ON FINDING SYMBOLIC THEMES DIRECTLY FROM AUDIO FILES USING DYNAMIC PROGRAMMING

Antti Laaksonen[1], **Kjell Lemström**[2,1]
[1] Department of Computer Science, University of Helsinki
[2] Laurea University of Applied Sciences
ahslaaks@cs.helsinki.fi, kjell.lemstrom@laurea.fi

ABSTRACT

In this paper our goal is to find occurrences of a theme within a musical work. The theme is given in a symbolic form that is searched for directly in an audio file. We present a dynamic programming algorithm that is related to an existing time-warp invariant algorithm. However, the new algorithm is computationally more efficient than its predecessor, and it can also be used for approximate time-scale invariant search. In the latter case the note durations in the query are taken into account, but some time jittering is allowed for. When dealing with audio, these are important properties because the number of possible note events is large and the note positions are not exact. We evaluate the algorithm using a collection of themes from Tchaikovsky's symphonies. The new approximate time-scaled algorithm seems to be a good choice for this setting.

1. INTRODUCTION

In this paper we present new content-based music retrieval (CBMR) algorithms for discovering occurrences of a given musical theme, called the *pattern*, in a musical work under consideration, called the *database*. The algorithms have both important scientific applications and theoretical interest: they can be used in music analysis to locate occurrences of themes in a single musical work or to search for a given theme within a music database. Moreover, to the best of our best knowledge, our algorithm for time-warped search is the most efficient algorithm available for the task.

We allow both the pattern and the database to be polyphonic. Traditionally CBMR problems like this have been tackled by using a linear string representation combined with a string matching algorithm. However, the representation does not effectively capture important intrinsic features of music. The geometric modeling of music [8] is more appropriate for this case and has recently been successfully used by several authors [4, 7, 10, 13]. Reasonable geometric modelling allows the matching process to ignore extra intervening notes in the database that do not appear in the query. Such extra notes always occur because of polyphony and unexpected noise.

Until recently, the geometric framework suffered from the fact that the timing of the notes had to be exact. Recent studies have challenged this problem. First, in [5, 10] the occurrences were allowed to be transposed and time-scaled copies of the query. Under this *transposition and time-scale invariance* (the *TTSI* setting), however, the queries still need to be given exactly in tempo. Under a more realistic, *transposition and time-warp invariance* (the *TTWI* setting), local time jittering is allowed for in every note-onset. The first solutions for this setting were introduced by Lemström and Laitinen [4, 6].

In this paper we introduce a new dynamic programming algorithm that can be used for both time-scale and time-warp invariant search. As an application of the algorithm, we search for musical themes from audio files. Only a few other algorithms for symbolic queries from audio have been proposed in the literature [2, 11, 12]. Our experiment shows that our algorithms are efficient in practice, and the music search results from audio files are also promising.

Let us next define the problems under consideration. The problems are elaborations of Ukkonen, Lemström and Mäkinens P1 problem [13]. In each case, the music is represented in a pitch-against-time representation of note-on information. The database T contains n note events, and the query pattern P contains m note events. In a typical retrieval case, the pattern P is monophonic and much shorter than the polyphonic database T. The problems are:

W1: Find *time-warped translations* of P such that each note in P matches with a note in T [4, 6].

S1: Find *time-scaled translations* of P such that each note in P matches with a note in T [5, 10].

AS1: Find *approximate time-scaled translations* of P such that each note in P matches with a note in T.

The new problem (AS1) can be seen as an intermediate form of W1 and S1. Instead of a constant time-scaling factor α, we allow the scaling to vary in a range $[\alpha/\epsilon, \alpha\epsilon]$ where $\epsilon \geq 1$. If $\epsilon = 1$, the problem is the same as S1.

