TOWARDS MORE ROBUST GEOMETRIC CONTENT-BASED MUSIC RETRIEVAL

Kjell Lemström
Department of Computer Science
University of Helsinki

ABSTRACT

This paper studies the problem of transposition and time-scale invariant (*ttsi*) polyphonic music retrieval in symbolically encoded music. In the setting, music is represented by sets of points in plane. We give two new algorithms. Applying a search window of size w and given a query point set, of size m, to be searched for in a database point set, of size n, our algorithm for exact *ttsi* occurrences runs in $O(mwn \log n)$ time; for partial occurrences we have an $O(mnw^2 \log n)$ algorithm. The framework used is flexible allowing development towards even more robust geometric retrieval.

1. INTRODUCTION

Query-by-humming type problems have been under study for over fifteen years. First, the music under investigation was assumed to be monophonic [3], later the term has been used with a wider meaning addressing problems where the task is to search for excerpts of music, resembling a given query pattern, in a large database. Moreover, both the query pattern and the database may be polyphonic, and the query pattern constitutes only a subset of instruments appearing in the database representing possibly a full orchestration of a musical piece. Although current audio-based methods can be applied to rudimentary cases where queries are directed to clearly separable melodies, the general setting requires methods based on symbolic representation that are truly capable of dealing with polyphonic subset matching.

To this end, several authors have recently used geometric-based modeling of music [1, 7–9]. Geometric representations usually also take into account another feature intrinsic to the problem: the matching process ignores extra intervening notes in the database that do not appear in the query pattern. Such extra notes are always present because of the polyphony, various noise sources and musical decorations. There is, however, a notable downside to the current geometric methods: they do not usually allow distortions in tempo (except for individual outliers that are not even discovered) that are inevitable in the application. Even if the query could be given exactly on tempo, the occurrences in the database would be time-scaled versions of the query (requiring *time-scale invariance*). If the query is to be given in a live performance, local jittering will inevitably take place and a stronger invariance, namely the *time-warping invariance* [4], would be a desired property for the matching process.

In this paper, new time-scale invariant geometric algorithms that deal with symbolically encoded, polyphonic music will be introduced. We use the pitch-against-time representation of note-on information, as suggested in [9] (see Fig 1). The musical works in a database are concatenated in a single geometrically represented file, denoted by T; $T = t_0, t_1, \ldots, t_{n-1}$, where $t_j \in \mathbb{R}^2$ for $0 \leq j \leq n-1$. In a typical retrieval case the query pattern P, $P = p_0, p_1, \ldots, p_{m-1}$; $p_i \in \mathbb{R}^2$ for $0 \leq i \leq m-1$, to be searched for is often monophonic and much shorter than the database T to be searched. Sometimes a search window w is applied and typically $w < m$, that is $w < m \ll n$. It is assumed that P and T are given in the lexicographic order. If this is not the case, the sets can be sorted in $O(m \log m)$ and $O(n \log n)$ times, respectively.

The problems under consideration are modified versions of two problems originally represented in [8]. The following list gives both the original problems and the modifications under consideration; for the partial matches in P2 and S2, one may either use a threshold α to limit the minimum size of an accepted match, or to search for maximally sized matches only.

(P1) Find translations of P such that each point in P matches with a point in T.

(P2) Find translations of P that give a partial match of the points in P with the points in T.

(S1) Find time-scaled translations of P such that each point in P matches with a point in T.

(S2) Find time-scaled translations of P that give a partial match of the points in P with the points in T.

Fig. 2 gives 4 query patterns to be searched for in the excerpt of Fig. 1, exemplifying these 4 problems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.
© 2010 International Society for Music Information Retrieval.

In [6], Romming and Selfridge-Field gave a geometric hashing-based algorithm for S2. Without windowing, it works in $O(n^3)$ space and $O(n^2m^3)$ time. This paper studies another way to solve problems S1 and S2. The new algorithms to be introduced resemble Ukkonen et al's PI and PII algorithms. The algorithm for S1 runs in time $O(mn^2 \log n)$ and $O(mn^2)$ space; the algorithm for S2 in $O(m^2n^2logn)$ time and $O(m^2n^2)$ space. An advantage of our method over the one by Romming and Selfridge-Field is that the performance can be spedup by applying an index-filter preprocessing [5]. Our method also seems to be adaptable to time-warping invariant cases. Thus, the method is an important step towards more robust geometric content-based music retrieval.

