Understanding and Compressing Music with Maximal Transformable Patterns

David Meredith[0000−0002−9601−5017]

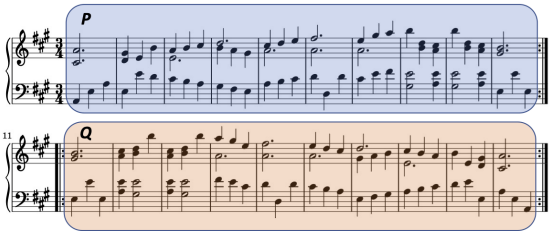
Aalborg University, Aalborg, Denmark dave@create.aau.dk
https://vbn.aau.dk/en/persons/119171
https://www.titanmusic.com

Abstract. We present a polynomial-time algorithm that discovers all maximal patterns in a point set, $D \subset \mathbb{R}^k$, that are related by transformations in a user-specified class, F, of bijections over $\mathbb{R}^k$. We also present a second algorithm that discovers the set of occurrences for each of these maximal patterns and then uses compact encodings of these occurrence sets to compute a losslessly compressed encoding of the input point set. This encoding takes the form of a set of pairs, $E = \{\langle P_1, T_1\rangle, \langle P_2, T_2\rangle, \ldots \langle P_\ell, T_\ell\rangle\}$, where each $\langle P_i, T_i\rangle$ consists of a maximal pattern, $P_i \subseteq D$, and a set, $T_i \subset F$, of transformations that map P_i onto other subsets of D. Each transformation is encoded by a vector of real values that uniquely identifies it within F and the length of this vector is used as a measure of the complexity of F. We evaluate the new compression algorithm with three transformation classes of differing complexity, on the task of classifying folk-song melodies into tune families. The most complex of the classes tested includes all combinations of the musical transformations of transposition, inversion, retrograde, augmentation and diminution. We found that broadening the transformation class improved performance on this task. However, it did not, on average, improve compression factor, which may be due to the datasets (in this case, folk-song melodies) being too short and simple to benefit from the potentially greater number of pattern relationships that are discoverable with larger transformation classes.

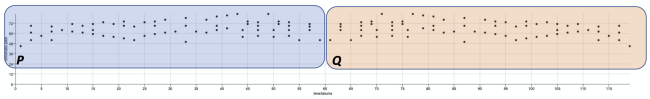
Keywords: Pattern discovery · Music analysis · SIA · Maximal transformable patterns · Parsimony principle · Point-set patterns · Data compression.

1 Introduction

Suppose we want to understand a set of observations, D, in the form of a set of n, k-dimensional points—that is, $D \subset \mathbb{R}^k$. We call D a *dataset* and we want an explanation for D, preferably in the form of an algorithm that outputs D. There may be many different algorithms that output a given dataset. However, we expect only a small subset of these to faithfully describe the *actual* process that gave rise to the dataset. In science, to decide between two or more competing theories that account equally well for a set of observations, we often apply



(a) Score



(b) Point set

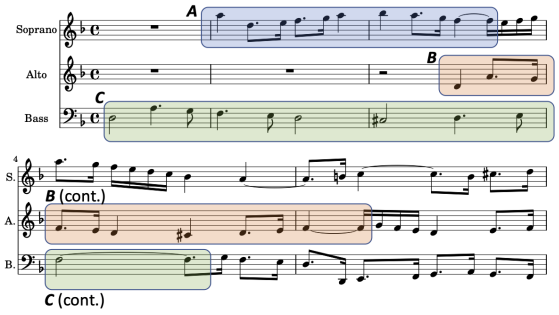
Fig. 1: Second movement of J. Haydn's Sonata in A Major, Hob.XVI:26, Op. 13 No. 6 ("Menuetto al Rovescio"). The second half of the piece, Q, is an exact retrograde of the first half, P. (b) shows a point set representing the score in (a), where each point represents a note, with the x-value indicating the temporal midpoint of the note and the y-value representing its chromatic pitch [11, p. 127].

the parsimony principle and favour the simplest explanation, according to some measure of complexity. We believe that, all else being equal, a simple explanation that accurately accounts for some observed data is less likely to do so by chance than an equally accurate, but more complex, explanation. We therefore want to find one of the *simplest* algorithms that outputs D. We assume the dataset is given to us in the form of an *extensional* description in which each coordinate of each point is explicitly listed. Such a description represents the dataset as if it were a meaningless collection of unrelated observations. A description of a dataset never needs to be longer than its extensional description.

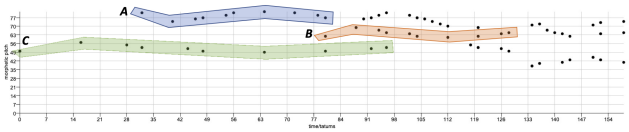
Suppose there is a set of points, P, in our dataset, D, that is mapped by a transformation, f, onto another set of points, Q, in D—that is, $Q = f(P)$. We call P and Q *patterns* in D. We can now describe $P \cup Q$ by extensionally describing P and then specifying f instead of additionally extensionally describing Q. If f can be described using fewer bits than are needed to extensionally describe Q, then a description in terms of P and f is a compressed encoding of $P \cup Q$. Moreover, by describing $P \cup Q$ as the result of applying the transformation f to P, we make progress towards *explaining* $P \cup Q$, because we have now described how Q might have arisen by transforming P.

Now suppose that $D \subset \mathbb{R}^2$ and that each point, $\langle t, p \rangle$, in D represents a note (or a sequence of tied notes) in a musical score, with t representing the time at which the note occurs and p representing its pitch. As an example, Fig. 1 shows the "Menuetto al Rovescio" from Haydn's Sonata in A major, Hob.XVI:26. Note that pattern Q in Fig. 1(b) can be obtained by reflecting P in the line $t = 0$ (i.e.,

finding its retrograde) and then translating it parallel to the time axis. The full movement can be described by extensionally describing P and then describing the transformation that maps P onto Q. This gives a compressed description of the piece as well as an explanation for its second half.



(a) Score



(b) Point set

Fig. 2: Bars 1–5 of *Contrapunctus VI* from J. S. Bach's *Die Kunst der Fuge*, BWV 1080. In (a) pattern B is an inversion of pattern A and pattern C is an augmentation of pattern B. (b) shows a point set representing the passage, with each note or sequence of tied notes represented by a single point, $\langle t, p \rangle$, where t represents the onset time of the note and p is its morphetic pitch [11, p. 127]. In (b), pattern B can be obtained from pattern A by a reflection in the time axis followed by a translation, while pattern C can be obtained from pattern B by stretching parallel to the time axis.

