

# Effective Retrieval of Polyphonic Audio with Polyphonic Symbolic Queries

Iman S. H. Suyoto

Alexandra L. Uitdenbogerd

Falk Scholer

School of Computer Science and Information Technology, RMIT  
GPO Box 2476V, Melbourne, Victoria 3001, Australia  
firstname.lastname@rmit.edu.au

## ABSTRACT

Accurately finding audio recordings in response to symbolic queries is one of the key challenges in the field of music information retrieval. Pitch is one of the main features of music; in this paper we propose and evaluate approaches for using pitch information in polyphonic symbolic queries to retrieve full tracks of audio recordings. The audio data is first converted into symbolic data, using an automated transcription process. This is a noisy process, adding up to three times as many notes to the transcription than are actually present. Nevertheless, recordings can be accurately retrieved by manually-constructed queries (either in full or truncated) using the longest common subsequence algorithm (and a sliding window if the queries are truncated). Precision at 1 of about 80% was achieved, and around 85% of queries return correct answers in the top 10 from a collection of 1808 recordings. Truncated queries are as effective as untruncated queries for retrieving correct answers in the first rank position. Thus, the burden on users is reduced as they only need to produce a small fraction of a song as a query.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*query formulation, retrieval models, search process*; H.5.5 [Information Interfaces and Presentation]: Sound and Music Computing—*methodologies and techniques, signal analysis, synthesis, and processing*

## General Terms

Algorithms, experimentation, measurement, performance

## Keywords

Audio, dynamic programming, longest common subsequence, music information retrieval, polyphony, symbolic, transcription

## 1. INTRODUCTION

In the field of music information retrieval, researchers are developing methods of finding and retrieving music in a variety of ways.

Some methods use queries that represent part of the musical content of the target music. If the queries precisely specify the musical notes, they are called *symbolic*. An alternative that is often researched is the matching of a sung query, that is *audio*, against a collection of music. In this case, the query is not symbolic but *monophonic* (one note at a time) audio; and the collection being matched against is typically symbolic [19, 21, 25, 37] and sometimes *polyphonic*, such as in Dannenberg et al. [9].

Converting a monophonic audio recording into a sequence of notes or symbols is reasonably accurate. For example, for singing, a good transcriber achieves about 80% accuracy in terms of note identification [6]. However, doing so with a polyphonic recording results in many errors. This is due to the confounding factors of timbre (the sound of the instruments),<sup>1</sup> and the acoustic environment in which the music is situated, both of which contribute harmonics that may be interpreted erroneously as notes. For example, the transcriber used in our work, TS-AudioToMidi, adds approximately three times as many notes to the symbolic representation of a piece of music than are actually present (see Section 4 for the settings we used in our experiment). Furthermore, not all actual notes are present in the symbolic representation.

In our work we consider the case of retrieving from an audio collection of music using a symbolic polyphonic query, where the polyphonic query is based on a manually-constructed MIDI rendition of the music that is to be retrieved. In this paper we demonstrate techniques that are successful for queries that are either the full musical work being retrieved, or only a fraction of its length.

While this work could be considered a stepping stone to better query-by-score systems for audio collections, the query type is useful in its own right. For example, a user may have a MIDI file of a piece of music and wish to retrieve real performances of the same piece.

In our initial approach to solving this problem [40], we used a relative-pitch standardisation (see Section 3.2) combined with global alignment and local alignment. However, only a minority of the 100-symbol queries used could rank correct answers highly. Shorter queries were even less effective. Using untruncated queries was the most effective approach, although less than 40% of queries could retrieve the correct answers in the first rank position.

Our new approaches use the longest common subsequence (LCS) algorithm, which identifies the number of symbols common in two

<sup>1</sup>Timbre is described in the physics literature, e.g. Giancoli [14, page 334].

strings and appearing in the same sequence, without adjacency requirement [17, pages 227–228]. An additional process is required for truncated queries: a candidate answer is divided into substrings, each of which generates a candidate score resulting from the use of the LCS algorithm between the query and the substring. The strings in this case are in an absolute pitch form. This algorithm is described in detail in Section 3. Experiments and results are presented in Section 4, and discussed in Section 5. We draw conclusions and suggest future research direction in Section 6.

To give some perspective on our new techniques, our previous approach with full queries resulted in a mean precision at 1 (see Section 4 for the definition of precision) of less than 40%, with mean precision at 10 of about 10%, while our current technique for the task achieves a mean precision at 1 of more than 80% with a mean precision at 10 of about 16%. Our previous approach with 100-symbol queries resulted in mean precision at 1 of less than 35%, with mean precision at 10 of less than 10% [40]. On the other hand, our current technique achieves a mean precision at 1 of 80% for 100-symbol queries, with a mean precision at 10 of 17%, which is about 88% of the theoretical maximum for the test collection (see Table 5).

Accompanying materials for this paper can be found at <http://mirt.cs.rmit.edu.au/pubs/sd>.

## 2. RELATED WORK

Our full-query task is closely related to score-to-audio alignment. Both aim to match an audio file with its symbolic equivalent, although the focus differs slightly. The score-to-audio alignment task typically has its effectiveness measured in a function of number of correctly or incorrectly identified notes [27, 33, 36, 38], whereas ours is a retrieval task and therefore more concerned with the ability to discriminate correct answers from incorrect ones and ranking the correct ones highly (the actual number of correctly identified notes is not a matter of concern). Also, the score-to-audio alignment task is concerned with structural differences contained within a piece, while a retrieval task requires approximate matches to be ranked highly. Nevertheless, similar techniques can be used for both tasks.

There has been other work performing the full-query task. Pickens et al. [30] attempted a similar problem to ours. They implemented a harmonic modelling approach using polyphonic audio queries on a collection of polyphonic symbolic pieces. Some of the queries were recordings of human performances, and some were synthesised from MIDI to obtain audio. For the 3 collections shown, the highest mean average precision (MAP; see Section 4 for its definition) was 0.479. In our work, we use an audio collection with symbolic queries. The audio files are transcribed to obtain symbolic sequences, which in turn are matched with symbolic queries. Pickens et al. made use of statistical patterns of chords in music, while our approach does not.

Hu et al. [20] also investigated using polyphonic audio queries to retrieve polyphonic symbolic collection. Similarly to our approach, they also use chroma.<sup>2</sup> They used a collection of 259 MIDI files and 51 audio queries, all are the Beatles’ songs. The MIDI files

<sup>2</sup>The term “chroma” and “pitch classes” are often used interchangeably. See, e.g. Müller et al. [26]. “Chroma” is usually used in the audio signal analysis context, while “pitch classes” is usually used in music theory and symbolic contexts.

were synthesised to generate audio files. The chroma were then extracted from the audio. The result of applying their method against their collection was mean precision at 1 of 0.49. In our work, we extract pitch classes directly from MIDI files. The audio transcriptions in our work contain extraneous notes (see Section 3.1), making the retrieval task considerably difficult. Our collection is also much larger (see Section 4).

