Transposition and Time-Scaling Invariant Algorithm for Detecting Repeated Patterns in Polyphonic Music

Antti Laaksonen(✉), Kjell Lemström, and Otso Björklund

Department of Computer Science, University of Helsinki, Helsinki, Finland
ahslaaks@cs.helsinki.fi

Abstract. This paper presents an algorithm for the time-scaled repeated pattern discovery problem in symbolic music. Given a set of n notes represented as geometric points, the algorithm reports all time-scaled repetitions in the point set. The idea of the algorithm is to use an onset-time-pair representation of music, which reduces the musical problem of finding repeated patterns to the geometric problem of detecting maximal point sets where all points are located on one line. The algorithm works in $O(n^4 \log n)$ time, which is almost optimal because the size of the output can be $\Theta(n^4)$. We also experiment with the algorithm using real musical data, which shows that when suitable heuristics are used to restrict the search, the algorithm works efficiently in practice and is able to find small sets of potentially interesting repeated patterns.

Keywords: Music pattern discovery · Transposition and time-scaling invariance · Geometric algorithms

1 Introduction

In this paper we consider the problem of finding repetitions in Western, equal tempered, polyphonic music. Various kinds of repetitions are frequent both in pop and in classical music. For example, already the structure of a pop song is often based on repetitions such as the usual ABABCBB structure. However, here we concentrate on repetitions taking place at the note level. In classical music one can find various forms of such repetitions, e.g., themes, motifs, imitations, drones, pedals and Alberti basses. Heinrich Schenker [14] stated already in 1954, that repetitions form the basis for music as an art. In general, repetitions make it easier for listeners to detect and remember musical ideas [10].

As the musical repetitions tend to appear in different pitches (different perceived height), it is important to apply transposition invariance in the searching process. If the repetitions are searched for in monophonic music, i.e., music with just one voice, or within a single voice, string-based algorithms can be used efficiently for the task (see e.g. [1,3,4]). Combining transposition invariance with polyphonic music – where repetitions may be scattered around distinct instrument or voices – makes a complex problem setting for which solutions based on

M. Montiel et al. (Eds.): MCM 2022, LNAI 13267, pp. 168–179, 2022.
https://doi.org/10.1007/978-3-031-07015-0_14

Table 1. A taxonomy of musical pattern matching and repeated pattern discovery problems that shows the gap filled by this paper. The table shows associated literature and the best known solutions for the problems.

Problem	Time complexity
Exact pattern matching	
Plain	$O(nm)$ [15]
Time-scaled	$O(n^2m)$ [5]
Time-warped	$O(n(m + \log n))$ [5]
Partial pattern matching	
Plain	$O(nm \log m)$ [15]
Time-scaled	$O(n^2m^2 \log n)$ [7]
Time-warped	$O(n^2m^2 \log n)$ [8]
Repeated pattern discovery	
Plain	$O(n^2 \log n)$ [11]
Time-scaled	$O(n^4 \log n)$ (new)
Time-warped	$O(n^2 \log n)$ [6]

linear string representations are not sufficient but a geometric point set (pitch-against-time) representation can be used effectively [11]. Furthermore, the data may be acquired by converting audio data into symbolic form where any voice information is lost during the process. In the sequel, we shall concentrate on the more general and complex problem and, therefore, use the point set representation. An example of the point set representation with real musical repetitions that are transposed and time-scaled is given in Fig. 5.

Let us denote by S a two-dimensional point set that represents a musical work (or several musical works concatenated one after another). The number of points in S is denoted by n. If a pattern in S is moved vertically or horizontally, it is *transposed* or *time-shifted*, respectively. A *translated* pattern may be both transposed and time-shifted; we call this the *plain* case. Moreover, a translated pattern may also be *time-warped*, in which case the translated points are time-shifted by some order-preserving, individual amount. *Time-scaling* is a special case of time-warping where time-shifting is applied using a universal multiplication factor to the onset times of the points.

The problem of finding repeating musical patterns is closely related to the musical pattern matching problem where the occurrences of a pattern P of m points are searched for in S. The matching may be *exact* or *partial*, meaning that all or only some of the points in P have to appear in a match, correspondingly. There are algorithms for the pattern matching problem for the plain translated case [13,15], time-scaled case [5,7,13] and time-warped case [5,8]. For the repeated pattern finding problem, there are algorithms for the plain [11] and time-warped case [6], but to our best knowledge there is no published algorithm

for the time-scaled case which is discussed in this paper. Table 1 summarizes the literature and the best known time complexities for all these problems.