Figure 2 shows another case where discovering simple geometric relationships between patterns is essential to understanding the music. In this extract from the opening of *Contrapunctus VI* from J. S. Bach's *Die Kunst der Fuge*, BWV 1080, pattern B is an inversion of pattern A, and pattern C is an augmentation of pattern B. The set of points, $A \cup B \cup C$, can be described succinctly by extensionally defining A and then specifying the transformations that map A onto B and C. This example illustrates that we can encode such a set of interrelated pattern occurrences as a pair, $\langle P, T \rangle$, where P is an extensionally encoded pattern occurrence and T is a set of transformations that map P onto other occurrences of P in the dataset. If each transformation in T can be described using fewer bits than would be required to extensionally describe the occurrence that it generates from P, then we get a compressed encoding that we expect will

represent a better way of understanding the data than that represented by an uncompressed, extensional description.

Given a set of bijections, F, over $\mathbb{R}^k$, we say that a pattern, P, is *transformable* within a dataset, D, with respect to F, if there exists a transformation $f \in F$ such that $f(P) \subseteq D$. Because we believe in the parsimony principle, our goal is to find losslessly compressed encodings of datasets that are as short as possible, so we are primarily interested in finding *maximal* transformable patterns: we define a transformable pattern to be *maximal* within a dataset, D, for a transformation, $f \in F$, if there is no other pattern, $Q \subseteq D$, such that $f(Q) \subseteq D$ and $P \subset Q$.

In this paper, we present an algorithm that discovers the non-empty maximal transformable patterns (MTPs) in a dataset, $D \subset \mathbb{R}^k$, for transformations in a class, F, of bijections over $\mathbb{R}^k$. We also present an algorithm that computes a losslessly compressed encoding of D in the form of a set of pairs, $\{\langle P_1, T_1\rangle, \langle P_2, T_2\rangle, \ldots \langle P_\ell, T_\ell\rangle\}$, where each P is an MTP in D with respect to F, and T is the set of transformations in F that map P onto subsets of D.

In the next section, we review some related previous work. Then, in Section 3, we present some concepts and theory needed to understand the new algorithms, which will be described in Section 4. These algorithms could be used in a wide range of musical applications, including, for example, thematic and formal analysis, algorithmic composition, genre classification and composer recognition. In Section 5, we present the results of an experiment in which we use the algorithms to automatically classify folk songs into tune families. Finally, in Section 6, we summarise the main contributions of this paper and suggest possible directions for future research.

2 Related work

Meredith, Lemström and Wiggins [15] defined a *maximal translatable pattern* for a vector, v, in a point set, D, to be the set of points in D that are mapped to other points in D when translated by v. They presented an algorithm, SIA, that computes the maximal translatable patterns in a set of n, k-dimensional points in $O(kn^2 \log n)$ worst-case time and $O(kn^2)$ space. They defined two patterns to be *translationally equivalent* if and only if one can be obtained from the other by translation alone. They defined two patterns to be in the same *translational equivalence class* (TEC) if and only if they are translationally equivalent. They also presented an algorithm, SIATEC, that computes the TEC for each maximal translatable pattern in a set of n, k-dimensional points in $O(kn^3)$ time and $O(kn^2)$ space. More recently, Laaksonen and Lemström [9] presented a sweepline algorithm for finding maximal translatable patterns that has the same running time as SIA but requires only $O(kn)$ space.

In [16], Meredith, Lemström and Wiggins introduced an algorithm called COSIATEC that iteratively uses SIATEC to compute a compressed description of a point set in the form of a set of TECs that collectively cover the input dataset. COSIATEC achieves compression by encoding each TEC as an ordered

pair, $\langle P, V \rangle$, where P is one occurrence of the TEC's pattern and V is the set of vectors that map P onto its other occurrences in the dataset. If T is a TEC and C is the union of the occurrences in T, then the compression factor achieved by encoding T in the form $\langle P, V \rangle$, as just described, is approximately $|C|/(|P|+|V|)$. C is the TEC's *covered set* and $|C|$ is the *coverage* of the TEC.

Collins et al. [4] claimed that maximal translatable patterns commonly consist of a musically important pattern along with other isolated points. To solve this problem, they proposed a "compactness trawling" technique that involves searching within each maximal translatable pattern for subsets that are larger than some specified minimum size and more compact than some minimum compactness threshold. They combined this technique with SIA to produce a new variant of the algorithm called SIACT.

In an attempt to improve on the precision and running time of SIA, Collins [1] designed a modified version of SIA called SIAR. Whereas SIA computes inter-point vectors from each point to every lexicographically later point, SIAR only computes the inter-point vectors from each point to the points that are not more than r steps lexicographically later than it in the dataset, where r is a parameter of the algorithm. This is approximately equivalent to running SIA with a moving window.

In [2], Collins et al. address what they call the *inexactness problem* and the *precision problem*. The *inexactness problem* is that the basic versions of SIA, SIATEC and COSIATEC are limited to only discovering exact occurrences of patterns under translation. The *precision problem* is that SIA and SIATEC typically discover many patterns in a dataset of which usually only a small fraction are of interest. In COSIATEC, this problem is addressed by using heuristics (such as compactness, compression ratio and coverage) to select only the "best" TEC returned on each run of SIATEC. Collins et al. [2] propose an algorithm for computing inexactly related patterns, called SIARCT-CFP, in which they combine SIAR and SIACT with a categorisation and fingerprinting (CFP) technique. Collins et al. [3] used SIARCT-CFP to discover interesting instances of thematic and harmonic re-use in Beethoven's piano sonatas.

Forth [6] presented an algorithm that, like COSIATEC, generates an encoding in the form of a set of TECs whose occurrences collectively cover the input point set. However, unlike COSIATEC, Forth's algorithm runs SIATEC only once and the covered sets of the TECs in these encodings may intersect. Forth's algorithm assigns a "structural salience" to each of these TECs and then attempts to construct a cover consisting of structurally highly salient TECs.

