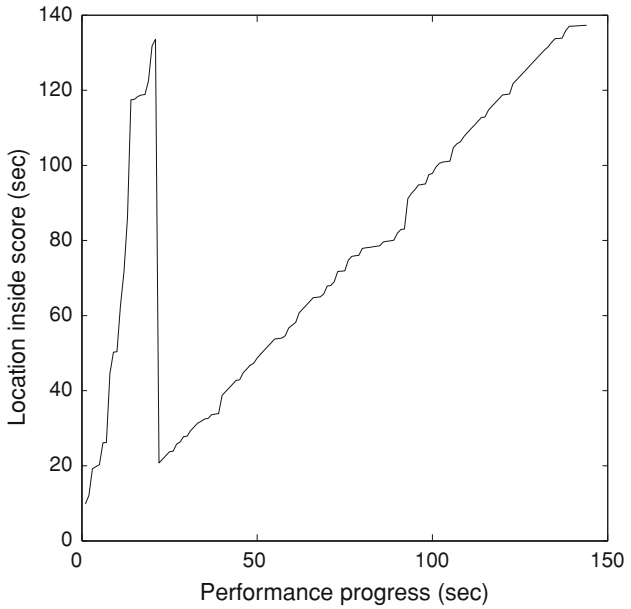

Fig. 7 An example of recovery after a failure in following *the Nutcracker* using only a neighbor ($k = 1$)

k neighbors are global (for all audio target) and not local (only near the tracked position) the error is eventually corrected as shown in Fig. 7. In this case an error no longer recurs and progresses properly until the end of the performance.

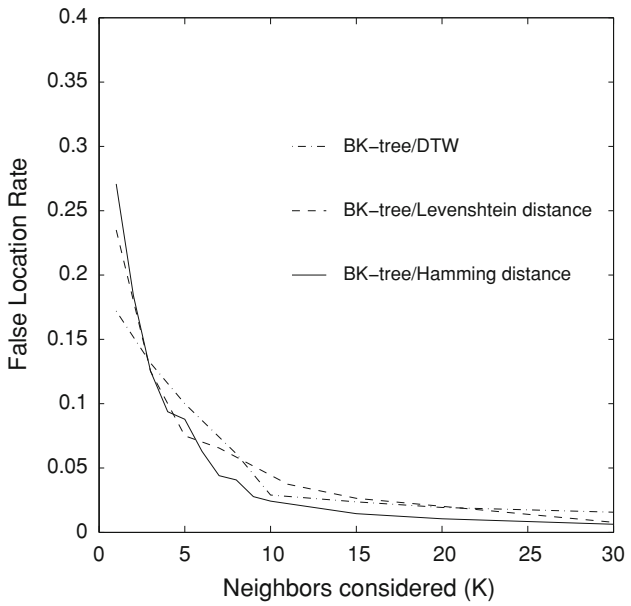

Fig. 8 False locations regime (FLR) obtained after following up each of the songs in our library and varying k (The number of nearest neighbors considered). FLR falls as k increases

Repeating the process described above for the rest of the musical performances we saw from our collection of tests that our system always recovered (and most of the time immediately) when a false position occurred so that the system never lost track of the musical. A false location is posted whenever the position reported by the monitoring system changed abruptly to one too distant in time compared to the previous position reported by the same system (We use 5 s as the threshold). We determined the system's false location rate (FLR) for k in the range of 1–30 with Eq. 3.

$$FLR = \frac{\text{False locations}}{\text{False locations} + \text{True locations}}. \quad (3)$$

Figure 8 shows the FLR obtained after tracking all songs Library and vary k.