Rather interestingly, time-scaled problems seem to be harder than time-warped problems, although the invariance needed for the latter problems is stronger [9]. The reason for this is that we can use dynamic programming to efficiently solve time-warped problems that consists of independent subproblems. However, it seems difficult to use a dynamic programming based approach in time-scaled problems because they have a global scaling factor that affects all subproblems.

As stated above, time-scaling is actually a special case of time-warping. However, the stronger the invariance, the more false positives the searching algorithms produce, i.e., using a time-warping algorithm for our problem would require a separate post-processing phase to filter out the vast majority of time-warped but non-time-scaled occurrences. We expect this to be much more tedious and time consuming than what we present in this paper.

The paper is organized as follows: In Sect. 2, we define the time-scaled repeated pattern discovery problem, show a lower bound for the output size of the algorithm, and present a simple algorithm for the problem. In Sect. 3, we describe our $O(n^4 \log n)$ algorithm for the problem and discuss heuristics that can be used with the algorithm. In Sect. 4, we study the efficiency of the algorithm and the effect of the heuristics. Finally, in Sect. 5, we present our conclusions.

2 Problem Definition

The input for the problem is a two-dimensional point set S that consists of n real-valued points. Each point $p \in S$ corresponds to a musical note: the coordinates $p.x$ and $p.y$ denote the onset time and pitch of the note, respectively.

Given a real number α (time-scaling factor) and a real-valued vector v (translation vector), let

$$\mathrm{MTTP}(\alpha, v) = \{p \mid p \in S, (\alpha p.x + v.x, p.y + v.y) \in S\}$$

denote a *maximal time-scaled translatable pattern* which corresponds to a repeated time-scaled pattern in music. For example, if $\alpha = 2$, the duration of each note is doubled in the repetition.

The problem discussed in this paper is to create an algorithm that discovers and reports all MTTPs in a point set. However, to make the problem more meaningful, we have two restrictions in the search. First, we only report patterns where $\alpha \geq 1$ because if $\alpha \neq 1$, any pattern can be represented in two ways using scaling factors α and $1/\alpha$. In addition, we only consider patterns that have two points with different x values. If this is not the case, the problem would not be well-defined because there would be an infinite number of possible (α, v) pairs.

Note that if $\alpha = 1$, $\mathrm{MTTP}(\alpha, v)$ corresponds to a two-dimensional maximal translatable pattern $\mathrm{MTP}(v)$ that can be found using the SIA algorithm [11].

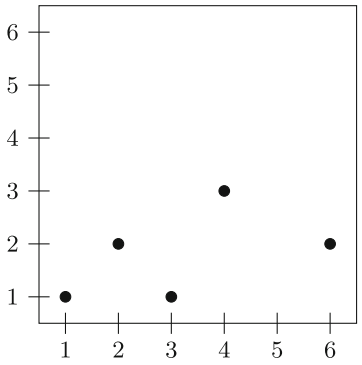

Fig. 1. Example point set

2.1 Example

As an example, consider a point set

$$S = \{(1,1),(2,2),(3,1),(4,3),(6,2)\},$$

shown in Fig. 1. In this case the patterns are as follows:

- $\mathrm{MTTP}(3/2,(0,-1)) = \{(2,2),(4,3)\}$
- $\mathrm{MTTP}(5/3,(-4,0)) = \{(3,1),(6,2)\}$
- $\mathrm{MTTP}(2,(0,1)) = \{(1,1),(2,2),(3,1)\}$
- $\mathrm{MTTP}(5/2,(-4,-1)) = \{(2,2),(4,3)\}$
- $\mathrm{MTTP}(3,(-8,0)) = \{(3,1),(4,3)\}$
- $\mathrm{MTTP}(3,(0,0)) = \{(1,1),(2,2)\}$
- $\mathrm{MTTP}(5,(-4,0)) = \{(1,1),(2,2)\}$

Note that there are two ways to produce the patterns $\{(2,2),(4,3)\}$ and $\{(1,1),(2,2)\}$ using two distinct (α, v) combinations.

2.2 Lower Bound

Next we show that the size of the output of the algorithm can be $\Theta(n^4)$, which means that any algorithm for the problem requires $\Omega(n^4)$ time in the worst case.