A related problem has previously been considered by Shalev-Shwartz et al. [35]. In contrast with our work, their symbolic queries were monophonic, while ours are polyphonic. Also, their collection contained 832 one-minute opera performances with orchestra accompaniment, whereas we have 1 808 untruncated pieces of classical music performances. In their work, polyphonic audio and monophonic symbolic queries were aligned using a probabilistic approach with temporal and spectral features being used for the process. With 25-second queries, average precision of 95% was achieved.

The Shazam system<sup>3</sup> can retrieve specific recordings matching the query, exactly the same version, in spite of audio quality difference [44]. This is achieved by using audio signatures. However, cover versions of the query cannot be normally found. Early work on cover version matching used long-term structure, such as variation in loudness [13] or timbral texture [1, 2]. However, all of these approaches were evaluated with collections of less than 100 pieces. Techniques for matching MIDI files with monophonic hummed queries have been more successful [9, 19, 21, 25, 37]. More recent attempts at cover version retrieval have used audio to audio matching. Marolt used a melody-based representation extracted from audio [24]. The collection contained 1 820 pieces with 36 of them used as the queries. Using a combination of melody-based and chroma representations yielded the highest effectiveness of 27% of hits in the top 5 returned answers. A recent approach by Gómez and Herrera [15] using chroma achieved an accuracy of about 50% with a 90-piece collection.

The LCS algorithm has previously been used for monophonic music matching [16]. However, the algorithm was modified in order to handle variations in speed and inaccuracies in rhythm. A subset of the collection was randomly chosen and manipulated to form the queries. The manipulations were done in various ways: stretching the songs; shortening and lengthening randomly chosen notes to simulate inaccuracies in rhythm; and insertion and deletion of random notes. The best technique achieved almost 90% of correct answers ranked top 5. It was when the queries were chunks of original songs with some notes randomly inserted. However, as our results show, it is not necessary to modify the LCS algorithm for the efficacy of our tasks.

The problem of matching polyphonic symbolic queries with a polyphonic symbolic collection is mostly solved [29, 31, 41, 43]. However, a technique that is highly effective for matching polyphonic symbolic queries with polyphonic symbolic collections does not work well for polyphonic audio collections, as our earlier work shows [40].

We are not aware of any other work that examines exactly the same problem as ours, aside from our own previous work [40]. We therefore use the best results from our previous approaches to this re-

<sup>3</sup>See <http://www.shazam.com>.

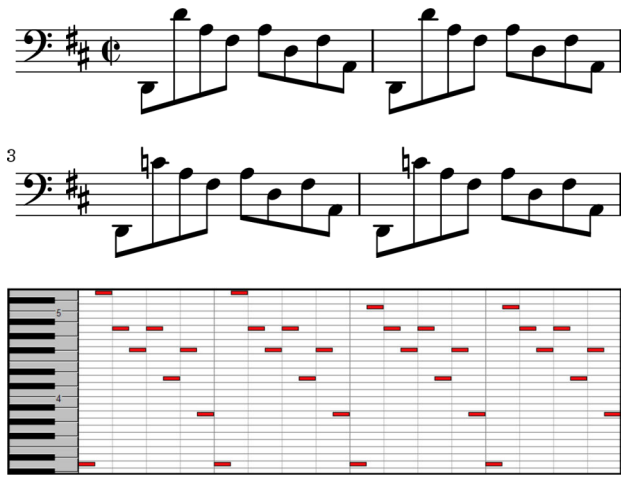


Figure 1: The first four bars of J. S. Bach’s BWV 1010 Prelude. The original composition is written in E $\flat$  major. In this figure, it is transposed to D major to match the transcription (see Figure 2) of a performance using the Baroque tuning (A = 415 Hz).

trieval task as the state-of-the-art baseline in our experiments (fully described in Section 4).

### 3. MATCHING

The matching process involves three stages: transcription (of audio to symbolic data), standardisation, and alignment. They are discussed in the following subsections.

#### 3.1 Transcription

For automatic music transcription, we use the software package TS-AudioToMidi 3.30.<sup>4</sup> This produces transcriptions in the Standard MIDI file format.<sup>5</sup> The transcriptions contain noise in the form of extraneous notes; this normally happens, particularly when the transcribed audio is produced by instruments whose timbre contains many harmonic components. These harmonic components are also more difficult to identify in polyphonic music. In monophonic music, the fundamental frequency sounding at any time may be selected as “the note,” whereas in polyphonic music the distinction between the harmonic frequency of a note and a fundamental frequency of another note is often unclear. Even with monophonic music, other factors such as reverberation may cause the transcription process to introduce extra notes. Moreover, the problem of finding melody lines is only partially solved [11, 22, 23, 28, 32].

Figure 1 shows an excerpt of one of the tracks used in our experiment, J. S. Bach’s BWV 1010 Prelude performed by Phoebe Carrai. To illustrate the noise problem in automatic music transcription, Figure 2 shows the transcription result, which contains many extraneous notes.

#### 3.2 Standardisation

After the transcription results are obtained, they are transformed into strings in this *standardisation* stage. This enables approximate string matching techniques to be applied against them.

<sup>4</sup>See <http://audioto.com/eng/aud2midi.htm>.

<sup>5</sup>See <http://www.midi.org/about-midi/abtmidi2.shtml>.

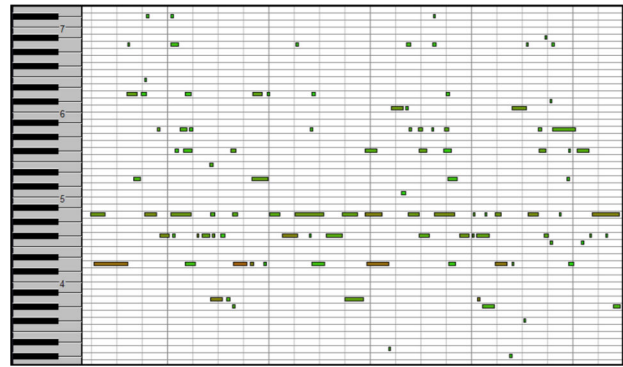


Figure 2: The transcription result of the first four bars of J. S. Bach’s BWV 1010 Prelude performed by Phoebe Carrai.



Figure 3: A melody from “A Wish for the Better” by ade ishs. It is represented as “G A E E D C D G G”.



Figure 4: Chords from “Little Butterfly” by ade ishs. It is represented as “B $\flat$  D F B $\flat$  D F B $\flat$  D F B $\flat$  D F B $\flat$  C E $\flat$  G B $\flat$  C E $\flat$  G B $\flat$  C E $\flat$  B $\flat$  D F”.

Our previous work [39, 40, 43] used the directed modulo-12 standardisation. This belongs to the class of representations that encode pitches as relative values (included in this class are contour standardisations) [42, pages 78–86].

This work uses the pitch class of notes to represent them. A pitch is encoded as an absolute value by using its own pitch name, without the octave. Rest notes are ignored. For example, the melody shown in Figure 3 is represented as “G A E E D C D G G”.