We were concerned with the fact that the temporal evolution inside the 1-s audio segments might be too different between different musical interpretations of the same song. So we decided to use distances that take that into account. What we needed was a distance to establish how two different binary matrices of size 24 × 24 were at the same time flexible enough to allow for time alignments between the pieces of a second that were compared. The dynamic time warping (DTW) gives us a distance that meets this requirement. To determine DTW dynamic programming is used [16]. The matrix D is the cost matrix

where the first column and row are filled by Eqs. 4 and 5 respectively. The rest of the matrix is filled by Eq. 6. Finally the DTW distance is the value left when the last cell filled. In our case it is $D(24,24)$.

$$D_{i,0} = \sum_{k=0}^{i} d_{i,0} \tag{4}$$

$$D_{0,j} = \sum_{k=0}^{j} d_{0,j} \tag{5}$$

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} + 2d_{i,j} \\ D_{i-1,j} + d_{i,j} \\ D_{i,j-1} + d_{i,j} \end{cases} \tag{6}$$

$d_{i,j}$ is the Hamming distance between the i-th column of the MBSES from a piece of a 1-s of the audio interpretation being followed up, and the j-th column of the MBSES of a 1-s from the audio target. Note that this is not Hamming distance between binary matrices but between two columns of 24 bits each.

Another distance that sets how different two binary matrices are and allows time alignment is the Levenshtein distance [11]. The Levenshtein distance between two strings is the minimum number of edit operations (insertions, deletions and substitutions) to convert one into the other. To calculate the Levenshtein distance between string t of length N and string p of length M, Eqs. 7, 8 and 9 can be used. In our case $M = N = 24$.

$$L_{i,1} = i \qquad \forall \quad i = 1 \ldots N \tag{7}$$

$$L_{1,j} = j \qquad \forall \quad j = 1 \ldots M \tag{8}$$

$$L_{i,j} = \begin{cases} L_{i-1,j-1} & t_i = p_j \\ \min \begin{cases} L_{i-1,j-1} + d_{i,j} \\ L_{i,j-1} + 1 \\ L_{i-1,j} + 1 \end{cases} & t_i \neq p_j \end{cases} \tag{9}$$

for $j = 2 \ldots M$ and $i = 2 \ldots N$, $d_{i,j}$ is the Hamming distance standard (at 0.0–2.0) between the i-th column from the MBSES of a 1-s segment of the musical performance and the j-th column from the MBSES of a 1-s piece of the target audio. The costs of insertion and deletion are both 1.0 while the cost of replacement was the normalized Hamming distance.

As seen in Fig. 8 we obtained a smaller value of fake locations regime low k when we use the Levenshtein distance or DTW distance compared to the FLR obtained when using only Hamming distance to compare the pieces of audio 1 s. This confirmed our expectations considering the different temporal evolution of a musical performance and audio target. However, as k increases, the FLR decreases and surprisingly, has almost no noticeable significant advantage over the use of the Hamming distance for high k and since the Hamming distance is calculated much faster than the DTW distance or the Levenshtein distance we conclude that the Hamming distance is the most suitable for our purpose.

In a real scenario, a musical performance is distorted by the presence of noise, so we mixed all our musical performances with children playing and shouting (recorded at a party), after mixing the signal to noise ratio (SNR was approximately 5 dB. Non mixed files were used as the target audio. After repeating all the experiments using this distorted set the results are very similar to those obtained before and can be seen in Fig. 9. This occurred thanks to the robustness of the audio fingerprint based on multiban spectral entropy (MBSES).

4.1 Avoiding false locations

For practical purposes, tracking failures should not lead to the implementation of actions that should be avoided. False locations appear due to the fact that none of the k neighbors occur close (in time) from the last position reported by the system. A simple way to prevent implementation of undesirable actions can be set as follows: When none of the k nearest neighbors occur near the last position reported by the monitoring system, that position should not move as if the latest piece of audio had not been received, with this modification. The false locations do not seem to occur even for $k = 1$. The Algorithm 2 tracks performances and avoids false locations.

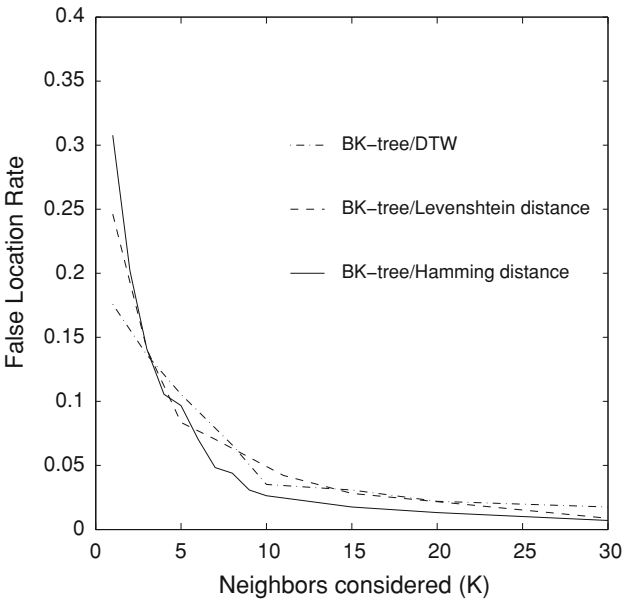

Fig. 9 False locations regime obtained after tracking all the songs with noise from our collection and vary k