In [12], Meredith presented a compression algorithm based on SIATEC, called SIATECCompress. Like COSIATEC and Forth's algorithm, SIATECCompress computes a compressed encoding of a point set in the form of a set of maximal translatable pattern TECs that collectively cover the point set. Like Forth's algorithm, but unlike COSIATEC, SIATECCompress runs SIATEC only once and computes covers in which the TEC covered sets may intersect. Like COSIATEC, SIATECCompress prefers TECs that individually have better compression factors, larger covered sets and more compact patterns. Typically,

the encodings generated by SIATECCompress are not as compressed as those computed by COSIATEC. On the other hand, SIATECCompress is faster than COSIATEC because it only runs SIATEC once.

Meredith [13] compared COSIATEC, SIATECCompress and Forth's algorithm on three musicological tasks: classifying folk tunes into tune families, discovery of repeated themes and sections, and detection of fugal subjects and countersubjects. He also evaluated the effect of using SIAR and/or SIACT with each of the three basic algorithms. On the folk-song classification task, COSIATEC performed best, while variants of SIATECCompress performed best on the other two tasks. More recently, in [14], two techniques were presented for improving the compression factor achievable using any TEC cover algorithm (such as COSIATEC, SIATECCompress or Forth's algorithm). The first of these techniques, *removal of redundant translators* (RRT), involves analysing each TEC and removing as many occurrences as possible without reducing the TEC's covered set. The second technique, embodied in an algorithm called RecurSIA, is to recursively apply the selected TEC cover algorithm to the patterns in each TEC and to replace the extensional description of each pattern in each $\langle P, V \rangle$ pair with a compressed encoding in terms of maximal translatable pattern TECs discovered *inside* that pattern.

3 Patterns and transformations in point sets

In this section we define some concepts required in order to understand the algorithms presented in Section 4. We assume we are provided with a point set, $D \subset \mathbb{R}^k$, that we call a *dataset*. We use the term *pattern* to refer to any set, $P \subset \mathbb{R}^k$, usually (but not always) under consideration in the context of a specified dataset. In the context of a dataset, $D \subset \mathbb{R}^k$, we define a *transformation* to be a bijection, $f : \mathbb{R}^k \to \mathbb{R}^k$. If a transformation, f, maps a point p onto a point q, then we can write $q = f(p)$. Bijections have unique inverses, therefore the inverse, f^{-1}, of a transformation, f, is defined as follows: $f^{-1}(p) = q$ if and only if $f(q) = p$ for all $p, q \in \mathbb{R}^k$. If $S \subset \mathbb{R}^k$ and $f : \mathbb{R}^k \to \mathbb{R}^k$ is a transformation, then, for convenience, we define that $f(S) = \bigcup_{p \in S} \{f(p)\}$ and $f^{-1}(S) = \bigcup_{p \in S} \{f^{-1}(p)\}$.

3.1 Transformation classes

Given a dataset, $D \subset \mathbb{R}^k$, we define a *transformation class*, F, to be some specified set of transformations over $\mathbb{R}^k$. We define each transformation, $f \in F$, to have a distinct *parameter*, $\alpha_F(f)$, that uniquely identifies f within F. A transformation class, F, therefore has an associated *parameter set*, $A(F) = \bigcup_{f \in F} \{\alpha_F(f)\}$. We define a function, ϕ_F, such that, for each $f \in F$, there exists an expression, $\phi_F(\alpha_F(f), p)$, which is equal to $f(p)$ for all $p \in \mathbb{R}^k$. We call ϕ_F the *transformation class function* for the class F. We can therefore define any transformation class F as follows:

$$F = \{f : (\exists \alpha \in A(F) : \forall p \in \mathbb{R}^k, f(p) = \phi_F(\alpha, p))\}.$$

A transformation class, F, is therefore determined by its parameter set, $A(F)$, and its transformation class function, ϕ_F.

As an example, let $F_{2\mathrm{T}}$ be the set of 2-dimensional translations. If $D \subset \mathbb{R}^2$ represents a passage of music like the datasets in Figs. 1 and 2, then $F_{2\mathrm{T}}$ contains the transformations that map patterns onto exact (but possibly transposed) restatements of those patterns. Each transformation, $f \in F_{2\mathrm{T}}$, is a function, $f : \mathbb{R}^2 \to \mathbb{R}^2$, that translates a point, $p \in \mathbb{R}^2$, by a vector, $v \in \mathbb{R}^2$. The vector v uniquely determines f within the transformation class, $F_{2\mathrm{T}}$. That is, for each $f \in F_{2\mathrm{T}}$, $\alpha_{F_{2\mathrm{T}}}(f) = v$. The set of transformation parameters for this transformation class, $A(F_{2\mathrm{T}})$, is therefore equal to $\mathbb{R}^2$ and the transformation class function is

$$\phi_{F_{2\mathrm{T}}}(\alpha_{F_{2\mathrm{T}}}(f), p) = p + \alpha_{F_{2\mathrm{T}}}(f)\,.$$

$F_{2\mathrm{T}}$ can therefore be defined to be the set

$$\{f : (\exists \alpha \in A(F_{2\mathrm{T}}) : \forall p \in \mathbb{R}^2, f(p) = \phi_{F_{2\mathrm{T}}}(\alpha_{F_{2\mathrm{T}}}(f), p))\}\,,$$

which reduces to

$$\{f : (\exists v \in \mathbb{R}^2 : \forall p \in \mathbb{R}^2, f(p) = p + v)\}\,.$$