2. RELATED WORK

Let α denote a similarity threshold for P2, and let $p_0, p_1, \ldots, p_{m-1}$ and $t_0, t_1, \ldots, t_{n-1}$ be the pattern and database points, respectively, *lexicographically sorted* according to their co-ordinate values: $p_i < p_{i+1}$ iff $p_i.x < p_{i+1}.x$ or ($p_i.x = p_{i+1}.x$ and $p_i.y < p_{i+1}.y$), and $t_j < t_{j+1}$ iff $t_j.x < t_{j+1}.x$ or ($t_j.x = t_{j+1}.x$ and $t_j.y < t_{j+1}.y$). In our application the elapsing time runs along the horisontal axes, represented by x, the perceived height, the *pitch*, is represented by y. A translation of P by vector f is denoted $P + f$: $P + f = p_0 + f, \ldots, p_{m-1} + f$. Using this notation, problem P1 is expressible as the search for a subset I of T and some f such that $P + f = I$. Note that decomposing translation f into horisontal and vertical components $f.x$ and $f.y$, respectively, captures two musically distinct phenomena: $f.x$ corresponds to aligning the pattern time-wise, $f.y$ to *transposing* the musical excerpt to a lower or higher key. Note also that a musical time-scaling σ, $\sigma \in \mathbb{R}^+$, has an effect only on the horisontal translation, the vertical translation stays intact.

Example 2.1. *Let* $p = \langle 1, 1\rangle$, $f = \langle 2, 2\rangle$ *and* $\sigma = 3$. *Then* $p + \sigma f = \langle 7, 3\rangle$.

A straight-forward algorithm solves P1 and P2 in $O(mn \log(mn))$ time. The algorithm first exhaustively collects all the translations mapping a point in P to another point in T. The set of the collected translation vectors are then sorted in lexicographic order. In the case of P1, a translation f that has been used m times corresponds to an occurrence; in the case of P2, any translation f that has been used at least α times would account for an occurrence. Thoughtful implementations of the involved scanning (sorting) of the translation vectors, will yield an $O(mn)$ ($O(mn \log m)$) time algorithm for P1 (P2) [8].

Indeed, the above $O(mn \log m)$ time algorithm is the fastest online algorithm known for P2. Moreover, any significant improvement in the asymptotic running time, exceeding the removal of the logarithmic factor, cannot be seen to exist, for P2 is known to be a `3SUM`-hard problem [2]. It is still possible that P2 is also a `Sorting X+Y` -hard problem, in which case Ukkonen et al's PII algorithm would already be an optimal solution. In [2], Clifford et al introduced an $O(n \log n)$ time approximation algorithm for P2.

To make the queries more efficient, several indexing schemes have been suggested. The first indexing method using geometric music representation was suggested by Clausen et al. [1]. Their sublinear query times were achieved by using inverted files, adopted from textual information retrieval. The performance was achieved with a lossy feature extraction process, which makes the approach non-applicable to problems P1 and P2. Typke [7] proposed the use of metric indexes that works under robust geometric similarity measures. However, it is difficult to adopt his method to support translations and partial matching at the same time. Lemström et al's approach [5] combines sparse indexing and (practically) lossless filtering. Their index is used to speed up a filtering phase that charts all the promising areas in the database where real occurrences could reside. Once a query has been received, the filtering phase works in time $O(g_f(m) \log n + n)$ where function $g_f(m)$ is specific to the applied filter f. The last phase involves checking the found c_f ($c_f \leq n$) candidate positions using Ukkonen et al's PI or PII algorithm executable in worst-case time $O(c_f m)$ or $O(c_f m \log m)$, respectively.

The only non-brute-force solution known for S1 and S2 is by Romming and Selfridge-Field [6]. It is based on geometric hashing and works in $O(n^3)$ space and $O(n^2m^3)$ time. By applying a window on the database such that w is the maximum number of events that occur in any window, the above complexities can be restated as $O(w^2n)$ and $O(wnm^3)$, respectively.

3. MATCHING ALGORITHMS

Our matching algorithms for the time-scale invariant problems S1 and S2 resemble somewhat Ukkonen et al's PI and PII algorithms in that they all use a priority queue as a focal structure. Ukkonen et al's PI and PII work on trans-set translations, or *trans-set vectors*, $f = t - p$, where p and t are points in a given query pattern, of length m, and in the underlying database, of length n, respectively. Let us assume (without loss of generality) that all the points, both in the pattern and in the database, are unique. The number of trans-set vectors is within the range $[n+m-1, nm]$. In order to be able to build an index on the database in an offline phase, Lemström et al's method [5] is based on *intra-set vectors*. For instance, translation vector f is an *intra-pattern vector*, if there are two points p and p' ($p, p' \in P$) such that $p + f = p'$. *Intra-database vectors* are defined accordingly. Naturally, the number of intra-pattern and intra-database vectors are $O(m^2)$ and $O(n^2)$, respectively.