2. ALGORITHMS

The dynamic programming algorithm of Laitinen and Lemström [4] solves the problem W1 in $O(nmw)$ time where w is the window size. In this section we present a modification for the algorithm that reduces the time complexity to

$O(nm)$. Furthermore, the new algorithm also solves problems S1 and AS1 in $O(nms)$ time where s is the number of possible scaling factors.

The input of the algorithm consists of two lists: the database notes $T[1], T[2], \ldots, T[n]$ and the pattern notes $P[1], P[2], \ldots, P[m]$. Each database note $T[k]$ is a triplet: $T[k].time$ is the onset time, $T[k].pitch$ is the pitch value and $T[k].score$ is a real value in the range $[0, 1]$. Each pattern note $P[k]$ is a pair with the fields $P[k].time$ and $P[k].pitch$. We assume that both of the lists are sorted in increasing order according to the onset times and that the pitches are represented in semitones.

The score field of the database notes is only used with audio material. If the input is symbolic music, each score is simply set 1. However, when audio material is used, the note events are approximations. The idea is that the score reflects the amplitudes of the frequencies that correspond to the note in the audio.

The output of the algorithm is a list $S[1], S[2], \ldots, S[n]$ consisting of real values. Each value $S[k]$ is the maximum score for a pattern occurrence starting from the database note k. In an occurrence, each pattern note is matched with a database note, and the score is calculated as a sum of individual scores of the database notes. In the case of a symbolic database, a score of m denotes that the pattern is found, and in the case of an audio database a high score suggests that the pattern is found.

Next, in Section 2.1 we present our modifications to the time-warped search algorithm. After this, in Section 2.2, we show how the algorithm can also be used for time-scaled search.

2.1 Time-warped search

Let us next introduce a reformulation of the dynamic programming algorithm for W1 that was presented in [4]. The original algorithm uses a three-dimensional array $M[t, p, c]$ where t and p refer to a database and pattern note and c is a counter that limits the search window. Instead of this, we use a two-dimensional array $M[t, p]$ and go through all possible values of c.

```
for p = m → 1 do
  for t = n → 1 do
    if p = m then
      M[t,p] = T[t].score
    else
      b = −∞
      for c = 1 → min(w, n − t) do
        d1 = T[t + c].pitch − T[t].pitch
        d2 = P[p + 1].pitch − P[p].pitch
        if d1 = d2 then
          b = max(b, M[t + c, p + 1] + T[t].score)
        end if
      end for
      M[t,p] = b
    end if
  end for
end for
```

This algorithm can be used to calculate the list S because $S[k] = M[k, 1]$ for each k. The time complexity is still $O(nmw)$, but next we will show how to reduce it to $O(nm)$. The idea is to remove the innermost loop and find the best value more efficiently. To achieve this, we maintain a deque of maximum values for each possible pitch and use a standard sliding window maximum algorithm. Note that we assume that the set of possible pitches is constant. This is a very natural assumption when we are working with pitch values in semitones.

Consider the situation where p and t have been fixed and $p < m$. We are looking for maximum value $M[t', p + 1]$ where $t < t' \leq t+w$ with the constraint that $T[t'].pitch = T[t].pitch + P[p + 1].pitch - P[p].pitch$. Now, for each pitch v, we maintain a deque $D[v]$ containing pairs whose first element is the value $M[t', p + 1]$ and second element is the position t'. The last pair in the deque always corresponds to the maximum value in the window.

We update the deques as follows. When the value t has changed, we always add the pair $(M[t+1, p+1], t+1)$ to the front of deque $D[T[t + 1].pitch]$ and remove the pair $(M[t + w + 1, p + 1], t + w + 1)$ from the tail of deque $D[T[t + w + 1].pitch]$. Before adding the new pair, we remove all pairs from the front of the deque whose value is smaller than the new value. When removing the pair, we require that the second element equals the current position (i.e. the pair has not been removed earlier).

The amortized time complexity for finding all the maximum values for fixed p is $O(n)$ because every value is added and removed to a deque only once. Thus the time complexity of the whole algorithm is $O(nm)$.