Input: BK-Tree S of target audio, audio-stream A to be tracked, number of neighbors K
Output: Location flow relative to target audio
PP is a Binary Matrix of 24 by 24;
t is an array for K positions within the target audio (integers);
$CurrentTrackingPosition \leftarrow 0$;
while *There are more one-second pieces of audio in the input stream* **do**
  Read one second of audio from A, determine its MBSES and store in PP;
  Search the K-nearest neighbors of PP using Bk-tree S and store in t their locations inside the score;
  Search among the K-nearest neighbors the one whose location (stored in t) is closest in time to CurrentTrackingPosition and store its location in p;
  if $|CurrentTrackingPosition - p| < threshold$ **then**
    $CurrentTrackingPosition \leftarrow p$;
  end
  Report $CurrentTrackingPosition$;
end

Algorithm 2 Algorithm to track an audio streamimproved to avoid false locations

Once this change is made to our monitoring system peaks as in Fig. 6 do not occur, even for $k = 1$. Figure 10 shows a tracking of the first movement of Mozart's Requiem. This follow-up was performed, avoiding false locations according to Algorithm 2 with a threshold value (threshold) of 5 s.

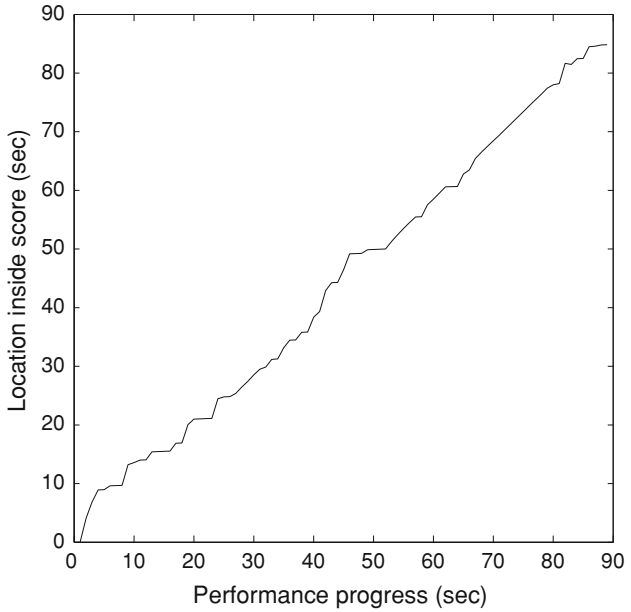

Fig. 10 Tracking of the first movement of Mozart's Requiem, using Algorithm 2 avoiding false locations with a threshold value of 5 s

4.2 Time analysis

Two steps are required for locating within the target audio a 1-s segment of audio taken from the musical interpretation. The first step is MBSES Extraction from the audio signal. The second step is finding the k nearest neighbors to a 24×24 binary matrix extracted in the previous step using the BK-tree as it is implemented in the library SISAP [9, 2]. Using a laptop Dual-core 1.46 GHz Pentium with 1 GB of memory, it took 130 ms to complete the first step, and only 10 ms for the second step. The BK-tree construction is carried out prior to the process monitoring so that the duration of this process is not critical. It is an off-line process equivalent to the Training phase of a HMW. An application of real time such as discussed in the Motivation section used approximately 140 ms for each second of audio received without parallelization.

We look for pieces of a second in a BK-Tree, but that does not mean we must accept the 1-s delay to start the follow-up process that actually processes the audio time frames of 46.25 ms. We begin with a binary matrix 24×24 which is full of zeros. After reading the first part of the audio, we made a shift in the columns of the matrix bit to the left and updated the last column of the binary matrix. The process is repeated each time another 46.25 ms frame of audio is received. In this way we can start to follow immediately after receiving the first ms 46.25 frame. The number of frames read before search in the audio target is a parameter of the system.