The set of *positive intra-pattern vectors* include translations $p_{i'} - p_i$ where in the case of S1: $0 \leq i < m$

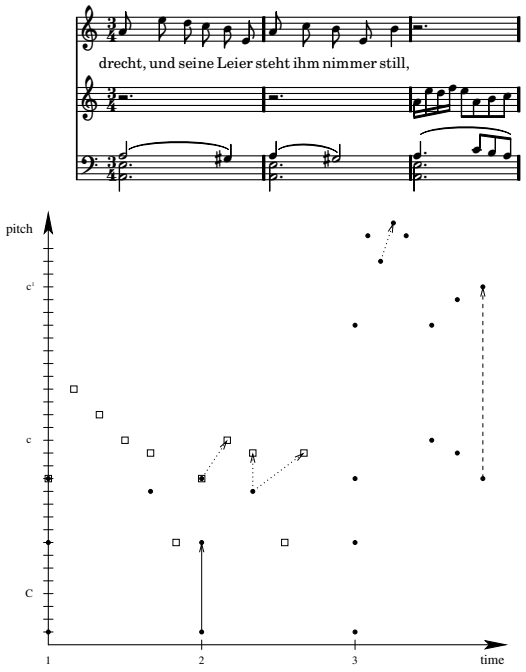


Figure 1. On top, an excerpt from Franz Schubert's song cycle Winterreise. Below, the related geometric, point-set representation. The points associated with the vocal part are represented distinctly (by squares). The depicted 6 intra-database vectors will be discussed later.

and $i' = i + 1$, and in the case of S2: $0 \leq i < i' \leq m$. The set of *positive intra-database vectors* include translations $t_{k'} - t_k$ where, independently of the case, $0 \leq k < k' \leq n$. To reduce the search space, one may apply a window that restates the bounds for i' (in the case of S2) and k' in the obvious way: $0 \leq i < i' \leq \min\{i + w, m\}$ and $0 \leq k < k' \leq \min\{k + w, n\}$.

For the convenience of the algorithms, we pretend that there is an extra elements p_m in the pattern and t_n in the database. The matching algorithms take as input intra-set vectors, stored in tables $K[i], 0 \leq i < m$. Table $K[i]$ stores intra-database translations that may match [1] the positive intra-pattern vectors $p_{i'} - p_i$, i.e., translation vectors starting at point p_i. See Fig. 3 for an illustration on tables $K[i]$.

The entries in our main data structures will be sorted in a lexicographic order. We will specify the underlying order by an ordered set $\aleph$. $\aleph$ is formed by members of $\{a, b, s\}$, where a, b and s correspond to the columns named accordingly in tables $K[i]$. For instance, lexicographic order $\langle a, s\rangle$ is firstly based on the values on column a (the starting point of the associated intra-database vector), secondly on the values on column s (the associated scaling value). A main loop that goes exhaustively through all the possibilities of positive intra-pattern and positive intra-database vectors to initialise the tables $K[i]$ is needed. To this end, let a positive intra-database vector $g = t_{k'} - t_k$ be such that there is a positive intra-pattern vector $f = p_{i'} - p_i$

[1] Please note the distinction between an occurrence and a match. An *occurrence* involves as many *matching* pairs of intra-database and intra-pattern vectors as is required.

$\langle 0, 7\rangle$	$\langle 0, 12\rangle$	$\langle 2, 15\rangle$	$\langle 0, 5\rangle$	$\langle 2, 8\rangle$	$\langle 4, 4\rangle$
$\langle \mathbf{2}, \mathbf{3}\rangle$	$\langle 4, -1\rangle$	$\langle 4, 2\rangle$	$\langle 2, -4\rangle$	$\langle 2, -1\rangle$	$\langle 4, -8\rangle$
$\langle \mathbf{0}, \mathbf{3}\rangle$	$\langle 2, -4\rangle$	$\langle \mathbf{4}, \mathbf{3}\rangle$	$\langle 2, -7\rangle$	$\langle 4, 0\rangle$	$\langle 6, -14\rangle$
$\langle 2, 7\rangle$	$\langle 4, -7\rangle$	$\langle 4, 0\rangle$	$\langle 2, -14\rangle$	$\langle 2, -7\rangle$	$\langle 2, -2\rangle$
$\langle 0, 7\rangle$	$\langle 0, 12\rangle$	$\langle 0, 24\rangle$	$\langle 0, 5\rangle$	$\langle 0, 17\rangle$	$\langle 1, 24\rangle$
$\langle 0, 12\rangle$	$\langle 1, 19\rangle$	$\langle 2, 17\rangle$	$\langle 1, 7\rangle$	$\langle 2, 5\rangle$	$\langle 3, 8\rangle$
$\langle 1, -2\rangle$	$\langle 2, 1\rangle$	$\langle 3, 0\rangle$	$\langle \mathbf{1}, \mathbf{3}\rangle$	$\langle 2, 2\rangle$	$\langle 4, -14\rangle$
$\langle 1, -1\rangle$	$\langle 3, -17\rangle$	$\langle 3, -8\rangle$	$\langle 2, -16\rangle$	$\langle 2, -7\rangle$	$\langle 4, -18\rangle$
$\langle 0, 9\rangle$	$\langle 2, -1\rangle$	$\langle 2, 11\rangle$	$\langle 2, -10\rangle$	$\langle 2, 2\rangle$	$\langle 4, -12\rangle$
$\langle 0, 12\rangle$	$\langle 2, -2\rangle$	$\langle 2, 13\rangle$	$\langle 2, -14\rangle$	$\langle 0, 1\rangle$	$\langle 0, 15\rangle$