The following pseudocode clarifies the structure of the $O(nm)$ algorithm. The array D contains a deque for each possible note, and the deque operations work as described above. Note that the operation $remove$ only takes the second element of the pair to be removed.

```
for p = m → 1 do
  D.clear()
  for t = n → 1 do
    if p = m then
      M[t,p] = T[t].score
    else
      if t + 1 ≤ n then
        D[T[t+1].pitch].add((M[t+1, p+1], t+1))
      end if
      if t + w + 1 ≤ n then
        D[T[t + w + 1].pitch].remove(t + w + 1)
      end if
      v = T[t].pitch + P[p + 1].pitch − P[p].pitch
      if D[v].empty() then
        M[t,p] = −∞
      else
        M[t,p] = D[v].maximum() + T[t].score
      end if
    end if
  end for
end for
```

Note that if we use a symbolic database and every note score is 1, the deques are not needed. In this case it is sufficient to store, for each pitch v, the most recent position $R[v]$, for which $M[t', p+1] = n - p$. If $R[v] \leq t + w$, then $M[t, p]$ equals $n - p + 1$ and otherwise $-\infty$.

2.2 Time-scaled search

The algorithm described in Section 2.1 can also be used for time-scaled search. The only difference is that the size of the window is not constant. Instead of this, the size of the window depends on the position in the pattern and the time values of the notes. However, the sliding window maximum algorithm can still be used and the time complexity of the algorithm remains $O(nm)$ when the range of the scaling factor is fixed.

Let α be the scaling factor and ϵ be the jittering tolerance. In this case, pattern notes $P[p]$ and $P[p+1]$ can be matched to database notes $T[t]$ and $T[t']$ only if $(P[p+1].time - P[p].time)\alpha/\epsilon \leq T[t'].time - T[t].time$ and $(P[p+1].time - P[p].time)\alpha\epsilon \geq T[t'].time - T[t].time$. In other words, we are looking for t' values for which $t + w_1 \leq t' \leq t + w_2$, $T[t + w_1]$ is the first note that fulfils the first condition and $T[t + w_2]$ is the last note that fulfils the second condition.

We can combine the parameters w_1 and w_2 with the algorithm presented in Section 2.1 as follows. Consider the situation for fixed p. First, we set $w_1 = w_2 = n$. When t changes, we decrease w_1 as long as $(P[p+1].time - P[p].time)\alpha/\epsilon \leq T[w_1 - 1].time - T[t].time$. Then, we decrease w_2 as long as $(P[p+1].time - P[p].time)\alpha\epsilon \leq T[w_2 - 1].time - T[t].time$. When decreasing w_1, we add new values to the deque, and when decreasing w_2, we remove old values. Even if the deque operations are more irregular than in the time-warped search, each value is added and removed only once and the time complexity remains $O(nm)$.

In practice, the correct α value is unknown and several searches have to be performed. Thus, the time complexity becomes $O(nms)$ where s is the number of searches. In practice, however, the number of scaling factors that needs to be tested is small because the duration of a single note is typically at most some seconds.

3. EXPERIMENT

In this section, we present the results of our experiment where we searched for themes within audio material using the algorithms described in Section 2. We consider two scenarios: (1) finding the location of the given theme in a single audio file, and (2) finding the audio file which contains the given theme. Moreover, we analyse the performance of the $O(nmw)$ and $O(nm)$ algorithms in practice.

3.1 Material

The material used in the experiment, shown in Table 1, consists of Tchaikovsky's six symphonies.

We created the audio database from Deutsche Grammophon recordings found under the catalogue id 423 504-2. We converted the material from CD tracks into mono WAV files with a sample rate 44,100 Hz. Each audio file contains a single movement of a symphony. The total duration of the audio is 4 hours and 22 minutes.