A chord is represented as if it were an arpeggio, by sorting the notes in ascending order by *actual* pitch (the octave is taken into account for sorting). For example, in Figure 4, the first chord consists of B $\flat$ 3, D4, and F4, so it is represented as “B $\flat$  D F”. The whole score is represented as “B $\flat$  D F B $\flat$  D F B $\flat$  D F B $\flat$  D F B $\flat$  C E $\flat$  G B $\flat$  C E $\flat$  G B $\flat$  C E $\flat$  B $\flat$  D F”.

The disadvantage of this standardisation is that the string representation is not transposition-invariant. Therefore, in the alignment process, transposition is incorporated as part of matching. This issue is addressed in more detail in Section 3.3. However, as we shall see in Section 4, this approach supports much higher retrieval effectiveness compared to relative-pitch standardisation [40].

#### 3.3 Alignment

The alignment step involves the use of dynamic programming. Such a technique has been shown to be useful for various retrieval

---

**Algorithm 1** The algorithm to calculate  $S(q, a)$ .  $t(q, s)$  is a function transposing  $q$  by  $s$  semitones ( $s \in \{0, 1, 2, \dots, 11\}$ ).  $\text{LCS}(T, a)$  is the LCS score between  $T$  and  $a$ .

---

**Require:** query  $q$ , answer  $a$

```

 $s \leftarrow 0$ 
for  $k = 0 \dots 11$  do
   $q' \leftarrow t(q, k)$ 
   $s' \leftarrow \text{LCS}(q', a)$ 
  if  $s' > s$  then
     $s \leftarrow s'$ 
  end if
end for
return  $s$ 

```

---

tasks, including symbolic music matching [9, 39, 43] and audio matching [1, 13].

We consider two problems: retrieving a recording using its *full* symbolic equivalent, and retrieving a recording using its *truncated* symbolic equivalent. Our preliminary experiments showed that the two tasks could not be handled using the same alignment procedure. Therefore, we discuss the approaches for both tasks; retrieval using full queries is discussed in Section 3.3.1, and retrieval using truncated queries is discussed in Section 3.3.2.

### 3.3.1 Retrieval using full symbolic versions

In this method, the longest common subsequence (LCS) [17, pages 227–228] score between a query and an answer is used as a candidate score. The query is then transposed by one semitone. This is done 11 times. For example, if the query is “C E G C”, it is transposed to “C# F G# C#”, and then to “D F# A D”, up to “B D# F# B”. The answer remains untransposed. The transposition causes the answer to be scanned 12 times, resulting in 12 candidate scores. The score with maximum similarity is picked as the final score for the answer. Mathematically, if  $S(q, a)$  is the similarity between query  $q$  and answer  $a$ :

$$S(q, a) = \max_s (\text{LCS}(t(q, s), a)) \quad (1)$$

where  $t(q, s)$  is a function transposing  $q$  by  $s$  semitones ( $s \in \{0, 1, 2, \dots, 11\}$ ), and  $\text{LCS}(T, a)$  is the LCS score between  $T$  and  $a$ . The full (unoptimised) algorithm is specified in Algorithm 1.

This method assumes that the queries are proportional to the correct answers in terms of length expressed as the number of symbols. Theoretically, a problem with using  $S(q, a)$  as a similarity measurement is that longer songs that may not be correct answers have a higher chance to yield (almost) maximum scores. This implies that  $S(q, a)$  is not sufficient to discriminate between a relevant answer and an irrelevant one. This is shown in our experimental results in Section 4.1 and discussed further in Section 5. We hypothesise that the scores should be normalised with a function of answer length to solve the problem. We attempted various normalised functions, and we found the following one to be highly effective:

$$S_y(q, a, y) = \frac{S(q, a)}{\log_e^y |a|} \quad (2)$$

where  $y$  is a tunable parameter. The experimental results for this function are also presented in Section 4.1.

---

**Algorithm 2** The algorithm to calculate  $S_d(q, a, d)$ .  $t(q, s)$  is a function transposing  $q$  by  $s$  semitones ( $s \in \{0, 1, 2, \dots, 11\}$ ).  $\text{LCS}(T, a)$  is the LCS score between  $T$  and  $a$ . We use 0 as the base index.

---

**Require:** query  $q$ , answer  $a$ , parameter  $d$

```

 $W \leftarrow \lceil 2d|q| \rceil$ 
 $s \leftarrow 0$ 
for  $k = 0 \dots 11$  do
   $L \leftarrow 0$ 
   $q' \leftarrow t(q, k)$ 
  while  $L + W < |a|$  do
     $s' \leftarrow \text{LCS}(q', a_L \dots a_{L+W})$ 
    if  $s' > s$  then
       $s \leftarrow s'$ 
    end if
     $L \leftarrow L + \lceil d \rceil$ 
  end while
end for
return  $s$ 

```

---

### 3.3.2 Retrieval using truncated symbolic versions

In this task, we simulate the scenario when a user has a query which is only a short part of a full piece. We investigated the techniques presented in Section 3.3.1 in our preliminary experiments for the current task. However, the results showed that the technique is not effective if the symbolic query is much shorter than the desired piece. Instead, we propose the use of a sliding window technique.

To calculate the similarity score for a query  $q$  and a possible answer item  $a$ , we use a sliding window on the candidate answer. We devise this technique because query lengths are not proportional to answer lengths. Therefore, we cannot use the technique explained in Section 3.3.1 as it violates the assumption. Our algorithm uses a window size parameter  $d$ . The window size function  $W$  is:

$$W = \lceil 2d|q| \rceil \quad (3)$$

The actual window size itself is  $W + 1$ . For a string  $z = a_0 \dots a_W$ , the LCS length between  $z$  and  $q$  is calculated, giving a score  $s$ . The window then slides by  $\lceil d \rceil$  positions, so that  $z$  becomes  $a_{\lceil d \rceil} \dots a_{\lceil d \rceil + W}$ . Again, the LCS between  $z$  and  $q$  is calculated, giving a new score  $s'$ . If  $s' > s$ ,  $s$  is updated to the new value,  $s'$ . The window slides again by  $\lceil d \rceil$ , so that  $z$  becomes  $a_{2\lceil d \rceil} \dots a_{2\lceil d \rceil + W}$ . The LCS score between  $z$  and  $q$  is calculated again and assigned to  $s'$ . If  $s' > s$ , the current value of  $s$  is updated to a new value of  $s'$ . This is repeated as long as  $n\lceil d \rceil + W < |a|$ ,  $n$  is a non-negative integer. After that, the query is transposed by one semitone. The whole process is repeated for all keys, that is for transpositions over up to 11 semitones. The score of the alignment between query  $q$  and answer  $a$  with the parameter  $d$  is expressed as  $S_d(q, a, d)$  and the value is the final value of  $s$ . The (unoptimised) algorithm is shown in Algorithm 2.<sup>6</sup>

As an example of how to apply Algorithm 2, suppose we want to align the query  $q = \text{“E A C#”}$  (hence  $|q| = 3$ ) with the answer  $a = \text{“F F C F# D A# D C C C A A# A# G”}$  (hence  $|a| = 14$ ). Let us choose  $d = 1.3$ . This gives  $W = \lceil 2 \times 1.3 \times 3 \rceil = 8$ , hence the window size of 9. The current window is **highlighted**.