Given a definition of the transformation class function, ϕ_F, for a transformation class, F, we can uniquely specify any transformation, $f \in F$, by giving the transformation parameter for f within F, $\alpha_F(f)$. A transformation parameter typically consists of some combination of vectors, matrices, or numerical values. However, it will always be possible to modify a transformation class function so that the transformation parameter is "serialized" as a vector of real values, that the modified class function first parses in order to construct the transformation parameter. We call such a vector a *transformation parameter vector* and we denote the transformation parameter vector corresponding to $\alpha_F(f)$ by $\sigma(\alpha_F(f))$. For notational convenience, we define that $\sigma(\alpha_F(f)) = \sigma_F(f)$. The vector, $\sigma_F(f)$, contains all and only the independent numerical values required to uniquely specify f within F. We define the *transformation class complexity*, $K(F)$, of a transformation class, F, to be the number of independent values in each of its transformation parameter vectors. That is, $K(F) = |\sigma_F(f)|$ for any $f \in F$. We define the *modified transformation class function*, ϕ'_F, to be the modified form of ϕ_F, the transformation class function, that first constructs the transformation parameter, $\alpha_F(f)$, from the transformation parameter vector, $\sigma_F(f)$, before applying ϕ_F. In other words, $\phi'_F(\sigma_F(f), p)$ is always equal to $\phi_F(\alpha_F(f), p)$. As a simple example, each transformation parameter for the class $F_{2\mathrm{T}}$ is a 2-dimensional vector, which is already simply a vector of two independent real values. So for each function, f, in $F_{2\mathrm{T}}$, we can define that $\sigma_{F_{2\mathrm{T}}}(f) = \alpha_{F_{2\mathrm{T}}}(f)$ and therefore $\phi_{F_{2\mathrm{T}}}$ is the same as $\phi'_{F_{2\mathrm{T}}}$ and $K(F_{2\mathrm{T}}) = 2$.

3.2 Transformable patterns, occurrences and occurrence sets

Given a dataset, $D \subset \mathbb{R}^k$, we define a pattern, $P \subseteq D$, to be *transformable* in D with respect to a transformation class, F, if and only if there exists a

transformation, $f \in F$, such that $f(P) \subseteq D$. If this is the case, we say that P is *transformable by* f in D and that $f(P)$ is an *occurrence of* P in D with respect to F. We define the *occurrence set* of P in D with respect to F to be the set,

$$OS(P, D, F) = \{P\} \cup \{f(P) : f(P) \subseteq D \text{ and } f \in F\} \ .$$

Note that this definition of the term, "occurrence set", is more general than that which has been used in the MIREX Competitions on Discovery of Repeated Themes and Sections [5]. In these competitions, an "occurrence set" contains only those occurrences of a pattern that are inter-related by *translation* within pitch–time space.

Given an occurrence set, $\mathcal{S} = OS(P, D, F)$, we define the *covered set* of $\mathcal{S}$ to be the set $COV(\mathcal{S}) = \bigcup_{P \in \mathcal{S}} P$. An occurrence set, $OS(P, D, F)$, can be precisely specified by providing an extensional description of P along with the transformation parameter of each transformation, $f \in F$, for which $f(P) \subseteq D$. This implies that, given F, we can describe $COV(OS(P, D, F))$ using at most $k|P| + K(F)(|OS(P, D, F)| - 1)$ independent values. This compares with the $k|COV(OS(P, D, F))|$ independent values required to describe $COV(OS(P, D, F))$ extensionally. Disregarding the constant length of a given definition of ϕ'_F, the *compression factor* that we can achieve with an occurrence set, $OS(P, D, F)$, is therefore at least

$$\frac{k|COV(OS(P, D, F))|}{k|P| + K(F)(|OS(P, D, F)| - 1)} \ .$$

This compression factor is achieved by encoding the covered set, $COV(OS(P, D, F))$, by the ordered pair,

$$\langle P, \{\sigma : \phi'_F(\sigma, P) \subseteq D \wedge \phi'_F(\sigma, P) \neq P\}\rangle \ ,$$

where $\phi'_F(\sigma, P)$ is a shorthand notation for $\bigcup_{p \in P}\{\phi'_F(\sigma, p)\}$. If we define

$$\Psi = \{\sigma : \phi'_F(\sigma, P) \subseteq D \wedge \phi'_F(\sigma, P) \neq P\} \ ,$$

then, given an encoding, $\langle P, \Psi\rangle$, the covered set itself can be reconstructed by evaluating the expression $P \cup \bigcup_{\sigma \in \Psi} \phi'_F(\sigma, P)$.

As an example, let $F_{2\mathrm{TR}}$ be those transformations that consist of a 2-dimensional translation, optionally followed by a reflection in the x-axis. If the dataset represents a score of a piece of music, where each point represents a note or sequence of tied notes, the x-axis represents onset time and the y-axis represents pitch, then $F_{2\mathrm{TR}}$ contains the transformations that map patterns onto exact repetitions, possibly inverted and/or transposed (e.g., the transformation that maps pattern A onto pattern B in Fig. 2). Each transformation, f, in $F_{2\mathrm{TR}}$ is uniquely identified by a parameter, $\alpha_{F_{2\mathrm{TR}}}(f)$ which is an ordered pair, $\langle v, b\rangle$, where v is a 2-dimensional translation vector and b, which must be either 1 or -1, is a scale factor for a stretch parallel to the y-axis, so that $b = -1$ indicates a reflection in the x-axis and $b = 1$ indicates no reflection. In this case, if $\alpha = \langle v, b\rangle$,

then we can define that $\sigma(\alpha) = \langle \alpha[0][0], \alpha[0][1], \alpha[1], \rangle$, so that $K(F_{2\mathrm{TR}}) = 3$. We can therefore define the modified transformation class function for this transformation class as follows: $\phi'_{F_{2\mathrm{TR}}}(\sigma, p) = \langle p[0] + \sigma[0], \sigma[2](p[1] + \sigma[1]) \rangle$.

As another example, consider the class of 2-dimensional transformations that consist of a scaling parallel to the x-axis, followed by a translation, optionally followed by a reflection in the x-axis. We denote this class of transformations by $F_{2\mathrm{STR}}$. Let $D \subset \mathbb{R}^2$, such that each point, $\langle t, p \rangle \in D$, represents a note or sequence of tied notes, with t representing the temporal mid-point of each note (or sequence of tied notes), and p representing its pitch. $F_{2\mathrm{STR}}$ then corresponds to the class of traditional contrapuntal transformations, consisting of any combination of transposition, inversion, retrograde, augmentation and diminution. $F_{2\mathrm{STR}}$ contains the transformations that are required to describe the relationships between the patterns identified in the examples in Figs. 1 and 2. Each transformation, f, in $F_{2\mathrm{STR}}$ is uniquely specified by a transformation parameter, $\alpha_{F_{2\mathrm{STR}}}(f) = \langle s, v, b \rangle$, where s is a non-zero real value representing a scale factor for a stretch parallel to the x-axis (positive or negative so as to include retrogrades), $v \in \mathbb{R}^2$ is a 2-dimensional vector and $b \in \{-1, 1\}$ is a scale factor for a stretch parallel to the y-axis, so that $b = -1$ indicates a reflection in the x-axis and $b = 1$ indicates no reflection. The transformation class function for $F_{2\mathrm{STR}}$ is

$$\phi_{F_{2\mathrm{STR}}}(\alpha, p) = \langle p[0]\alpha[0] + \alpha[1][0], \alpha[2](p[1] + \alpha[1][1]) \rangle \,.$$