Work	Movements	Total length	Themes
Symphony 1	4	44 min	10
Symphony 2	4	35 min	11
Symphony 3	5	46 min	15
Symphony 4	4	42 min	13
Symphony 5	4	49 min	15
Symphony 6	4	45 min	11
Total	25	262 min	75

Table 1. The material used in the experiment.

The themes to be searched for were taken from the book *A Dictionary of Musical Themes* [1]. The book contains 75 themes from Tchaikovsky's symphonies and we used all of them in the experiment. Each theme has an identifier in the form of TXXX where X is a digit. The themes are also available online [1] as MIDI files which we automatically converted to simple symbolic notation. In addition, for each theme, we marked the locations where it appears in the audio material to allow automatic evaluation.

The duration of a theme ranges from about 5 seconds to about 25 seconds. We kept all of the themes unchanged with two exceptions. First, theme T231 actually consists of two themes from which the second one is very short. They are indexed separately online, and we have removed the second theme from our material. Second, there is an error in the notation of theme T243 both in the book and online: only the first two triplets are marked and the other notes are too slow. We corrected this in our material.

3.2 Audio processing

Our algorithms require that the database is represented as a list of note events. However, there is no direct way to convert an audio file into symbolic notation. Therefore we estimated the note events using the discrete Fourier transform. In general, strong notes in the music can be observed as strong frequencies in the audio signal.

We used standard techniques for processing the audio signal: we divided each audio file into frames of 4,096 samples. After this, we calculated a frequency spectrum for each frame using the discrete Fourier transform. We assumed that the frequency of A4 is 440 Hz and calculated the amplitude of its frequency in the spectrum for each note from G3 to B5. The selected note range encompasses a large part of the melodies in orchestral music and avoids very low and very high notes. All the amplitudes were normalized and estimated using linear interpolation.

We built three databases of note events, which are summarized in Table 2. The value k refers to the amount of note events produced from a single audio file. The values $\min k$ and $\max k$ are the minimum and maximum amounts

[1] `http://www.multimedialibrary.com/barlow/`

Database	$\min k$	$\max k$	n
D1	107,793	353,249	4,948,386
D2	33,830	107,914	1,541,098
D3	37,510	122,703	1,732,997

Table 2. The number of note events in the databases.

of note events in a file, and the value n denotes the total number of note events in the database.

Database D1 contains a note event for each note in each frame, and the score of the note event is the amplitude of the note in the spectrum. In this case the database includes a lot of note events with low scores that are unlikely to be a part of a theme occurrence.

Databases D2 and D3 are subsets of D1, and the aim is to locate potential note candidates from the full spectrum. Database D2 contains the notes whose amplitudes are peak amplitudes (i.e. the note with pitch p is selected only if notes with pitches $p-1$ and $p+1$ have a weaker amplitude). Database D3 contains the 10 strongest notes in terms of the amplitude for each frame. A value of 10 is selected so that the sizes of D2 and D3 are nearly equal to each other.

This estimation of note events resembles salience value calculation in automatic melody extraction (for a review, see [9]). However, salience values are usually calculated as combinations of the harmonics of the note. We experimented with various ways to use higher harmonics, but to our surprise, using only the fundamental frequency produced the best results with this material.

3.3 Algorithms

We implemented the audio processing method described in Section 3.2 and the time-warped and time-scaled algorithms described in Sections 2.1 and 2.2 by using C++ and FFTW library. For the rest of the paper, we call the time-warped algorithm W and the time-scaled algorithm S.

When using algorithm W, we had to choose the window length w. In this experiment, w has to be relatively high because there can be a large number of note events between two consecutive theme notes in the database. More precisely, the amount of note events per second is about 300 in D1 and about 100 in D2 and D3, and a note in a theme occurrence can have a duration of several seconds.