Table 1. The intra-database vectors generated by the example given in Fig. 1 when ignoring the first bar and setting $w = 3$. The first and the last intra-database vectors under consideration are depicted in Fig. 1 as arrows with solid and dashed stems, respectively.

for which $g.y = f.y$ (ie. the pitch intervals of the two vectors match). Because g may be part of an occurrence, a new row, let it be the hth, in $K[i]$ is allocated and the following updates are conducted:

$$K[i]_h.a \leftarrow k; \quad K[i]_h.b \leftarrow k'; \tag{1}$$

$$K[i]_h.s \leftarrow \frac{t_{k'}.x - t_k.x}{p_{i'}.x - p_i.x}; \tag{2}$$

$$K[i]_h.y \leftarrow \mathbf{nil}; \quad K[i]_h.w \leftarrow 1; \tag{3}$$

$$K[i]_h.c \leftarrow i'; \quad K[i]_h.z \leftarrow 0. \tag{4}$$

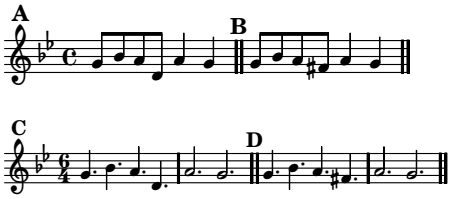
Above, in (1), the associated starting and ending points of the matching intra-database vector are stored in $K[i]_h.a$ and $K[i]_h.b$, respectively. The required time scaling for the intra-vectors to match is stored in $K[i]_h.s$ (2); here extra carefulness is needed to avoid zero division: If both the numerator and the denominator equal zero, we set $K[i]_h.s = 1$. If only one of them equals zero or both equal infinity, the whole row is deleted from the table (they would represent impossible time scalings). Columns y and w, initialised in (3), are used for backtracking a found occurrence and storing the length of a candidate occurrence, respectively. The last columns, initialised in (4), will be needed when searching for partial occurrences (in Section 3.2): column c stores the ending point of the associated intra-pattern vector, z is used for identifying an occurrence.

Denoting by Σ_{p_i} the number of rows generated above for table $K[i]$, $0 \leq i < m$, for the aforementioned extra elements we set:

$$K[i]_{\Sigma_{p_i}}.a \leftarrow K[i]_{\Sigma_{p_i}}.b \leftarrow \infty;$$
$$K[i]_{\Sigma_{p_i}}.s \leftarrow K[i]_{\Sigma_{p_i}}.w \leftarrow 0;\ K[i]_{\Sigma_{p_i}}.c \leftarrow i + 1$$

As each iteration of the main loop takes constant time, this exhaustive initialisation process runs in time $O(nmw^2)$. Finally, the columns in $K[i]$ are sorted in lexicographic order $\langle a, s\rangle$. The matching algorithms have an associated priority queue Q_i for each table $K[i]$, $0 < i \leq m$ [2]. For Q_i, a lexicographic order $\langle b, s\rangle$ is used. As a reminder, the order is given in the superscript of a priority queue (e.g. $Q_i^{\langle b,s\rangle}$).

[2] A single priority queue would suffice, but the algorithm would become more complicated.



$\langle \mathbf{6}, \mathbf{3} \rangle \quad \langle 6, -2 \rangle \quad \langle 6, -7 \rangle \quad \langle 6, 7 \rangle \quad \langle 12, -2 \rangle$

Figure 2. On top, 4 example queries. For query **A** an occurrence in the excerpt given in Fig. 1 would be found in all four cases P1, P2, S1 and S2; for **B** in cases P2, S1 and S2; for **C** in S1 and S2; and for **D** in S2 only. At the bottom, the positive intra-pattern vectors, associated with the query **C** in case S1.