Consider a point set

$$S = \{(1,1),(2,1),(3,1),\ldots,(n,1)\}$$

where each note has the same pitch. There are $\Theta(n^4)$ ways to select four distinct points $p_1 < p_2 < q_1 < q_2$ such that $p2 - p1 \leq q2 - q1$. In each such case we have found two points p_1 and p_2 with distinct x values that have a repetition (points q_1 and q_2) with scaling $a \geq 1$, which means that the algorithm reports them. In addition, for fixed p_1 and p_2, each repetition has a distinct pair (α, v), so p_1 and p_2 are reported separately for each case. Thus, we have found a construction where the size of the output is $\Theta(n^4)$.

2.3 Simple Algorithm

Before presenting our actual algorithm, an interesting question is what would be a simple brute force algorithm for the problem.

Even creating such an algorithm is not trivial, but one possible idea is to go through all subsets of four points $p_1, p_2, q_1, q_2 \in S$ where $p_1.x < p_2.x$ and $q_1.x < q_2.x$. If $p_2.y - p_1.y = q_2.y - q_1.y$, we have found a potential repeating pattern and can define

$$\alpha = (q_2.x - q_1.x)/(p_2.x - p_1.x)$$

and

$$v = (q_1.x - ap_1.x, q_1.y - p_1.y).$$

This corresponds to a nonempty pattern MTTP(α, v), which will be reported if $\alpha \geq 1$ and the pattern contains two points with different x values. Note that the algorithm can generate a pair (α, v) several times and should only process the first occurrence.

This algorithm works in $O(n^5 \log n)$ time, because there are $O(n^4)$ subsets of four points, and for each subset it takes $O(n \log n)$ time to go through the points in S and find the points that belong to the corresponding pattern. The algorithm can be used to process small data sets, but there is no obvious way to improve it.

3 Algorithm Description

In this section, we describe an $O(n^4 \log n)$ time algorithm for the time-scaled pattern discovery problem. The algorithm uses the onset-time-pair representation presented in [6], and it reduces the problem of finding time-scaled repetitions into the problem of finding all maximal point sets where the points are located on one line.

The algorithm forms for each possible transposition a set C_i ("canvas") which consists of point pairs in S whose pitch interval is i. Since each point pair in a canvas has a constant pitch interval, it is enough to encode the onset times of the pair: $(x_1, x_2) \in C_i$ means that there are two points in S with onset times x_1 and x_2 and interval i. Now, a maximal set of points on the same line corresponds to a a maximal time-scaled translatable pattern (MTTP) whose time-scaling factor is the slope of the line (see Fig. 2).

More formally, given a set S of n points, the algorithm creates a collection of sets where each set is of the form

$$P_i = \{(a, b) \mid a \in S, b \in S, b.y - a.y = i\},$$

i.e., it contains all note pairs (a, b) whose interval $b.y - a.y$ is a constant i. Then, the algorithm generates for each set P_i an onset-time-pair representation

$$C_i = \{(a.x, b.x) \mid (a, b) \in P_i\}.$$

whose each point consists of x coordinates of a note pair in P_i. This representation is useful when finding time-scaled repetitions, because each repetition corresponds to a set of points that are on the same line. Thus, the remaining problem is to detect all such maximal point sets.

Let us assume that a set C_i consists of k notes. We can find all maximal point sets in $O(k^2 \log k)$ time as follows. We go through all point pairs $p_1, p_2 \in C_i$ where $p_1.x < p_2.x$ and $p_1.y < p_2.y$, and calculate for each such pair two values: a *slope*

$$s = \frac{p_2.y - p_1.y}{p_2.x - p_1.x}$$

of the line defined by the points, and an *offset*

$$z = p_1.y - p_1.x \cdot s,$$

which corresponds to the y coordinate where the line would intersect with the y axis. Then, the triples (s, z, p_1) and (s, z, p_2) are added to a list. After processing all $O(k^2)$ point pairs, we sort the list in $O(k^2 \log k)$ time, and after that, all points that are on the same line are next to each other in the list and we can detect them in $O(k^2)$ time.

Note that there is a direct correspondence between the parameters of a maximal translatable pattern and the parameters of a line in the onset-time-pair representation. Each pattern with parameters (α, v) corresponds to a line in $C_{v.y}$ so that the slope of the line is $s = \alpha$ and offset of the line is $z = v.x$.

Since the total number of points in C_i sets is $O(n^2)$, the algorithm works in $O(n^4 \log n)$ time.