5 Conclusions and future work

We have successfully carried out tracking of musical performances by searching for pieces using a metric index. We show that it is possible to use a global alignment technique, avoiding error accumulation unlike standard approaches. Our method does not require an iterative training process (parameter estimation). Unlike the online time warping approach, we do take advantage of the fact the target audio is known *a priori* (which is certainly an advantage of the approach with HMM's).

We emphasize that in view of our global alignment technique approach is that we can recover from false locations (errors) that occur. An error does not imply that the system loses audio track. Finally, our approach has the additional advantage that follow-up may begin at any time. No alternative approach has the ability to start when the musical performance has begun. For example, if the system is turned on after the onset of interpretation.

There are other metrics that we should consider alternatively when the metric space consists of more than a

song. This can be useful to index an entire collection of songs instead of indexing each song separately as we did in this work.

Acknowledgments We thank the reviewers for their helpful suggestions and comments, and thank Dr. Grigori Sidorov for his help improving the wording of this article.

References

1. Bilmes JA (1998) A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical report TR-97-021, Department of Electrical Engineering and Computer Science U.C. Berkeley
2. Burkhard WA, Keller RM (1973) Some approaches to best-match file searching. Commun ACM 16(4):230–236. doi:http://doi.acm.org/10.1145/362003.362025
3. Camarena-Ibarrola A, Chavez E (2006) On musical performances identification, entropy and string matching. In: Fifth Mexican international conference on artificial intelligence 2006 (MICAI2006)
4. Camarena-Ibarrola A, Chavez E, Tellez ES (2009) Robust radio broadcast monitoring using a multi-band spectral entropy signature. In: 14th Iberoamerican congress on pattern recognition. Springer, pp 587–594
5. Cano P, Loscos A, Bonada J (1999) Score-performance matching using hmms. In: ICMC99. Audiovisual Institute, Pompeu Fabra University, Spain
6. Dixon S (2005) Live tracking of musical performances using online time warping. In: 8th International conference on digital audio effects (DAFx'05). Austrian Research Institute for Artificial Intelligence, Vienna
7. Dixon S, Widmer G (2005) Match: a music alignment tool chest. In: 6th International conference on music information retrieval (ISMIR). Austrian Research Institute for Artificial Intelligence, Vienna
8. Edgar Chávez ACI (2010) Real time tracking of musical performances. In: 9th Mexican international conference on artificial intelligence (MICAI'2010), LNCS. Springer, pp 138–148
9. Figueroa K, Chávez E, Navarro G (2010) The SISAP metric indexing library.URL http://www.sisap.org/Metric_Space_Library.html
10. Haitsma J, Kalker T (2002) A highly robust audio fingerprinting system. In: International symposium on music information retrieval ISMIR
11. Navarro G, Raffinot M (2002) Flexible pattern matching in strings. practical on-line search for texts and biological sequences, vol 17. Cambridge University Press
12. Orio N, Déchelle F (2001) Score following using spectral analysis and hidden Markov models. In: Proceedings of the ICMC, pp 151–154
13. Orio N, Lemouton S, Schwarz D (2003) Score following: state of the art and new developments. In: Proceedings of the 2003 conference on new interfaces for musical expression. National University of Singapore, p 41
14. Rabiner L, Juang B (2003) An introduction to hidden markov models. ASSP Mag IEEE 3(1):4–16
15. Rabiner RL (1989) A tutorial on hidden markov models and selected aplications in speech recognition. Proc IEEE 77(2):257–286
16. Rabiner RL, Rosenberg AE, Levinson SE (1978) Considerations in dynamic time warping for discrete word recognition. In: IEEE trans on acoustics, speech and signal processing ASSP-26, pp 622–635
17. Sakoe H, Chiba S (1978) Dynamic programming algortihm optimization for spoken word recognition. IEEE transactions on acoustics and speech signal processing (ASSP), pp 43–49
18. Sethares W, Morris R, Sethares J (2005) Beat tracking of musical performances using low-level audio features. IEEE Trans Speech Audio Process 13(2):275–285