3.1 S1: Time Scaled Exact Matching

Once the tables $K[i]$ have been initialised and their columns have been sorted in lexicographic order $\langle a, s \rangle$, the transposition-invariant time-scaled exact occurrences can be found using the matching algorithm given in Fig. 4. The algorithm works by observing *piecewise matches* between positive intra-database and intra-pattern vectors

$$t_{k_{i'}} - t_{k_i} = \sigma_i(p_{i+1} - p_i) \tag{5}$$

that are stored in the associated $K[i]$. Above $\sigma_i \in \mathbb{R}^+$ is the time-scaling factor (recall Example 2.1). The piecewise matches may form a chain $T_{\tau_0 \ldots \tau_{m-1}} = t_{\tau_0}, t_{\tau_1}, \ldots, t_{\tau_{m-1}}$, where $\tau_0, \tau_1, \ldots, \tau_{m-1}$ is an increasing sequence of indices in T; $t_{\tau_{i+1}} - t_{\tau_i} = \sigma(p_{i+1} - p_i)$ for $0 \leq i < m-1$ and $\sigma \in \mathbb{R}^+$ is a time-scaling factor common to all the piecewise matches in the chain. As such chains represent transposition-invariant, time-scaled exact occurrences, the task is to look for them.

A chain $T_{\tau_0 \ldots \tau_{m'}}$ $m' < m-1$, is called a *prefix occurrence* (of length m'); $T_{\tau_{m'-1}, \tau_{m'}}$ is the *final suffix* of the prefix occurrence $T_{\tau_0 \ldots \tau_{m'}}$. Let $t_{\tau_{i+1}} - t_{\tau_i}$ (that, by definition, equals $\sigma(p_{i+1} - p_i)$) be the final suffix of a prefix occurrence $T_{\tau_1 \ldots \tau_{m'}}$. The prefix occurrence is *extensible* if there is a piecewise match $t_{k'_{i+1}} - t_{k_{i+1}} = \sigma(p_{i+2} - p_{i+1})$ such that

$$t_{\tau_{i+1}} = t_{k_{i+1}} \tag{6}$$

and scaling factor σ is the one that was used in forming $T_{\tau_1 \ldots \tau_{m'}}$. The binding in Equation 6 is called the *binding of extension*, $t_{\tau_{i+1}} - t_{\tau_i}$ the *antecedent* and $t_{k'_{i+1}} - t_{k_{i+1}}$ the *postcedent of the binding.*

Lemma 3.1. *If a prefix occurrence is extensible, its final suffix is also extensible.*

Proof. Immediate. □

To be more efficient, at point $i+1$, the algorithm actually considers any piecewise match $t_{k'_i} - t_{k_i} = \sigma_i(p_{i+1} - p_i)$ as an antecedent to the binding and tries to extend it. Because in this case the piecewise matches in an occurrence chain have to be consecutive in P, the antecedents of the binding are all found in $K[i]$ and their possible extensions, postcedents, in $K[i+1]$. To process all the bindings of extension at point $i+1$, therefore, involves going through all the entries both in $K[i]$ and in $K[i+1]$. To make this process efficient, no entry of either of the tables should be observed more than once for one iteration. In order for this to be possible, both sides of the binding of extension (associated with antecedents and postcedents) should be enumerated in the same (increasing) order. However, the lefthand side of the binding involves end points of the intra-database vectors in $K[i]$ and the righthand side the start points of the intra-database vectors in $K[i+1]$. Therefore, we use a priority queue $Q_i^{\langle b,s \rangle}$ whose entries are addresses to rows associated with the antecedents of the binding at i. In this way, the binding of extension at i can be done efficiently by enumerating the entries in Q_i and $K[i]$. Note that the set of piecewise matches extended this way also includes all the final suffixes, and therefore, according to Lemma 3.1, also all the prefix occurrences.

The binding of extension takes place in line (8) of the algorithm. If a piecewise match is extensible, its length is updated (line 9) and a backtracking link is stored (line 10). The latter becomes useful if any of the extended piecewise matches extends into a proper occurrence, and the whole occurrence is to be revealed (instead of just reporting it).

Let us now demonstrate the main idea of the algorithm by using a musical example.