3.1 Example

Consider again the point set

$$S = \{(1,1), (2,2), (3,1), (4,3), (6,2)\},$$

shown in Fig. 1. Let us focus on repetitions whose interval is 1 which can be found by creating the sets

$$P_1 = \{((1,1),(2,2)), ((1,1),(6,2)), ((2,2),(4,3)), \\ ((3,1),(2,2)), ((3,1),(6,2))\}$$

and

$$C_1 = \{(1,2), (1,6), (2,4), (3,2), (3,6)\}.$$

In this case, the points $(1,2)$, $(2,4)$ and $(3,6)$ are on the same line (Fig. 2) with slope 2 and offset 0. This point set corresponds to

$$\mathrm{MTTP}(2, (0,1)) = \{(1,1), (2,2), (3,1)\}.$$

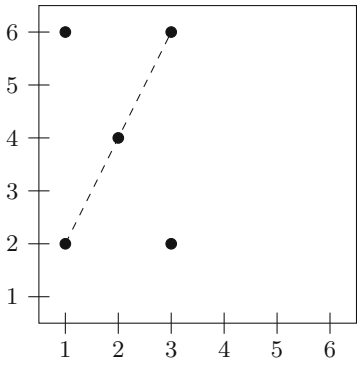

Fig. 2. Points $(1, 2)$, $(2, 4)$ and $(3, 6)$ are on the same line in the onset-time-pair representation C_1. This corresponds to MTTP$(2, (0, 1))$.

3.2 Filtering Repetitions

Since the algorithm typically produces a large number of repetitions, we can add heuristics (based on musical knowledge) to improve the results of the algorithm (see e.g. [1,2,12]). In this paper, we consider the following heuristics:

Inter-onset-Intervals. The inter-onset-intervals between two consecutive notes in a musical pattern cannot be large. Thus, when processing a set C_i in the algorithm, we can choose a constant max_d and only consider pairs $p_1, p_2 \in C_i$ where

$$p_2.x - p_1.x < max_d$$

and

$$p_2.y - p_1.y < max_d.$$

This heuristic can improve the running time of the algorithm, because it can be applied when generating point pairs for the onset-time-pair representation.

Pattern Properties. Most of the patterns found by the algorithm are usually short, while musically interesting repetitions are likely longer. For this reason, we can choose a constant min_n and only report patterns that have at least min_n notes.

In addition, notes in musically interesting patterns typically have several different pitches, so we can choose a constant min_p and only report patterns that have at least min_p different pitches.

Scaling Factors. Since we are interested in time-scaled repetitions, we can focus on repetitions where $\alpha \neq 1$. When combined with other heuristics, this can greatly reduce the number of patterns reported by the algorithm.

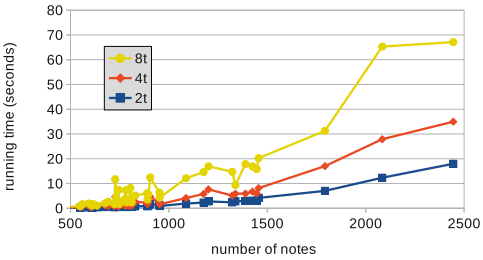

Fig. 3. Efficiency of the algorithm for max_d values $2t$, $4t$ and $8t$ where t denotes the number of ticks in a beat.

4 Experiments

In this section, we study the efficiency and usefulness of our algorithm on real musical data. We have implemented the algorithm in C++, and verified that it produces correct results. The implementation is available in our GitHub repository (https://github.com/c-brahms/time-scaled-repeated).

The data set used in the experiments consists of 48 MIDI files: 24 preludes and fugues from the first book of Bach's *Das wohltemperierte Klavier*. We converted each file into a point set where onset times are MIDI time values (ticks) and pitches are MIDI note numbers. The number of note events in a file ranges from about 400 to 2500.

We conducted the experiments using a 1.8 GHz Intel Core i7 computer in a Linux environment. In all experiments we searched for patterns where $\alpha \neq 1$, i.e., time-scaling is used in the repetition.

4.1 Efficiency

In the first experiment, we measured the running time of the algorithm for each file. It turned out that the general algorithm without the max_d parameter would be too slow for processing the files, so we only consider tests where the max_d parameter is used. The other filtering parameters only control the reporting after the search, so they do not affect the efficiency of the algorithm.