If $\alpha = \alpha_{F_{2\mathrm{STR}}}(f)$ is the transformation parameter of a transformation, f, in $F_{2\mathrm{STR}}$, then we can define the corresponding transformation parameter vector to be

$$\sigma(\alpha) = \langle \alpha[0], \alpha[1][0], \alpha[1][1], \alpha[2] \rangle \,,$$

which implies that $K(F_{2\mathrm{STR}}) = 4$ and we can define the modified transformation class function for $F_{2\mathrm{STR}}$ as follows:

$$\phi'_{F_{2\mathrm{STR}}}(\sigma, p) = \langle p[0]\sigma[0] + \sigma[1], \sigma[3](p[1] + \sigma[2]) \rangle \,.$$

3.3 Maximal transformable patterns

Given a dataset, D, and a transformation, f, we define the *maximal transformable pattern* (MTP) in D for the transformation, f, to be the set of points, $M(D, f) = D \cap f^{-1}(D)$. It follows directly from this definition that $M(D, f^{-1}) = f(M(D, f))$. Note that this definition requires that a transformation f has a well-defined inverse and is therefore a bijection. The MTP for the transformation f in the dataset D, denoted by $M(D, f)$, is thus the set of points in D that are mapped by f onto points in D. We say that a transformation, f, *occurs* in a dataset, D, if and only if $M(D, f)$ is non-empty.

As explained in Section 1 above, we can make progress towards both compressing and understanding a dataset, D, by discovering pairs of patterns in the dataset related by transformations that belong to transformation classes whose complexities are low. Suppose that P and Q are patterns in a dataset, $D \subset \mathbb{R}^k$,

related by a transformation, f, in a transformation class, F, so that $P = f(Q)$. This allows us to describe $P \cup Q$ using $k|P| + K(F)$ independent values, compared with the $k|P \cup Q|$ values required to describe $P \cup Q$ extensionally, implying that we can potentially achieve a compression factor of $\frac{k|P \cup Q|}{k|P|+K(F)}$. For a given f, we therefore want to maximize $|P|$ and minimize $|P \cap Q|$—hence our interest in finding the *maximal* transformable pattern for each transformation, f, that occurs in a dataset.

There is a one-to-one correspondence between a maximal transformable pattern and a partial symmetry. A geometrical object has a particular type of symmetry if there is a transformation of that type (e.g., a reflection or a rotation) that maps the object onto itself. An object or point set, $D \subset \mathbb{R}^k$, is symmetrical (or invariant) with respect to some class of transformations, F, if there is a transformation, $f \in F$, such that $f(D) = D$. In this case, f defines a *total symmetry* on D. More generally, a transformation, $g \in F$, defines a *partial symmetry* on D if $g(D) \cap D \neq \emptyset$. It follows from the definition of an MTP given above, that a transformation, f, defines a partial symmetry on a dataset, D, if and only if its MTP in D, $M(D, f)$, is non-empty (i.e., iff f occurs in D). If $M(D, f) = D$, then f defines a total symmetry on D.

4 Algorithms for discovering maximal transformable patterns and computing compressed encodings

We now present an algorithm for computing maximal transformable patterns (MTPs), together with a second algorithm that uses maximal transformable patterns to compute a compact encoding of a point set. Our pseudocode follows the conventions used in a number of related studies, e.g. [13]. Briefly, block structure is indicated by indentation alone and the assignment operator is '$\leftarrow$'. We denote ordered sets (aka. sequences, lists, vectors, arrays) using angle brackets, $\langle\cdot\rangle$, and we use zero-based indexing, so that, if A is an ordered set, then $A[0]$ and $A[|A|-1]$ denote the first and last elements in A, respectively. If we have two ordered sets, $A = \langle a_1, a_2, \ldots a_k\rangle$ and $B = \langle b_1, b_2, \ldots b_\ell\rangle$, then the *concatenation of B onto A* is $A \oplus B = \langle a_1, a_2, \ldots a_k, b_1, b_2, \ldots b_\ell\rangle$. If $\Phi = \langle A_1, A_2, \ldots A_n\rangle$, where each A_i is an ordered set, then we define that $\bigoplus_{i=1}^{n} A_i = \bigoplus_{A \in \Phi} A = A_1 \oplus A_2 \oplus \ldots \oplus A_n$.

4.1 Computing the maximal transformable patterns in a point set

Figure 3 shows an algorithm that computes the non-empty maximal transformable patterns of size at least $s_{\min}$, in a dataset, $D \subset \mathbb{R}^k$, with respect to a transformation class, F. When F is a class of translations, the algorithm reduces to being SIA [15]. The algorithm generalizes the strategy adopted in SIA to the discovery of MTPs that are related by *any user-specified class of bijections over* $\mathbb{R}^k$.

Suppose we have two ordered sets of points, $\mathbf{P}$ and $\mathbf{Q}$, such that $|\mathbf{P}| = |\mathbf{Q}| = \beta$ and there exists at least one transformation, f, in a transformation class, F, such that $\mathbf{Q}[i] = f(\mathbf{P}[i])$ for all $0 \leq i < |\mathbf{P}|$. For a given F, we can choose β to be

```
MaximalTransformablePatterns(D, F, s_min)
1   V ← ⟨⟩
2   β ← GetBasisSize(F)
3   B ← ComputeObjectBases(D, β)
4   R ← ComputePermutationIndexSequences(β)
5   for i ← 0 to |B| − 1
6       b_obj ← B[i]
7       for j ← i to |B| − 1
8           b_img ← B[j]
9           for each r ∈ R
10              b'_img ← ⨁_{ℓ=0}^{β−1} ⟨b_img[r[ℓ]]⟩
11              T ← GetTransformations(F, b_obj, b'_img)
12              for each f ∈ T
13                  V ← V ⊕ ⟨⟨f, b_obj⟩, ⟨f^{−1}, b_img⟩⟩
14  V ← Sort_Lex(V)
15  M ← ⟨⟩
16  if |V| > 0
17      P ← nil, f ← nil
18      for i ← 0 to |V| − 1
19          if P = nil ∧ f = nil
20              ⟨f, P⟩ ← V[i]
21          else if V[i][0] = f
22              P ← P ∪ V[i][1]
23          else
24              if |P| ≥ s_min
25                  M ← M ⊕ ⟨⟨f, P⟩⟩
26              ⟨f, P⟩ ← V[i]
27      if |P| ≥ s_min
28          M ← M ⊕ ⟨⟨f, P⟩⟩
29  return M
```