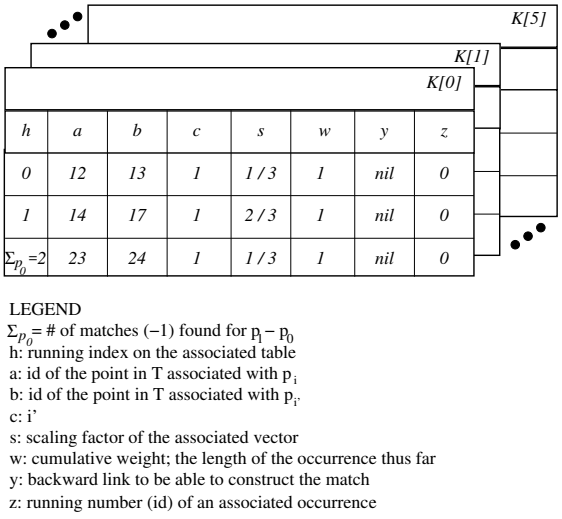


Figure 3. Illustration of tables $K[i]$ when considering problem (S1) and searching for occurrences for the query **C** of Fig. 2 within the two last bars of Fig. 1, $w = 3$. The intra-database vectors under consideration are the ones given in Table 1. Having initialized the tables K in equations (1-4), $K[0]$ contains the depicted 3 rows.

Example 3.2. *The vocal line in Fig. 1 ends in a suspension that is dissolved at the beginning of the third*

TimeScaledExactOccurrence($K[i]$)
(0) $K[0]_{\Sigma_{p_0}}.s \leftarrow \infty$
(1) **for** $j \leftarrow 0, \ldots, \Sigma_{p_0}$ **do**
(2) $\quad Q_1^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[0]_j)$
(3) **for** $i \leftarrow 1, \ldots, m-2$ **do**
(4) $\quad q \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(5) $\quad$ **for** $j \leftarrow 0, \ldots, \Sigma_{p_i} - 1$ **do**
(6) $\quad\quad$ **while** $[q.b, q.s] < [K[i]_j.a, K[i]_j.s]$ **do**
(7) $\quad\quad\quad q \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(8) $\quad\quad$ **if** $[q.b, q.s] = [K[i]_j.a, K[i]_j.s]$ **then**
(9) $\quad\quad\quad K[i]_j.w \leftarrow q.w + 1$
(10) $\quad\quad\quad K[i]_j.y \leftarrow q$
(11) $\quad\quad\quad Q_{i+1}^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[i]_j)$
(12) $\quad\quad\quad q \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(13) $\quad K[i]_{\Sigma_{p_i}}.s \leftarrow \infty$
(14) $\quad Q_{i+1}^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[i]_{\Sigma_{p_i}})$
(15) **if** $K[m-2]_j.w = m-1$ **for some** $0 \leq j \leq \Sigma_{p_{m-2}}$
(16) $\quad$ **then** `report an occurrence`

Figure 4. Algorithm for finding transposition-invariant time-scaled exact occurrences.

bar. As the expectation for the dissolution is strong and the lower part of the accompaniment resides in the same register as the vocal part, if suitably arranged, the listener perceives the higher "a" of the lower part of the accompaniment to belong to the vocal melody. The queries in Fig. 2 look for occurrences with such a dissolution. To solve S1 with the query **C** *of Fig. 2, the algorithm first fills Table* $K[0]$ *with rows corresponding to the intra-database vectors that match the interval of the first intra-pattern vector* $\langle \mathbf{6}, \mathbf{3} \rangle$ *(bolded in the figure). The matching vectors are depicted in Fig. 1 (arrows with a dotted stem) and given bolded in Table 1. Note that the vector* $\langle \mathbf{0}, \mathbf{3} \rangle$ *is not accepted since the associated scaling would actually squeeze any melody to a chord. The three accepted piecewise matches are stored in* $K[0]$ *(see Fig. 3) and the algorithm continues by looking for piecewise matches in* $K[1]$ *that could extend them.*

Analysis. Let us denote by $|Q_i|$ and $|K[j]|$ the number of entries in Q_i and $K[j]$, respectively. Clearly, in this case, $|Q_i| = |K[i-1]|$ for $1 \leq i \leq m$. Moreover, let $\Sigma = max_{i=1}^{m}(|Q_i|, |K[i-1]|)$. The outer loop (line (3)) is iterated m times. Within the inner loop (line (5)), all the entries in Q_i and in $K[i]$ are processed exactly once, resulting in $O(\Sigma)$ entry processing steps. The only operation taking more than a constant time is the updating of the priority queue; it takes at most $O(\log \Sigma)$ time. Thus, the algorithm runs in time $O(m\Sigma \log \Sigma)$. Moreover, the tables $K[i]$ and priority queues Q_i require $O(m\Sigma)$ space.