Figure 3 shows the results of the experiment. Three max_d values were used: $2t$, $4t$ and $8t$ where t denotes the number of ticks in a beat. As expected, the greater the max_d value, the slower the algorithm. In most cases, the processing time was less than two seconds, and the maximum processing time was 67 s for the largest input when the value $max_d = 8t$ was used.

The experiment shows that the algorithm can process real music files efficiently. While the time complexity $O(n^4 \log n)$ of the algorithm could indicate that it is of limited practical use, the max_d parameter considerably improves the practical efficiency of the algorithm. On the other hand, if the max_d parameter is not used, the algorithm can only be used for small inputs.

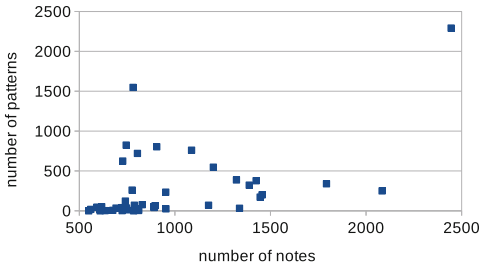

Fig. 4. The number of notes and discovered patterns in each file ($max_d = 4$, $min_n = 8$, $min_p = 5$).

4.2 Pattern Discovery

In the second experiment, we examined the patterns discovered by the algorithm. In this experiment, we used parameters $max_d = 4$, $min_n = 8$, and $min_p = 5$, i.e., the maximum inter-onset-interval is 4 ticks and the pattern must have at least 8 notes and 5 distinct pitches. We chose the parameters so that they filter musically interesting patterns and produce a sufficient number of results.

Figure 4 shows the results of the experiment. The x axis shows the number of notes in each file, and the y axis shows the number of discovered patterns. In most cases, the number of discovered patterns is small and it would be possible to check them all manually.

In almost all discovered patterns, the scaling factor α is one of 4/3, 3/2, 2, 3, and 4. This is not surprising because there are no tempo changes in our data set, and such scaling factors are also expected in real musical repetitions. This suggests that in some cases we could also only focus on finding repetitions whose scaling factors belong to a constant set and achieve an $O(n^2 \log n)$ time algorithm by using a standard pattern discovery algorithm several times. However, such a search would not find repetitions with unexpected scaling factors.

While the used heuristics reduce the number of results, it seems that most of the discovered patterns are still not musically interesting. A possible additional heuristic would be to somehow restrict the intervals between consecutive pattern notes. However, it seems to be difficult to add such a heuristic to the algorithm because the intervals are ignored in the onset-time-pair representation which is the main building block of the algorithm.

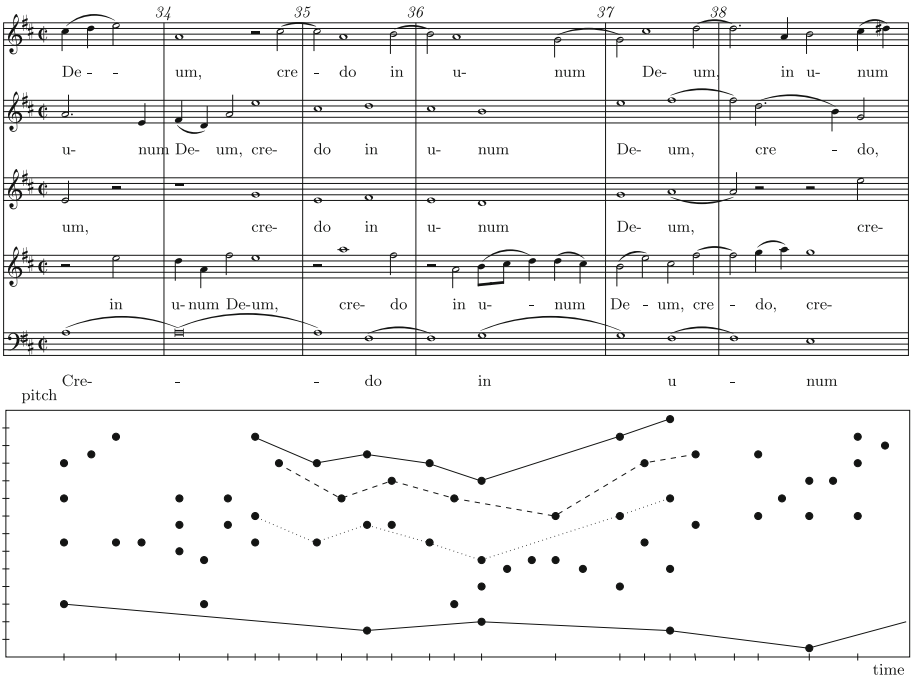

Fig. 5. An extract of the choir parts of J.S. Bach's *Mass in B minor* (*Credo in unum Deum*) and the corresponding, synchronized point set representation where each note-on is represented by a point. The theme is introduced in the beginning of the movement. In the depicted measures the fireworks start: the theme has four partly overlapping occurrences in different keys. Our algorithm detects also the repetition in the bass voice (continued beyond the illustrated area) although it has twice the duration of the other occurrences.