Fig. 3: Algorithm for computing the maximal transformable patterns of size at least $s_{\min}$, in a dataset, D, with respect to a transformation class, F.

just large enough so that there is a manageably small maximum number of distinct transformations, $f \in F$, for which this is true. For example, for the transformation class, $F_{2\mathrm{T}}$, we can set β to 1, since there is only one translation vector that maps any given single point onto any other single point. On the other hand, given two distinct points, p and q, there are exactly *two* transformations, g, in $F_{2\mathrm{TR}}$, such that $q = g(p)$, namely those whose parameter vectors are $\langle q[0]-p[0], q[1]-p[1], 1\rangle$ and $\langle q[0]-p[0], -p[1]-q[1], -1\rangle$. For the transformation class $F_{2\mathrm{STR}}$, if we set β to 2, then there are at most 2 transformations that map $\mathbf{P}$ to $\mathbf{Q}$. We use the term *object basis* and *image basis* to refer to such point sequences, $\mathbf{P}$ and $\mathbf{Q}$, that are just large enough to determine a manageably small number of functions within some transformation class that map $\mathbf{P}$ to $\mathbf{Q}$.

The algorithm in Fig. 3 first initializes the variable, $\mathbf{V}$, which will hold a list of pairs, $\langle f, \mathbf{b}\rangle$, in which f is a transformation that belongs to the class, F, and $\mathbf{b}$ is a basis that is transformable within D by f. In line 2 of the algorithm, we determine the basis size, β, for F. Then, in line 3, the complete set of $\frac{|D|!}{\beta!(|D|-\beta)!} = O(\frac{|D|^\beta}{\beta!})$ β–combinations in D is computed and each combination is represented as an *ordered* set in which the points are in lexicographical order. Each of these ordered sets of points is an object basis and the complete set of object bases is stored in $\mathbf{B}$. In line 4, the set of $\beta!$ permutations of the sequence $\langle 0, \ldots \beta-1\rangle$ is computed

and stored in lexicographical order in $\mathbf{R}$. The basis size, β, will typically be small (in the cases considered in this paper, $\beta \leq 2$), so $\beta!$ is also typically small. In lines 5–11, we take each object basis, $\mathbf{b}_{\text{obj}} \in \mathbf{B}$ and then, for each permutation, $\mathbf{b}'_{\text{img}}$, of each basis, $\mathbf{b}_{\text{img}}$, that does not occur before $\mathbf{b}_{\text{obj}}$ in $\mathbf{B}$, we compute (in line 11) the transformations in F that map $\mathbf{b}_{\text{obj}}$ onto $\mathbf{b}'_{\text{img}}$. In lines 12–13, for each of these computed transformations, $f \in \mathbf{T}$, we append both $\langle f, \mathbf{b}_{\text{obj}} \rangle$ and $\langle f^{-1}, \mathbf{b}_{\text{img}} \rangle$ to $\mathbf{V}$. In line 14, $\mathbf{V}$ is sorted lexicographically, so that $\langle \text{transformation}, \text{basis} \rangle$ pairs in $\mathbf{V}$ with the same transformation form contiguous segments. The MTP for each transformation, $f \in F$, that occurs in D is then equal to the union of the bases in the contiguous pairs in $\mathbf{V}$ whose first elements are equal to f. The non-empty MTPs can therefore be found by scanning $\mathbf{V}$ just once using the process described in lines 17–28. For a k–dimensional dataset of size n, the overall running time is $O\left(\frac{k\tau}{(\beta-1)!} n^{2\beta} \log n\right)$ where τ is the maximum number of transformations returned by the GetTransformations function called in line 11. The algorithm uses $O\left(\frac{\tau(K(F)+k\beta)}{\beta!} n^{2\beta}\right)$ space.

4.2 Compressing point sets using maximal transformable patterns

Figure 4 outlines an algorithm that computes a compressed encoding of a point set in the form of a set of MTP occurrence sets. It takes three arguments: the input dataset, D, a transformation class, F, and a minimum size, $s_{\min}$, for the MTPs used in the encoding. The algorithm uses a simple greedy strategy, similar to that used in Forth's algorithm [6] and SIATECCompress [12], to find a set of MTP occurrence sets with high compression factors, whose covered sets collectively cover the input dataset. The first step is to use the algorithm in Fig. 3 to compute the non-empty MTPs in D for the transformations in F (line 1 in Fig. 4). The MTPs are stored in a list, $\mathbf{M}$, in which each element is an ordered pair, $\langle f, P \rangle$, where P is the MTP in D for the transformation, f. Next, in line 2, the MTPs are indexed by their size in a data structure, **SM**, which is an array of $|D| + 1$ lists. At the conclusion of line 2, each $\mathbf{SM}[i]$ is a list of the MTPs whose size is i, sorted lexicographically by their patterns, so that the MTPs with a given pattern, P, form a contiguous segment. The variable $\mathbf{s}$ contains the set of distinct sizes of MTPs discovered, in increasing order. The set of MTPs of a given size can therefore be accessed in constant time and we can use $\mathbf{s}$ to iterate over just the non-empty elements in **SM** in $O(|\mathbf{s}|)$ time.

A single pattern can be the MTP for two or more transformations in a dataset. All the $\langle f, P \rangle$ pairs with a given pattern form a contiguous segment within one of the lists in **SM**, so we can efficiently convert each such cluster into a pair, $\langle P, T \rangle$, where T is the set of transformations for which P is an MTP in D. This is done in line 3 using the MergeMTPs function, which constructs an array, **OS**, of $|D| + 1$ lists, that indexes the MTP occurrence sets in a similar way to that in which **SM** indexes the MTPs themselves. For each non-empty list of MTPs in **SM**, MergeMTPs merges each cluster of same-pattern MTPs into a pair, $\mathbf{os} = \langle P, T \rangle$, that is added to the appropriate list in **OS**.

```
EncodePointSet(D, F, s_min)
1    M ← MaximalTransformablePatterns(D, F, s_min)
2    ⟨SM, s⟩ ← IndexMTPs(M, |D|)
3    OS ← MergeMTPs(SM, s, |D|)
4    OS ← ComputeOccurrenceSets(OS, SM, s)
5    SOS ← SortOccurrenceSets(OS, γ, s)
6    return ComputeEncoding(SOS)
```

Fig. 4: The EncodePointSet algorithm.