<sup>6</sup>The time complexity of this algorithm depends on the time complexity of the algorithm to calculate the LCS length. For two strings of lengths  $m$  and  $n$ , an unoptimised LCS length calculation is  $O(mn)$ . In this case, this algorithm takes  $O(|q|^2|a|)$ . However, much work has been done on optimising LCS calculation. This is outside the scope of this paper and is discussed elsewhere [7].

F F C F# D A# D C C C A A# A# G

The LCS score between “E A C#” and the highlighted substring is 0. This becomes  $s$ . The window then slides by  $\lceil 1.3 \rceil = 2$ :

F F C F# D A# D C C C A A# A# G

The LCS score between “E A C#” and the highlighted substring is 1. The candidate score is now 1 since  $1 > 0$ . The window then slides by 2 again:

F F C F# D A# D C C C A A# A# G

The LCS score between “E A C#” and the highlighted substring is 1. Next, the query is then transposed by one semitone, becoming  $q = \text{“F A# D”}$ . We rewind the window starting index.

F F C F# D A# D C C C A A# A# G

The LCS score between “F A# D” and the highlighted substring is 3. The candidate score is now 3 since  $3 > 1$ . This is repeated until  $q$  is transposed up to 11 times.

## 4. EXPERIMENTS

We are proposing two methods for matching, one for full queries and one for truncated queries. Results of our experiments for the first task are given in Section 4.1, and for the second task in Section 4.2.

Performance of search systems can be measured by evaluating the *precision* of a ranked results list [3, page 75]:  $P \equiv |\mathbf{Rel} \cap \mathbf{Ret}| / |\mathbf{Ret}|$  where  $\mathbf{Rel}$  is the set of relevant answers and  $\mathbf{Ret}$  is the set of retrieved answers.

Since users in the type of music retrieval task that we are investigating are likely to be most interested in items that are returned in the top positions of the answer list, we focus on mean precision at  $N$  retrieved items ( $\langle P_N \rangle$ ),  $N \in \{1, 2, 3, \dots, 20\}$ . We also report mean average precision (MAP), a widely-used metric in information retrieval that summarises performance over a set of queries. MAP calculates the average precision at each relevant item that is retrieved, over a run of queries. Relevant documents that are not retrieved contribute zero to the average [4]. Mathematically, the average precision for a query is defined as  $\frac{1}{|\mathbf{Rel}|} \sum_{N=1}^D P_N B(N)$  where  $D$  is the number of documents in the collection and  $B(N)$  is a binary relevance function for the answer returned at rank  $N$ . MAP is then obtained by averaging the per-query results over the query set.

Our collection contains tunes from the Magnatune classical music collection<sup>7</sup> (as at 28 April 2005) stored as MP3 (MPEG Layer 3)<sup>8</sup> files. It contains multiple versions of some pieces, for example J. S. Bach’s Suite I for Cello Solo (BWV 1007), performed by three different individuals.

<sup>7</sup>See <http://www.magnatune.com/genres/classical/>.

<sup>8</sup>See <http://www.iis.fraunhofer.de/amm/techinf/layer3/>.

**Table 1: The queries and the number of relevant covers in the collection (C). (KS: Kern Scores, MD-1: MuseData [optimised for printing], MD-P: MuseData [optimised for listening], MP: the Mutopia Project.)**

No	Title	Source	C
1	BWV 1011 Courante	KS	3
2	BWV 1011 Sarabande	KS	3
3	Corelli, Tr. Son. Op 1 No 7 C Maj (Mv 1)	KS	1
4	Corelli, Tr. Son. Op 1 No 7 C Maj (Mv 2)	KS	1
5	Corelli, Tr. Son. Op 1 No 7 C Maj (Mv 3)	KS	1
6	Dufay, Adieu Ces Bons Vins De Lannoys	KS	1
7	Joplin, Stoptime Rag	KS	1
8	K 545 Movement 1	KS	1
9	K 545 Movement 2	KS	1
10	K 238	KS	1
11	BWV 870 Fugue	MD-1	1
12	BWV 870 Prelude	MD-1	1
13	BWV 870 Fugue	MD-P	1
14	BWV 870 Prelude	MD-P	1
15	BWV 1007 Allemande	MP	3
16	BWV 1007 Courante	MP	3
17	BWV 1007 Gigue	MP	4
18	BWV 1007 Menuets	MP	3
19	BWV 1007 Prelude	MP	3
20	BWV 1007 Sarabande	MP	3
21	BWV 1010 Allemande	MP	3
22	BWV 1010 Bouree I	MP	3
23	BWV 1010 Bouree II	MP	3
24	BWV 1010 Courante	MP	3
25	BWV 1010 Gigue	MP	3
26	BWV 1010 Prelude	MP	3
27	BWV 1010 Sarabande	MP	3
28	BWV 1042 Adagio	MP	1
29	BWV 1042 Allegro	MP	1
30	BWV 1042 Allegro Assai	MP	1
31	BWV 846 Fugue	MP	1
32	BWV 846 Prelude	MP	1
33	BWV 860 Fugue	MP	1
34	BWV 860 Prelude	MP	1

We used the default setting of TS-AudioToMidi when transcribing the MP3 files. Some files could not be processed, leaving us with 1 808 MIDI files of transcriptions to work with.

For our query set, we gathered the MIDI versions of some of the covers in the Magnatune collection. The sources of the symbolic queries are the Mutopia Project,<sup>9</sup> Kern Scores,<sup>10</sup> and MuseData.<sup>11</sup> In total, we have 34 queries that have relevant answers in the collection. They are specified in Table 1. The same query set is used for both tasks, except that the queries are truncated for the short-query task. See Section 4.2 for more details.

The target pieces in the collection use various instrumentations, ranging from solo piano up to orchestra. Out of the 34 queries we have, only two of them—BWV 1010 Gigue and BWV 1011 Sarabande—are completely monophonic. However, those two

<sup>9</sup>See <http://www.mutopiaproject.org>.

<sup>10</sup>See <http://kern.humdrum.net>.

<sup>11</sup>See <http://www.musedata.org>.

compositions are originally monophonic. Moreover, the challenge given by the noisy transcription process still holds even for those queries, as it produces polyphonic transcriptions, even from monophonic pieces.

BWV 1007 Menuets consist of two parts. In our audio collection, there are three covers and all of them play the two parts in a single file. We obtained the query with both parts separated. They were concatenated to form one single query. A similar case happened with BWV 1010 Bourees, which also consist of two parts, and there are also three covers with all of them playing the two parts as a single track. However, for this one, there were two queries, one for each part (BWV 1010 Bouree I and BWV 1010 Bouree II). These disparities are useful to investigate how robust our  $S_y(q, a, y)$  function performs in such cases.