In this case $\Sigma = O(wn)$, because each table $K[i]$ contains the piecewise matches for the positive intra-pattern vector $p_{i+1} - p_i$, and there are $O(wn)$ possibilities to this end. Naturally $w = n$ if no windowing has been applied. □

TimeScaledPartialOccurrence($K[i]$)
(0) $\ell \leftarrow 0$; $K[0]_{\Sigma_{p_0}}.s \leftarrow \infty$
(1) **for** $i \leftarrow 0, \ldots, m-2$
(2) $\quad$ **for** $j \leftarrow 0, \ldots, \Sigma_{p_0}$
(3) $\quad\quad Q_{K[i]_j.c}^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[i]_j)$
(4) **for** $i \leftarrow 1, \ldots, m-2$ **do**
(5) $\quad q \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(6) $\quad$ **for** $j \leftarrow 0, \ldots, \Sigma_{p_i} - 1$ **do**
(6,5) $\quad\quad$ **if** $[q.b, q.s] > [K[i]_{\Sigma_{p_i}}.a, K[i]_{\Sigma_{p_i}}.s]$ **break**
(7) $\quad\quad$ **while** $[q.b, q.s] < [K[i]_j.a, K[i]_j.s]$ **do**
(8) $\quad\quad\quad q \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(9) $\quad\quad$ **if** $[q.b, q.s] = [K[i]_j.a, K[i]_j.s]$ **then**
(10) $\quad\quad\quad$ **while min**$(Q_i^{\langle b,s \rangle}) = [q.b, q.s]$ **do**
(11) $\quad\quad\quad\quad r \leftarrow$ **pop**$(Q_i^{\langle b,s \rangle})$
(12) $\quad\quad\quad\quad$ **if** $r.w > q.w$ **then** $q \leftarrow r$
(13) $\quad\quad\quad K[i]_j.w \leftarrow q.w + 1$
(14) $\quad\quad\quad K[i]_j.y \leftarrow q$
(15) $\quad\quad\quad$ **if** $K[i]_j.w = \alpha$ **then**
(16) $\quad\quad\quad\quad \ell \leftarrow \ell + 1$
(17) $\quad\quad\quad\quad K[i]_j.z = \ell$
(18) $\quad\quad\quad\quad \kappa[\ell] \leftarrow \&K[i]_j$
(19) $\quad\quad\quad$ **if** $K[i]_j.w > \alpha$ **then**
(20) $\quad\quad\quad\quad K[i]_j.z = q.z$
(21) $\quad\quad\quad\quad \kappa[q.z] \leftarrow \&K[i]_j$
(22) $\quad\quad\quad Q_{K[i]_j.c}^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[i]_j)$
(23) $\quad K[i]_{\Sigma_{p_i}}.s \leftarrow \infty$
(24) $\quad Q_{i+1}^{\langle b,s \rangle} \leftarrow$ **push**$(\&K[i]_{\Sigma_{p_i}})$
(25) ReportOccurrences(κ)

Figure 5. Algorithm for finding transposition-invariant time-scaled partial occurrences. The optimizer at line (6,5) can be omitted without breaking functionality; it can also be used to optimize the previous algorithm (as line (5,5)).

3.2 S2: Time Scaled Partial Matching

In order to be able to find transposition-invariant time-scaled partial occurrences, we need the two extra columns c and z, that were initialised in Equation 4, for tables $K[i]$. Recall that $K[i]_h.c$ stores the ending point i' for an intra-pattern vector $p_{i'} - p_i$ that is found to match an intra-database vector $t_{k'} - t_k$ with some scaling factor σ_i. Column z is used for storing a running number that is used as an id, for a found partial occurrence. Furthermore, we use an extra table, denoted by κ, for storing all the found occurrences.

The structure of the algorithm (see Fig. 5) is similar to the previous algorithm. Again, at point i, the antecedents in Q_i are to be extended by postcedents found in $K[i]$. However, as we are looking for partial occurrences this time, we cannot rely on piecewise

matches that are consecutive in P but any piecewise match associated with a positive intra-pattern vector

$$t_{k_{i'}} - t_{k_i} = \sigma_i(p_{i'} - p_i) \quad (7)$$

has to be considered. Here $0 \leq k_i < k_{i'} \leq \min\{k_i + w, n\}$; $0 \leq i < i' \leq \min\{i+w, m\}$ and $\sigma_i \in \mathbb{R}^+$. Given a threshold α, a chain $T_{\tau_0 \ldots \tau_{\beta-1}}$, such that $t_{\tau_j} - t_{\tau_{j-1}} = \sigma(p_{\pi_j} - p_{\pi_{j-1}})$, for $0 < j \leq \beta - 1$, $\beta \geq \alpha$, where $\tau_0 \ldots \tau_{\beta-1}$ and $\pi_0 \ldots \pi_{\beta-1}$ are increasing sequences of indices in T and P, respectively, would constitute for a transposition-invariant time-scaled partial occurrence (of length β).