5 Conclusions

In this paper, we have presented an $O(n^4 \log n)$ time algorithm for solving the time-scaled repeated pattern discovery problem in symbolic music. The presented algorithm is more efficient than an $O(n^5 \log n)$ time brute force approach, and it is almost an optimal algorithm because any algorithm for the problem requires $\Theta(n^4)$ time.

Our algorithm can be seen as a missing piece in the taxonomy of pattern matching and discovery algorithms in symbolic music. Exact and time-warped algorithms have been proposed for both pattern matching and discovery, but time-scaled algorithms have only been used in pattern matching. Like in pattern matching, the time-scaled problem is the most difficult also in pattern discovery.

Based on our experiments, our algorithm can be quite efficient in practice when some heuristics are used to filter interesting musical patterns. While there are theoretical constructions where the output size is $\Theta(n^4)$, the number of interesting patterns in actual musical data is much smaller and we can find them

efficiently by implementing the algorithm so that it avoids creating patterns that are not musically meaningful.

It is an interesting question whether the $O(n^4 \log n)$ time complexity of the algorithm could be improved. Since we use onset-time-pair representations and reduce the problem to a geometric problem of detecting all maximal sets of points that are on the same line, one way to that end would be to solve the geometric problem more efficiently. The problem at hand is somewhat easier than the general geometric problem: while the total number of points in onset-time-pair representations is $O(n^2)$, the number of *distinct* x and y values is only $O(n)$. In the future, we will study if we can use this observation to improve the algorithm and process the points in groups that have the same x or y coordinate.

References

1. Cambouropoulos, E.: Musical parallelism and melodic segmentation: a computational approach. Music Percept. **3**(23), 249–268 (2006)
2. Collins, T., Arzt, A., Flossmann, S., Widmer, G.: SIARCT-CFP: improving precision and the discovery of inexact musical patterns in point-set representations. In: Proceedings of the 14th International Conference on Music Information Retrieval (ISMIR 2013), pp. 549–554 (2013)
3. Hsu, J.L., Liu, C.C., Chen, A.: Discovering nontrivial repeating patterns in music data. IEEE Trans. Multimed. **3**(3), 311–325 (2001)
4. Knopke, I., Jürgensen, F.: A system for identifying common melodic phrases in the masses of Palestrina. J. New Music Res. **38**, 171–181 (2009)
5. Laaksonen, A.: Efficient and simple algorithms for time-scaled and time-warped music search. In: Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research, pp. 621–630 (2013)
6. Laaksonen, A., Lemström, K.: Discovering distorted repeating patterns in polyphonic music through longest increasing subsequences. J. Math. Music **15**(2), 99–111 (2021)
7. Lemström, K.: Towards more robust geometric content-based music retrieval. In: Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010), pp. 577–582 (2010)
8. Lemström, K., Laitinen, M.: Transposition and time-warp invariant geometric music retrieval algorithms. In: Proceedings of the 2011 International Conference on Multimedia and Expo (ICME 2011), pp. 1–6 (2011)
9. Lemström, K., Wiggins, G.: Formalizing invariances for content-based music retrieval. In: Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR 2009), pp. 591–596 (2009)
10. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. MIT Press, Cambridge (1983)
11. Meredith, D., Lemström, K., Wiggins, G.: Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. J. New Music Res. **31**(4), 321–345 (2002)
12. Nieto, O., Farbood, M.M.: Perceptual evaluation of automatically extracted musical motives. In: Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012), pp. 723–727 (2012)

13. Romming, C.A., Selfridge-field, E.: Algorithms for polyphonic music retrieval: the Hausdorff metric and geometric hashing. In: Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007), pp. 457–462 (2007)
14. Schenker, H.: Harmony. University of Chicago Press, London (1954)
15. Ukkonen, E., Lemström, K., Mäkinen, V.: Geometric algorithms for transposition invariant content-based music retrieval. In: Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003), pp. 193–199 (2003)