As the *baseline* benchmark in this work, we use the best results reported in our previous work [40], namely SBDG for the full-query task and SBDL100 for the truncated query task.<sup>12</sup> In brief, both techniques use a relative pitch representation that had initially been filtered to remove high pitch and low velocity notes. The intention was to produce a sequence of notes that represented the bass part of the music, under the assumption that the lowest notes in automatically transcribed music are more likely to be accurate than high ones. SBDG uses global alignment while SBDL100 uses local alignment.

#### 4.1 Experiments with full queries

We experiment with the  $S(q, a)$  similarity function (see Equation 1) and  $S_y(q, a, y)$  (see Equation 2) with  $y \in \{1.0, 1.1, 1.2, \dots, 3.0\}$ . Figure 5 shows the mean precision at  $N$  curves for  $S(q, a)$  and  $S_y(q, a, 2.0)$ . This conforms our hypothesis in Section 3.3.1 that LCS scores without normalisation, that is  $S(q, a)$ , do not have sufficient discriminatory power.  $S_y(q, a, 2.0)$  far outperforms  $S(q, a)$  and even the benchmark SBDG. The MAP values for various similarity measurements are presented in Table 2.  $y$  values in the range from 1.3 to 2.3 give excellent performance, statistically significantly better than the baseline (paired  $t$ -test,  $p < 0.0001$ ).

#### 4.2 Experiments with truncated queries

Our experiments evaluate two core aspects of the new approach: first, we investigate different truncated query lengths to establish the minimum length that is required for effective retrieval. Second, we investigate highly effective values for the parameter  $d$  that controls the window size. We first investigate coarse-grained settings, as discussed in Section 4.2.2, and then fine-tune the value of  $d$ , described in Section 4.2.3.

##### 4.2.1 Varying $|q|$

The average number of symbols across all queries is 814.15 (see Figure 6 for the distribution of query lengths). In this step, we experimented with different query lengths: queries were truncated to 20, 50, and 100 symbols. If a query is shorter than a specified post-truncation length, that query is left untruncated.

Figure 7 shows the mean precision curves for  $|q| \in \{20, 50, 100\}$  and  $d = 1$ . The MAP values are shown in Table 3. It can be seen that lengthening queries yields higher effectiveness. Once the length is 100 and  $d = 1$ , the mean precision at 1 value does not

<sup>12</sup>The results are available at <http://mirt.cs.rmit.edu.au/pubs/sbdg/>.

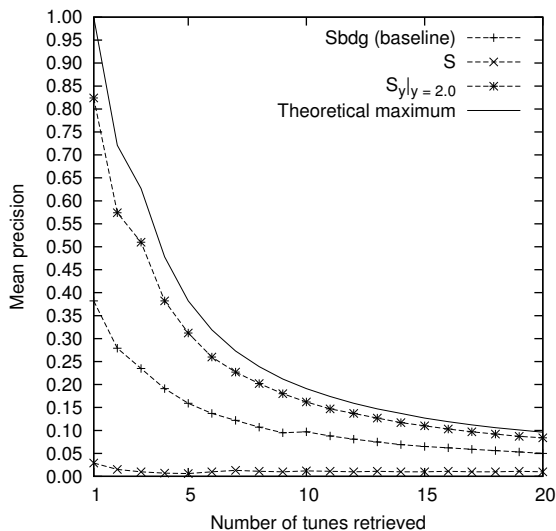


Figure 5: The mean precision at  $N$  ( $\langle P_N \rangle$ ) curves for SBDG (the baseline),  $S(q, a)$ , and  $S_y(q, a, 2.0)$ .

Table 2: The MAP values for SBDG (the baseline),  $S(q, a)$ , and  $S_y(q, a, y); y \in \{1.0, 1.1, 1.2, \dots, 3.0\}$ . The best value is **highlighted**.  $S_y(q, a, y); y \in \{1.0, 1.1, 1.2, \dots, 3.0\}$  is significantly better than the baseline (paired  $t$ -test,  $p < 0.01$ ).

Method	MAP
SBDG	0.374
$S(q, a)$	0.053
$S_y(q, a, 1.0)$	0.688
$S_y(q, a, 1.1)$	0.695
$S_y(q, a, 1.2)$	0.738
$S_y(q, a, 1.3)$	0.782
$S_y(q, a, 1.4)$	0.818
$S_y(q, a, 1.5)$	0.803
$S_y(q, a, 1.6)$	0.802
$S_y(q, a, 1.7)$	0.801
$S_y(q, a, 1.8)$	0.794
$S_y(q, a, 1.9)$	0.817
$S_y(q, a, 2.0)$	<b>0.826</b>
$S_y(q, a, 2.1)$	0.809
$S_y(q, a, 2.2)$	0.790
$S_y(q, a, 2.3)$	0.761
$S_y(q, a, 2.4)$	0.730
$S_y(q, a, 2.5)$	0.708
$S_y(q, a, 2.6)$	0.689
$S_y(q, a, 2.7)$	0.676
$S_y(q, a, 2.8)$	0.655
$S_y(q, a, 2.9)$	0.645
$S_y(q, a, 3.0)$	0.636

differ from that when using full queries with  $S_y(q, a, 2.0)$ . However, using full-length queries with  $S_y(q, a, 2.0)$  is still overall better as the MAP value (0.826) is considerably higher than that of  $S_d(q, a, 1); |q| = 100$  (0.745). Compared to the baseline, using  $S_d(q, a, 1); |q| = 100$  is statistically significantly better (paired  $t$ -test,  $p < 0.0001$ ).

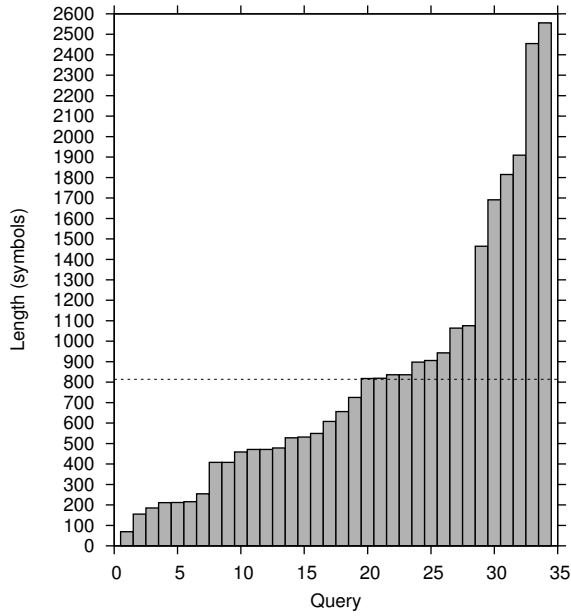


Figure 6: The distribution of query lengths before truncation. The average is indicated by the dashed line.