That piecewise matches can now be between any two points in the pattern makes the problem somewhat more challenging. This has the effect that, at point i, pushing a reference to a priority queue (lines (2) and (21) of the algorithm) may involve any future priority queue, from Q_{i+1} to Q_m, not just the successive one as in the previous case; the correct priority queue is the one that is stored in $K[i]_j.c$ (recall that it stores the end point of the intra-pattern vector associated with the piecewise match). Conversely, the antecedents at point i (stored in Q_i) may include references to any past tables within the window size, expanding the size of the priority queue Q_i.

The two remaining differences to the algorithm above, are in lines (11) and (14-20). In line (11), the algorithm chooses to extend the piecewise match that is associated with the longest prefix occurrence. This is a necessary step, once again, because we are no more dealing with piecewise matches that are consecutive in P. In lines (14-20) the algorithm deals with a found occurrence. Lines (14-17) deal with a new occurrence: generate a new running number, ℓ, for it (that is used as its id) and store a link to the found occurrence to the table of occurrences κ. Lines (18-20) deal with extending a previously found occurrence.

Analysis. Let $\Sigma = max_{i=1}^{m}(|Q_i|, |K[i-1]|)$. With an analogous reasoning to that of the previous analysis, we arrive at similar complexities: the algorithm runs in $O(m\Sigma \log \Sigma)$ time and $O(m\Sigma)$ space. Let us now analyse the order of Σ in this case. Still it holds that for a positive intra-pattern vector, $p_{i+1} - p_i$, there are $O(wn)$ possible piecewise matches. However, the table $K[i]$ may contain entries associated with piecewise matches with any positive intra-pattern vector ending at point $i+1$. Thus, $max_{i=1}^{m}(|K[i-1]|) = O(\min\{m, w\}wn)$. As $|Q_i| = |K[i-1]|$ for $0 < i \leq m$ and assuming $w < m$, we conclude that the algorithm has an $O(mnw^2 \log n)$ running time and works in a space $O(mnw^2)$. □

4. CONCLUSIONS

In this paper we suggested novel content-based music retrieval algorithms for polyphonic, geometrically represented music. The algorithms are both transposition and time-scale invariant. Given a query pattern $P = p_0, \ldots, p_{m-1}$ to be searched for in a music database $T = t_0, \ldots, t_{n-1}$ and applying a search window of size w, the algorithms run in $O(m\Sigma \log \Sigma)$ time and $O(m\Sigma)$ space, where $\Sigma = O(wn)$ when searching for exact occurrences under such a setting, and $\Sigma = O(nw^2)$ when searching for partial occurrences. Whether this is an improvement in practice over the existing algorithm by Romming and Selfridge-Field [6], working in space $O(w^2n)$ and time $O(wnm^3)$, is left for future experiments on real data.

However, the framework seems to be very flexible: it is currently under modification to a more complex case, where an uneven time deformation is known just to preserve the order of the notes; there are no known solutions for this time-warping invariant problem [4]. Moreover, it seems that with some modifications to the data structures and ideas presented in [5] it would be possible to adopt the idea of using a three-phase searching process (indexing, filtering and checking) resulting in a smaller search space and a better running time to those presented here.

Acknowledgements This work was supported by the Academy of Finland (grants #108547 and #218156).

5. REFERENCES

[1] M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz. Proms: A web-based tool for searching in polyphonic music. In *Proc. ISMIR'00*, Plymouth, MA, October 2000.

[2] R. Clifford, M. Christodoulakis, T. Crawford, D. Meredith, and G. Wiggins. A fast, randomised, maximal subset matching algorithm for document-level music retrieval. In *Proc. ISMIR'06*, pp. 150–155, Victoria, BC, October 2006.

[3] A. Ghias, J. Logan, D. Chamberlin, and B. Smith. Query by humming - musical information retrieval in an audio database. In *Proc. ACM Multimedia*, pages 231–236, San Francisco, CA, 1995.

[4] K. Lemström and G. Wiggins. Formalizing invariances for content-based music retrieval. In *Proc. ISMIR'09*, pp. 591–596, Kobe, October 2009.

[5] K. Lemström, N. Mikkilä, and V. Mäkinen. Filtering methods for content-based retrieval on indexed symbolic music databases. *Journal of Information Retrieval*, 13(1):1–21, 2010.

[6] C. Romming and E. Selfridge-Field. Algorithms for polyphonic music retrieval: The hausdorff metric and geometric hashing. In *Proc. ISMIR'07*, pp. 457–462, Vienna, September 2007.

[7] R. Typke. *Music Retrieval based on Melodic Similarity*. PhD thesis, Utrecht University, Netherlands, 2007.

[8] E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content-based music retrieval. In *Proc. ISMIR'03*, pp. 193–199, Baltimore, MA, October 2003.

[9] G. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proc. ISMIR'02*, pp. 283–284, Paris, October 2002.