If a pattern, Q, is transformable within D by a transformation, f, then any subset of Q will also be transformable within D by f. Moreover, for every transformation, $f \in F$ that occurs in D, there exists exactly one $\langle P, T\rangle$ pair in **OS** whose pattern, P, is the MTP of f. The occurrence set within D with respect to F of an MTP, P, is therefore the pair,

$$\langle P, \{f \mid (\exists \langle Q, T\rangle \in \mathbf{OS} \mid (f \in T \wedge P \subseteq Q))\}\rangle\,,$$

where, to promote readability, we define $\langle Q, T\rangle \in \mathbf{OS}$ if and only if there exists an i such that $\langle Q, T\rangle \in \mathbf{OS}[i]$. This implies that we can find the occurrence set of a pattern, P, simply by finding the union of the transformation sets of the pairs in **OS** whose patterns contain P. This is the strategy adopted in the ComputeOccurrenceSets function shown in Fig. 5 and called in line 4 of EncodePointSet.

```
ComputeOccurrenceSets(OS, SM, s)
1    for i ← 1 to |s| − 2
2        m ← s[i]
3        for each os ∈ OS[m]
4            for j ← i + 1 to |s| − 1
5                for each os_2 ∈ OS[s[j]]
6                    if os_2[0] ⊃ os[0]
7                        os[1] ← os[1] ∪ os_2[1]
8    for each m ∈ s
9        DeDupeAndSort(OS[m])
10       for each os ∈ OS[m]
11           os ← RemoveRedundantTransformations(os)
12       RemoveOccurrenceSetsWithNoTransformations(OS[m])
15   return OS
```

Fig. 5: The ComputeOccurrenceSets function.

For each occurrence set, $\langle P, T\rangle$, in **OS**, ComputeOccurrenceSets first searches through the occurrence sets with patterns larger than P for ones whose patterns contain P (lines 1–7 in Fig. 5). When it finds such an occurrence set, $\langle Q, T'\rangle$, it sets T to $T \cup T'$ (lines 6–7 in Fig. 5). This can result in occurrence sets that are equal to each other, even if their ordered-pair representations,

```
ComputeEncoding(SOS, D)
1    E ← ⟨SOS[0]⟩
2    S ← COV(SOS[0])
3    for i ← 1 to |SOS| − 1
4        os ← SOS[i]
5        ℓ ← Length(os[0]) + Length(os[1])
6        X ← COV(os) \ S
7        if ℓ < Dim(D) × |X|
8            E ← E ⊕ ⟨os⟩
9            S ← S ∪ X
10   R ← D \ S
11   if R ≠ ∅
12       E ← E ⊕ ⟨⟨R, ∅⟩⟩
13   return E
```

Fig. 6: The ComputeEncoding function.

$\langle P, T\rangle$, are different. In lines 8–9, the algorithm therefore iterates over the set of MTP sizes, **s**, and, for each size, m, it de-duplicates and sorts the list of occurrence sets, **OS**$[m]$. Redundant transformations are then removed from the transformation sets of the remaining occurrence sets in **OS** (line 10–11). Given an occurrence set, $\langle P, T\rangle$, if there are two transformations in T that map the object pattern, P, onto the same image pattern, then the more complex transformation is removed. For each occurrence set, $\langle P, T\rangle$, the algorithm also uses an adaptation of the RRT algorithm described in [14] to remove as many elements from T as possible without reducing the covered set of $\langle P, T\rangle$. Finally, in line 12, occurrence sets with empty transformation sets are removed, as they cannot contribute to compression.

When ComputeOccurrenceSets has finished executing in line 4 of EncodePointSet, **OS** contains MTP occurrence sets with compression factors greater than 1. The final step is to find a subset of these occurrence sets that collectively cover the dataset and whose combined description length is as low as we can make it using the simple, greedy compression strategy that we adopt here. In this strategy, we first sort the occurrence sets in **OS** to produce a sorted list, **SOS**, so that more preferred occurrence sets appear earlier in the list (line 5 of Fig. 4). In the implementation used to obtain the results reported below (Section 5), the occurrence sets were sorted into decreasing order by compression factor and then coverage. The ComputeEncoding function, shown in Fig. 6, is then called in line 6 of EncodePointSet to compute the final encoding of the dataset.

ComputeEncoding takes the sorted list of occurrence sets, **SOS**, and the dataset, D, and outputs an encoding in the form of a list, **E**, of occurrence sets, each represented by a $\langle P, T\rangle$ pair, in which P is an MTP and T is a set of transformations in F that map P onto other occurrences of P in D. The variable **E** is first initialized to hold the most preferred occurrence set, which is the first in **SOS** (line 1). In line 2, we then initialize a set, S, which holds the accumulated set of points collectively covered by the occurrence sets added to **E**. The algorithm iterates over the remaining occurrence sets in **SOS** (lines 3–9). The

description length, ℓ, of each **os** is computed (line 5) in terms of the total number of real-valued components in the points in its pattern and the parameter vectors in its transformation set. The set, X, is then computed, containing points in the covered set of **os** that are not yet present in S. X is the set of points that would be added to S if **os** were added to **E**. If ℓ is less than the product of the dimension of D and the number of points in X, then **os** is added to **E** as this results in X being encoded in a compressed form (line 7–8). The points in X are then added to S (line 9). Finally, after iterating over all the occurrence sets in **SOS**, there may remain some uncovered points, forming a so-called *residual point set* [13]. This can occur because **SOS** only contains occurrence sets whose compression factors are greater than one, as a result of non-compressing occurrence sets having been removed in line 12 of ComputeOccurrenceSets (see Fig. 5). In lines 10–12 of ComputeEncoding, this residual point set is computed, represented as an occurrence set with an empty transformation set, and then appended to the encoding.