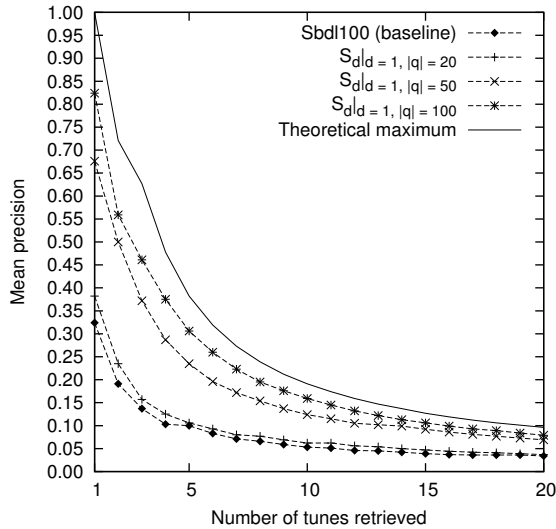


Figure 7: The mean precision at  $N$  ( $\langle P_N \rangle$ ) curves for  $S_d(q,a,d); d=1, |q| \in \{20, 50, 100\}$  and  $S_y(q,a,y)|_{y=2}$ .

#### 4.2.2 Varying $d$

In this step, we experiment with varying the window-size parameter  $d \in \{1, 2, 3, 4, 5\}$ , while holding a fixed value of  $|q| = 100$ . Figure 8 shows the mean precision at  $N$  curves for  $d \in \{1, 2, 3, 4, 5\}$ . The MAP values are shown in Table 4. It can be seen that lengthening the window size (by increasing the value of  $d$ ) lowers the effectiveness.

#### 4.2.3 Fine-tuning

From Sections 4.2.1 and 4.2.2, we have discovered that so far, the highest effectiveness is achieved by  $|q| = 100$  and  $d = 1$ .

Table 3: The MAP values for SBDL100 (the baseline) and  $S_d(q,a,1); |q| \in \{20, 50, 100\}$ . The best value is highlighted.  $S_d(q,a,1); |q| \geq 50$  is significantly better than the baseline (paired  $t$ -test,  $p < 0.0005$ ).

Method	MAP
SBDL100	0.229
$S_d(q,a,1);  q  = 20$	0.303
$S_d(q,a,1);  q  = 50$	0.575
$S_d(q,a,1);  q  = 100$	<b>0.745</b>

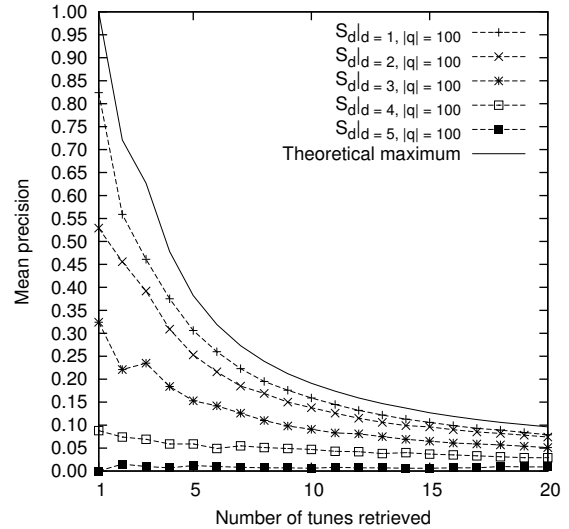


Figure 8: The mean precision at  $N$  ( $\langle P_N \rangle$ ) curves for  $S_d(q,a,d); d \in \{1, 2, 3, 4, 5\}, |q| = 100$ .

Table 4: The MAP values for SBDL100 (the baseline) and  $S_d(q,a,d); d \in \{1, 2, 3, 4, 5\}, |q| = 100$ . The best value is highlighted.  $S_d(q,a,d); d \leq 2$  is significantly better than the baseline (paired  $t$ -test,  $p < 0.0005$ ).

Method	MAP
SBDL100	0.229
$S_d(q,a,1)$	<b>0.745</b>
$S_d(q,a,2)$	0.495
$S_d(q,a,3)$	0.250
$S_d(q,a,4)$	0.112
$S_d(q,a,5)$	0.025

In this step, we tune the value of  $d$  by varying it in the range  $[0.5, 0.6, 0.7, \dots, 1.5]$ . The results are shown in Table 5. While the optimum value of  $d$  changes for different levels of  $N$ , it can be seen that the precision values vary only slightly within a range of good performance ( $1.0 \leq d \leq 1.4$ ). When the aim is to maximise performance in the high ranks ( $N = 1$  or  $2$ ) or even overall performance,  $d = 1.1$  is the best value.

**Table 5: Mean precision at  $N$  and MAP values for  $S_d(q, a, d); d \in \{0.5, 0.6, 0.7, \dots, 1.5\}, |q| = 100$ . The highest mean precision for every  $N$  and the highest MAP are highlighted.  $M$  is the theoretical maximum for the collection and query set.  $S_d(q, a, d); d \in \{0.5, 0.6, 0.7, \dots, 1.5\}, |q| = 100$  is significantly better than the baseline SBDL100 (paired  $t$ -test,  $p < 0.05$ ).**

$N$	$d$											$M$
	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	
1	0.441	0.529	0.618	0.706	0.765	0.824	0.824	0.765	0.735	0.706	0.676	1.000
2	0.250	0.309	0.397	0.485	0.515	0.559	0.588	0.574	0.574	0.574	0.574	0.721
3	0.176	0.206	0.274	0.382	0.421	0.461	0.500	0.490	0.490	0.500	0.490	0.627
4	0.140	0.162	0.243	0.309	0.338	0.375	0.397	0.382	0.375	0.397	0.382	0.478
5	0.112	0.129	0.212	0.247	0.276	0.306	0.318	0.312	0.312	0.324	0.312	0.382
6	0.103	0.113	0.181	0.206	0.245	0.260	0.265	0.265	0.265	0.270	0.265	0.319
7	0.088	0.101	0.156	0.181	0.214	0.223	0.227	0.227	0.231	0.231	0.227	0.273
8	0.077	0.088	0.140	0.162	0.188	0.195	0.199	0.199	0.206	0.202	0.202	0.239
9	0.069	0.085	0.124	0.147	0.167	0.176	0.176	0.180	0.183	0.180	0.183	0.212
10	0.065	0.076	0.112	0.132	0.153	0.159	0.159	0.162	0.168	0.165	0.165	0.191
11	0.064	0.070	0.102	0.120	0.139	0.145	0.145	0.147	0.153	0.150	0.150	0.174
12	0.061	0.066	0.093	0.113	0.127	0.132	0.132	0.135	0.140	0.137	0.137	0.159
13	0.057	0.066	0.086	0.104	0.118	0.122	0.122	0.127	0.129	0.127	0.127	0.147
14	0.052	0.061	0.080	0.099	0.109	0.113	0.115	0.117	0.119	0.117	0.117	0.137
15	0.049	0.057	0.075	0.092	0.102	0.106	0.108	0.110	0.112	0.110	0.110	0.127
16	0.046	0.057	0.070	0.086	0.095	0.099	0.101	0.103	0.105	0.103	0.103	0.119
17	0.043	0.054	0.066	0.081	0.090	0.093	0.097	0.097	0.099	0.097	0.097	0.112
18	0.041	0.053	0.062	0.079	0.085	0.089	0.092	0.092	0.093	0.092	0.092	0.106
19	0.040	0.053	0.061	0.074	0.081	0.084	0.087	0.087	0.088	0.087	0.087	0.101
20	0.038	0.050	0.059	0.071	0.076	0.079	0.082	0.082	0.084	0.082	0.082	0.096
<b>MAP</b>	<b>0.377</b>	<b>0.424</b>	<b>0.520</b>	<b>0.637</b>	<b>0.685</b>	<b>0.745</b>	<b>0.768</b>	<b>0.728</b>	<b>0.716</b>	<b>0.709</b>	<b>0.687</b>	<b>1.000</b>