In algorithm S there are several parameters. First, the scaling factors α have to be specified. In this experiment we assumed that the duration of theme is between 5 and 25 seconds and tested α values for which the time between the first and last note in the theme is $5, 6, 7, \ldots, 25$ seconds. The jittering tolerance ϵ was more difficult to choose; in orchestral music the rhythm is usually quite exact, thus indicating a small value to be the most suitable option.

Moreover, we had to choose how the occurrences of the themes were reported. For this purpose we used the parameter δ, a real number in the range $[0, 1]$. All occurrences with a score of at least δb were reported, where b is the best score. The lower the δ is, the more results are produced. However, a small δ would result in a large number of false positives. Finally, we required that the interval between two reported occurrences is at least 5 seconds. This was done to prevent one occurrence from being reported multiple times.

Database	Metric	W	S
D1	Themes found	29	34
	Total results	76	77
	Precision	0.38	0.44
D2	Themes found	30	29
	Total results	76	77
	Precision	0.39	0.38
D3	Themes found	31	33
	Total results	76	77
	Precision	0.41	0.43

Table 3. The best results of the algorithms. For W, parameter w is 500 in D1 and 150 in D2 and D3, and for S, parameter ϵ is 1.50

$w \setminus \delta$	1.00	0.99	0.95	0.90	0.75
100	12	15	28	46	69
	0.16	0.14	0.09	0.05	0.01
200	22	25	35	52	70
	0.29	0.24	0.12	0.06	0.01
500	29	34	51	64	71
	0.38	0.28	0.09	0.03	0.01
1000	23	32	61	71	74
	0.24	0.15	0.04	0.02	0.01
2000	22	41	72	73	74
	0.07	0.05	0.02	0.01	0.01

Table 4. The results of W on database D1 when w and δ change. Themes found and precision levels are reported.

3.4 File search

In the first part of the experiment, we searched for each theme in the file where it appears. We calculated three measures: the number of themes located correctly, the total number of occurrences reported and the search precision as the ratio of previous two values. We considered that the theme was found if the difference between one of the occurrences reported by the algorithm and one of the occurrences in the ground truth was less than 5 seconds.

Table 3 shows the best results of the algorithms. The parameter δ is 1.00 in each case. For algorithm W, parameter w is 500 in D1 and 150 in D2 and D3. For algorithm S, parameter ϵ is 1.50. Interestingly, the best results were achieved in D1 which contains a note event for each possible note in each audio frame. The results of S were in general somewhat better than the results of W.

Note that in some cases there can be several occurrences even if δ is 1.00. For example, when using W and D1, the number of occurrences is 76 instead of 75. In those cases all normalized note scores in the occurrences are 1, thus the total score is m where m is the number of notes.

$\epsilon \setminus \delta$	1.00	0.99	0.95	0.90	0.75
1.10	25	29	37	51	70
	0.33	0.27	0.13	0.06	0.02
1.20	30	34	43	53	71
	0.40	0.33	0.14	0.06	0.02
1.33	33	38	46	58	71
	0.44	0.36	0.14	0.06	0.02
1.50	34	37	49	60	72
	0.44	0.36	0.13	0.05	0.01
2.00	30	37	49	63	73
	0.36	0.25	0.07	0.03	0.01

Table 5. The results of S on database D1 when ϵ and δ change. Themes found and precision levels are reported.

Table 4 shows the results of W in D1 with different settings of parameters w and δ. The optimal value for w is approximately 500. Finally, Table 5 shows the results of S in D1 with different settings of parameters ϵ and δ change. The optimal value for ϵ is approximately 1.50. Using larger δ values, S would still find themes accurately in some situations. For example, when $\delta = 0.99$ and $\epsilon = 1.33$, S found 38 themes with a precision level of 0.36.

The database used in the experiment is challenging in general; finding some of the themes requires a moderate amount of work even for an experienced human listener. An interesting phenomenon is that the results of the algorithms varied very strongly depending on the symphony. For example, S was able to find 9 out of 10 themes from Symphony 1, but only 1 out of 11 themes from Symphony 6. A possible explanation for this is that the themes in Tchaikovsky's later works are more subtle.

3.5 Database search

In the second part of the experiment, we searched for each theme in the database D1. This resembles the query-by-humming setting [3]; however, our queries are exact (for example, created by a musician) and the database is atypical because it contains a small number of files, each having more than 100,000 note events.

The search was conducted as follows. For each file in the database, we calculated the maximum score for an occurrence of the theme. Having done this, we constructed a list of files that were sorted in decreasing order according to their scores. Following the usual convention, we concentrated on the rank of the correct file (the file where the theme actually appears) in the sorted list. For example, a rank of 5 means that the correct file has the 5th highest score in the search results.

Figures 1 and 2 show the distribution of ranks using W and S in D1 with the same parameters as in Table 3. There were 18 and 22 themes with a rank of 1, and 26 and 31 themes, respectively, with a rank of 1–3. Somewhat surprisingly, in this material locating the theme in the whole database was not much more difficult than locating the theme in the correct file.

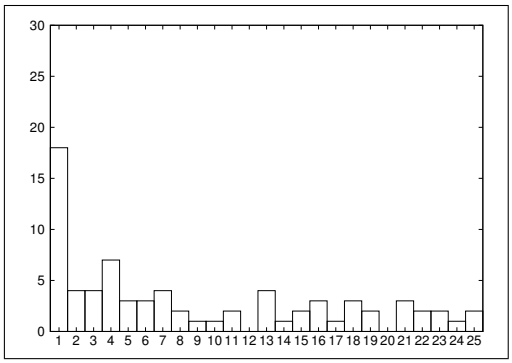

Figure 1. Distribution of correct file ranks using algorithm W and database D1.

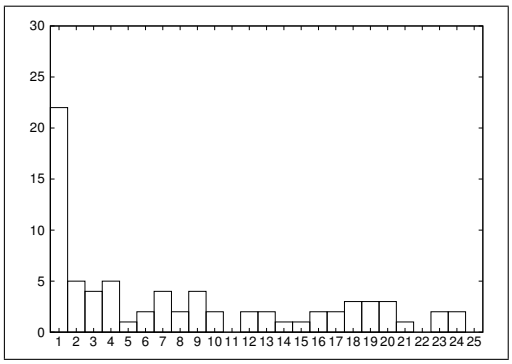

Figure 2. Distribution of correct file ranks using algorithm S and database D1.

3.6 Performance

Finally, we analyzed the practical performance of the algorithms. We ran the tests on an Intel Core 2 Duo Processor with a clock rate of 3.16 GHz.

Figure 3 shows the running times of three algorithms: S is the $O(nms)$ implementation of the time-scaled algorithm, and W and W' are $O(nm)$ and $O(nmw)$ implementations of the time-warped algorithm. We searched for theme T252 with 27 notes from each audio file in database D1 using parameters $\delta = 1.00$, $\epsilon = 1.50$ and $w = 500$, as shown in Table 3. As discussed previously, the variable k denotes the number of note events in the audio file.

As expected, W clearly outperformed the other two al-

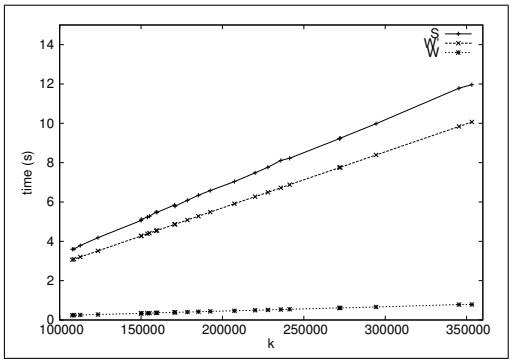

Figure 3. Running times of S, W' and W using the parameters shown in Table 3.

gorithms: it was more than 10 times faster than both W' and S. Furthermore, S was almost as fast as W' although it performed twenty separate searches for each query.