## 5. DISCUSSION

Our experiments on query length reveal that query lengths of 20 and 50 symbols do not have enough discriminatory power when used in conjunction with the algorithm we devise here. However, the results show that queries with lengths of 100 symbols are sufficiently effective. Only one query in our set has a length of less than 100 symbols. Given an average query length of 814.15 symbols, only  $\frac{100}{814.15} = 12.3\%$  of the average query length is needed for effective retrieval. Therefore, our new approach results in a substantially lesser burden on users when creating queries.

As the LCS algorithm does not penalise non-matches, using greater  $d$  increases the probability that a query obtains the maximum possible score, that is the length of the query itself or the length of the answer, whichever one is shorter. However, from our experimental results in Section 4.2.2, it can be seen that using greater  $d$  also leads to many incorrect answers being scored highly. In turn, this also greatly narrows the separation between correct answers and incorrect ones. For example, using  $d = 1.1$  and  $|q| = 100$  lowers the score for incorrect answers even further compared to, say, using  $d = 2.0$  and the same query length, causing the correct answers to be pushed up to the top ranks. To illustrate this, suppose that we have a query  $q = "A B C"$  (hence  $|q| = 3$ ) to be matched against  $a_1 = "A A E E B D B C G G F G A"$  (assumed to be the correct answer) and  $a_2 = "D E D F A A F\# F\# D G D F B F\# F C A G"$ . Suppose that  $d = 1.8$ , so using Equation 3, the window size is  $W + 1 = \lceil 2 \times 1.8 \times 3 \rceil + 1 = 12$ . The window that gives the highest possible score—equal to  $|q| = 3$ —in  $a_1$  is:

A A E E B D B C G G F G A

In  $a_2$ , there is also a window that gives the highest possible score:

D E D F A A F\# F\# D G D F B F\# F C A G

Now, if we set  $d = 0.3$ , using Equation 3,  $W = \lceil 2 \times 0.3 \times 3 \rceil = 2$ , hence a window size of 3. There is no 3-symbol window in  $a_1$  that gives the highest possible score. The best score produced by  $a_1$  is 2, given by:

A A E E B D B C G G F G A

However, the best score produced by  $a_2$  is even lower, 1. There is no 3-symbol window that contains at least two symbols from  $q$  in sequence. Therefore, smaller windows sizes lead to a greater level of discrimination between good and poor answers.

For the queries BWV 1010 Bouree I and BWV 1010 Bouree II, the target answers contain both parts of the song combined into one song file. With the  $S_y(q, a, y)$  approach, the queries were penalised too much so that they performed poorly. This made the lengths of the answers disproportional to the lengths of the queries. Compare that with the query for BWV 1007 Menuets, which contains the two parts of BWV 1007 Menuets like the target audio pieces. It did not suffer from the same problem. The  $S_d(q, a, d)$  measure does not suffer from this problem either, and it can retrieve a correct answer in the first rank position in both situations.

In comparison with our previous work using local alignment [40],  $S_d(q, a, d)$  performs much more effectively than method SBDL100 (which, in turn, was shown to be more effective than SBDL50).

SBDL100 uses local alignment—rewarding matches and penalising mismatches, insertions, and deletions—and was tested on sequences of relative pitches called directed modulo 12, whereas  $S_d(q, a, d)$  is tested on sequences of absolute pitches. There are two issues here: alignment algorithm and representation. The LCS algorithm—which is equivalent to local alignment but with no penalty for non-match operations—is more appropriate as the amount of noise between matching symbols is very high, thereby lowering the effectiveness of applying penalties for non-match operations. Local alignment with non-match penalties is appropriate if the length of the query is not much shorter than the matching subsequence in the answer. Therefore, local alignment with non-match penalties is not suitable for this task. As for representation, as was explained in Section 3.1, the transcription process inserts a lot of noise; this renders an interval-based representation unreliable, as has been shown elsewhere [40].

## 6. CONCLUSIONS AND FUTURE WORK

This work proposed and evaluated a novel approach to retrieving polyphonic audio with polyphonic symbolic queries. The approach uses the LCS algorithm and a sliding window to match audio transcription and manually constructed queries. We have demonstrated that:

- It is possible to effectively retrieve audio by using symbolic music, whether in its full length or truncated.
- Truncated queries are highly effective, and enable users to issue short queries, hence reducing the burden on them.
- Absolute pitch representation is more suitable for this task compared to relative pitch, since the former is not susceptible to noise in transcriptions.

In future work, we plan to investigate the effectiveness of our approach on larger collections. We feel that including a large number of pop songs into the collection will pose a challenge, as they have relatively lower entropy compared to classical music. We also plan to focus on the efficiency of our approach, and to investigate how the method scales to larger data sets. We will explore indexing techniques to reduce retrieval cost, thus enabling a practical song-cover query-by-score system.

## 7. ACKNOWLEDGEMENTS

We thank Justin Zobel for obtaining TS-AudioToMidi and Steven Garcia for his input on automating the transcription process. We also thank the anonymous reviewers for their comments.