4. CONCLUSIONS

In this paper we presented a new time-warp invariant polyphonic music search algorithm that works in $O(nm)$ time. Unlike the earlier algorithms [4, 6] with time complexities $O(n \log nmw)$ and $O(nmw)$, the running time of our algorithm does not depend on the window size w, and it can be used with arbitrarily large windows.

In addition, our algorithm can also be used for time-scale invariant search. We introduced a new approximate time-scale invariant schema which allows limited jittering within the scaled note durations. In this case, the time complexity of our algorithm is $O(nms)$ where s is the number of possible scalings. In typical queries s is small.

We used our algorithms to search for musical themes in Tchaikovsky's symphonies. We estimated the note events using amplitudes of frequencies in the audio files. Considering the challenging nature of orchestral music input, the results of the algorithms were promising, and the experiment also showed that the modified dynamic programming algorithm is efficient in practice.

Our future plan is to further develop the database construction. Limiting the number of notes in the database could make the search both more accurate and more efficient. However, within the context of a frame, it is difficult to decide which note pitches should be included; in our experiment, the best results were achieved when all pitches were included. Therefore, another approach would be to remove entire frames which are probably not needed in the search.

5. ACKNOWLEDGEMENTS

This work has been supported by the Helsinki Doctoral Programme in Computer Science and the Academy of Finland (grant number 118653).

6. REFERENCES

[1] H. Barlow and S. Morgenstern: *A Dictionary of Musical Themes*, Crown Publishers, New York, 1948.

[2] A. Duda, A. Nürnberger and S. Stober: "Towards query by singing/humming on audio databases," *Proceedings of the 8th International Conference on Music Information Retrieval*, 331–334, 2007.

[3] A. Ghias et al: "Query by humming: musical information retrieval in an audio database," *Proceedings of ACM Multimedia 95*, 231–236, 1995.

[4] M. Laitinen and K. Lemström: "Dynamic programming in transposition and time-warp invariant polyphonic content-based music retrieval," *Proceedings of the 12th International Society for Music Information Retrieval Conference*, 369–374, 2011.

[5] K. Lemström: "Towards more robust geometric content-based music retrieval," *Proceedings of the 11th International Society for Music Information Retrieval Conference*, 577–582, 2011.

[6] K. Lemström and M. Laitinen: "Transposition and time-warp invariant geometric music retrieval algorithms," *Proceedings of the 3rd International Workshop on Advances in Music Information Research*, 1–6, 2011.

[7] A. Lubiw and L. Tanur: "Pattern matching in polyphonic music as a weighted geometric translation problem," *Proceedings of the 5th International Society for Music Information Retrieval Conference*, 289–296, 2004.

[8] D. Meredith, G. Wiggins and K. Lemström: "Pattern induction and matching in polyphonic music and other multi-dimensional datasets," *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics*, 61–66, 2001.

[9] G. Poliner et al: "Melody transcription from music audio: approaches and evaluation," *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4), 1247–1256, 2007.

[10] C.A. Romming and E. Selfridge-Field: "Algorithms for polyphonic music retrieval: The hausdorff metric and geometric hashing," *Proceedings of the 8th International Society for Music Information Retrieval Conference*, 457–462, 2007.

[11] M. Ryynänen and A. Klapuri: "Query by humming of midi and audio using locality sensitive hashing," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2249–2252, 2008.

[12] J. Salamon, J. Serrà and E. Gómez: "Tonal representations for music retrieval: from version identification to query-by-humming," *International Journal of Multimedia Information Retrieval*, 2(1), 45–58, 2013

[13] E. Ukkonen, K. Lemström and V. Mäkinen: "Geometric algorithms for transposition invariant content-based music retrieval," *Proceedings of the 4th International Society for Music Information Retrieval Conference*, 193–199, 2003.