## 8. REFERENCES

- [1] J.-J. Aucouturier and M. Sandler. Using long-term structure to retrieve music: Representation and matching. In Downie and Bainbridge [10], pages 1–2.
- [2] J.-J. Aucouturier and M. Sandler. Finding repeating patterns in acoustical musical signals: Applications for audio thumbnailing. In *Proceedings of the Audio Engineering Society 22nd International Conference on Virtual, Synthetic and Entertainment Audio*, pages 412–421, Espoo, Finland, June 2002.
- [3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, USA, 1999.
- [4] C. Buckley and E. M. Voorhees. Retrieval system evaluation. In E. M. Voorhees and D. K. Harman, editors, *TREC: Experiment and Evaluation in Information Retrieval*, pages 53–78. MIT Press, Cambridge, USA, Sept. 2005.
- [5] C. L. Buyoli and R. Loureiro, editors. *Proceedings of the Fifth International Conference on Music Information Retrieval*, Barcelona, Spain, Oct. 2004. Audiovisual Institute Pompeu Fabra University.
- [6] L. P. Clarisse, J. P. Martens, M. Lesaffre, B. D. Baets, H. D. Meyer, and M. Leman. An auditory model based transcriber of singing sequences. In Fingerhut [12], pages 116–123.
- [7] M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, and J. F. Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Information Processing Letters*, 80(6):279–285, Dec. 2001.
- [8] R. Dannenberg, K. Lemström, and A. Tindale, editors. *Proceedings of the Seventh International Conference on Music Information Retrieval*, Victoria, Canada, Oct. 2006. University of Victoria.
- [9] R. B. Dannenberg, W. P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo. The Musart testbed for query-by-humming evaluation. In Hoos and Bainbridge [18], pages 41–47.
- [10] J. S. Downie and D. Bainbridge, editors. *Proceedings of the Second International Symposium on Music Information Retrieval*, Bloomington, USA, Oct. 2001.
- [11] J. Eggink and G. J. Brown. Extracting melody lines from complex audio. In Buyoli and Loureiro [5], pages 84–91.
- [12] M. Fingerhut, editor. *Proceedings of the Third International Conference on Music Information Retrieval*, Paris, France, Oct. 2002. IRCAM-Centre Pompidou.
- [13] J. Foote. Arthur: Retrieving orchestral music by long-term structure. In D. Byrd, J. S. Downie, T. Crawford, W. B. Croft, and C. Nevill-Manning, editors, *Proceedings of the First International Symposium on Music Information Retrieval*, Plymouth, USA, Oct. 2000.
- [14] D. C. Giancoli. *Physics: Principles with Applications*. Pearson Education, New Jersey, USA, sixth edition, 2005.
- [15] E. Gómez and P. Herrera. The song remains the same: Identifying versions of the same piece using tonal descriptors. In Dannenberg et al. [8], pages 180–185.
- [16] A. Guo and H. Siegelmann. Time-warped longest common subsequence algorithm for music retrieval. In Buyoli and Loureiro [5], pages 258–261.
- [17] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.
- [18] H. H. Hoos and D. Bainbridge, editors. *Proceedings of the Fourth International Conference on Music Information Retrieval*, Baltimore, USA, Oct. 2003. Johns Hopkins University.
- [19] N. Hu and R. B. Dannenberg. A comparison of melodic database retrieval techniques using sung queries. In G. Marchionini and W. Hersh, editors, *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 301–307, Portland, USA, July 2002.
- [20] N. Hu, R. B. Dannenberg, and G. Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *Proceedings of the 2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 185–188, New Paltz, USA, Oct. 2003.

- [21] J.-S. R. Jang, C.-L. Hsu, and H.-R. Lee. Continuous HMM and its enhancement for singing/humming query retrieval. In Reiss and Wiggins [34], pages 546–551.
- [22] A. Klapuri. Multiple fundamental frequency estimation by summing harmonic amplitudes. In Dannenberg et al. [8], pages 216–221.
- [23] M. Marolt. Gaussian mixture models for extraction of melodic lines from audio recordings. In Buyoli and Loureiro [5], pages 80–83.
- [24] M. Marolt. A mid-level melody-based representation for calculating audio similarity. In Dannenberg et al. [8], pages 280–285.
- [25] D. Mazzoni and R. B. Dannenberg. Melody matching directly from audio. In Downie and Bainbridge [10], pages 17–18.
- [26] M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In Reiss and Wiggins [34], pages 288–295.
- [27] M. Müller, F. Kurth, and T. Röder. Towards an efficient algorithm for automatic score-to-audio synchronization. In Buyoli and Loureiro [5], pages 365–372.
- [28] R. P. Paiva, T. Mendes, and A. Cardoso. On the detection of melody notes in polyphonic audio. In Reiss and Wiggins [34], pages 175–182.
- [29] B. Pardo and M. Sanghi. Polyphonic musical sequence alignment for database search. In Reiss and Wiggins [34], pages 215–222.
- [30] J. Pickens, J. P. Bello, G. Monti, T. Crawford, M. Dovey, M. Sandler, and D. Byrd. Polyphonic score retrieval using polyphonic audio queries: A harmonic modelling approach. In Fingerhut [12], pages 140–149.
- [31] J. Pickens and C. Iliopoulos. Markov random fields and maximum entropy modeling for music information retrieval. In Reiss and Wiggins [34], pages 207–214.
- [32] G. E. Poliner and D. P. W. Ellis. A classification approach to melody transcription. In Reiss and Wiggins [34], pages 161–166.
- [33] C. Raphael. A hybrid graphical model for aligning polyphonic audio with musical scores. In Buyoli and Loureiro [5], pages 387–394.
- [34] J. D. Reiss and G. A. Wiggins, editors. *Proceedings of the Sixth International Conference on Music Information Retrieval*, London, UK, Sept. 2005. Queen Mary, University of London.
- [35] S. Shalev-Shwartz, S. Dubnov, N. Friedman, and Y. Singer. Robust temporal and spectral modeling for query by melody. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 331–338, Tampere, Finland, Aug. 2002.
- [36] S. Shalev-Shwartz, J. Keshet, and Y. Singer. Learning to align polyphonic music. In Buyoli and Loureiro [5], pages 381–386.
- [37] J. Shifrin and W. P. Birmingham. Effectiveness of HMM-based retrieval on large databases. In Hoos and Bainbridge [18], pages 33–39.
- [38] F. Soulez, X. Rodet, and D. Schwarz. Improving polyphonic and poly-instrumental music to score alignment. In Hoos and Bainbridge [18], pages 143–148.
- [39] I. S. H. Suyoto and A. L. Uitdenbogerd. Effectiveness of note duration information for music retrieval. In L. Zhou, B. C. Ooi, and X. Meng, editors, *Proceedings of the Tenth International Conference on Database Systems for Advanced Applications*, pages 265–275, Beijing, China, Apr. 2005. Springer-Verlag. Published as LNCS 3453.
- [40] I. S. H. Suyoto and A. L. Uitdenbogerd. Aligning musical audio with symbols: A case study in Western classical music. Technical Report TR-07-1, School of Computer Science and Information Technology, RMIT, Mar. 2007. <http://mirt.cs.rmit.edu.au/pubs/sbdg/>.
- [41] R. Typke, F. Wiering, and R. C. Veltkamp. A search method for notated polyphonic music with pitch and tempo fluctuations. In Buyoli and Loureiro [5], pages 281–288.
- [42] A. L. Uitdenbogerd. *Music Information Retrieval Technology*. PhD thesis, School of Computer Science and Information Technology, RMIT, Melbourne, Australia, 2002.
- [43] A. L. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In D. Bulterman, K. Jeffay, and H. J. Zhang, editors, *Proceedings of the 1999 ACM Multimedia Conference*, pages 57–66, Orlando, USA, Nov. 1999.
- [44] A. Wang. The Shazam music recognition service. *Communications of the ACM*, 49(8):44–48, Aug. 2006.