5 Evaluation

EncodePointSet (Fig. 4) was evaluated on the task of classifying folk-song melodies into tune families, using the *Annotated Corpus* [8] of 360 melodies from the Dutch folk song collection, *Onder de groene linde* [7]. EncodePointSet was used as a compressor to calculate the normalized compression distance (NCD) [10] between each pair of melodies in the collection. Each melody was then classified using the 1-nearest-neighbour algorithm with leave-one-out cross-validation. The classifications obtained were compared with a ground-truth classification produced by expert musicologists. EncodePointSet was tested using the three transformation classes, $F_{2\mathrm{T}}$, $F_{2\mathrm{TR}}$ and $F_{2\mathrm{STR}}$, introduced above. We calculated the NCDs between the folk song melodies in the same way as that described in [13] and we evaluated performance in terms of average compression factor and classification success rate (see Table 1).

Table 1: Results of running EncodePointSet on the Annotated Corpus of the Dutch Folk Song Database with the three different transformation classes, $F_{2\mathrm{T}}$, $F_{2\mathrm{TR}}$ and $F_{2\mathrm{STR}}$. *SR* is classification success rate. Columns 3 and 4 give the mean compression factor achieved over, respectively, the corpus and the file-pairs used to compute the compression distances.

Transformation class	*SR*	*CF on corpus*	*CF on file pairs*
$F_{2\mathrm{T}}$	0.61	1.27	1.35
$F_{2\mathrm{TR}}$	0.61	1.27	1.26
$F_{2\mathrm{STR}}$	0.69	1.27	1.23

Increasing the complexity of the transformation class did not affect the average compression factor over the individual melodies and *decreased* the compres-

sion factor over the pair files. This may be because more complex contrapuntal transformations, such as inversion, retrograde, augmentation and diminution, do not occur frequently enough in these melodies to compensate for the increase in transformation class complexity, $K(F)$, when going from $F_{2\mathrm{T}}$ to $F_{2\mathrm{TR}}$ to $F_{2\mathrm{STR}}$. EncodePointSet did, however, achieve a higher classification success rate with $F_{2\mathrm{STR}}$ than with the simpler transformation classes.

6 Conclusions

We have presented an algorithm for discovering the maximal patterns in a dataset related by bijections within a user-specified transformation class. We have also described an algorithm that computes all the occurrences of these maximal patterns and uses these occurrence sets to construct a compressed encoding of the dataset. We have evaluated this compression algorithm with three different transformation classes on the task of classifying folk-song melodies into tune families. We found that, while the most complex transformation class gives the best classification success rate, it does not, on average, produce more compressed encodings on this corpus. This may be because the melodies in this corpus do not extensively employ more complex transformations like augmentation, diminution, inversion and retrograde. Despite this result, we believe the proposed approach is worthy of further investigation, since, in contrast to other current machine-learning approaches (e.g., neural network models), this approach computes *explanations* for data, in the sense that the encodings describe ways in which the data might have arisen as the result of transformations being applied to patterns within the data. The next step in our research will be to work on more thoroughly parallellising the proposed algorithms in order to be able to apply them to larger datasets in a variety of domains and on a broad range of machine-learning and information retrieval tasks.

References

1. Collins, T.: Improved methods for pattern discovery in music, with applications in automated stylistic composition. Ph.D. thesis, Faculty of Mathematics, Computing and Technology, The Open University, Milton Keynes (2011), http://oro.open.ac.uk/30103/4/TomCthesisEV.pdf
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: 14th ISMIR Conference (ISMIR 2013). pp. 549–554. Curitiba, Brazil (2013), https://archives.ismir.net/ismir2013/paper/000161.pdf
3. Collins, T., Arzt, A., Frostel, H., Widmer, G.: Using geometric symbolic fingerprinting to discover distinctive patterns in polyphonic music corpora. In: Meredith, D. (ed.) Computational Music Analysis, pp. 445–474. Springer, Cham, Switzerland (2016), http://link.springer.com/chapter/10.1007/978-3-319-25931-4_17
4. Collins, T., Thurlow, J., Laney, R., Willis, A., Garthwaite, P.H.: A comparative evaluation of algorithms for discovering translational patterns in baroque keyboard works. In: Proceedings of the 11th ISMIR Conference. pp. 3–8. Utrecht, The Netherlands (2010), https://archives.ismir.net/ismir2010/paper/000002.pdf

5. Collins, T., Volk, A., Janssen, B., Ren, I.: MIREX task on discovery of repeated themes & sections (2013–2017), https://www.music-ir.org/mirex/wiki/2017:Discovery_of_Repeated_Themes_%26_Sections
6. Forth, J.C.: Cognitively-Motivated Geometric Methods of Pattern Discovery and Models of Similarity in Music. Ph.D. thesis, Department of Computing, Goldsmiths, University of London (2012), http://research.gold.ac.uk/id/eprint/7803/
7. Grijp, L.P.: Introduction. In: Grijp, L.P., van Beersum, I. (eds.) Under the Green Linden—163 Dutch Ballads from the Oral Tradition, pp. 18–27. Meertens Institute/Music & Words (2008)
8. van Kranenburg, P., Volk, A., Wiering, F.: A comparison between global and local features for computational classification of folk song melodies. Journal of New Music Research **42**(1), 1–18 (2013)
9. Laaksonen, A., Lemström, K.: On the memory usage of the SIA algorithm family for symbolic music pattern discovery. In: Mathematics and Computation in Music (MCM 2022), Lecture Notes in Computer Science, Vol. 13267. pp. 180–191. Springer, Cham, Switzerland (2022), available online at https://link.springer.com/content/pdf/10.1007/978-3-031-07015-0_15.pdf
10. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M.B.: The similarity metric. IEEE Transactions on Information Theory **50**(12), 3250–3264 (2004), https://ieeexplore.ieee.org/document/1362909
11. Meredith, D.: The *ps13* pitch spelling algorithm. Journal of New Music Research **35**(2), 121–159 (2006), https://doi.org/10.1080/09298210600834961
12. Meredith, D.: COSIATEC and SIATECCompress: Pattern discovery by geometric compression. In: MIREX 2013 (Competition on Discovery of Repeated Themes & Sections) (2013), https://www.music-ir.org/mirex/abstracts/2013/DM10.pdf
13. Meredith, D.: Music analysis and point-set compression. Journal of New Music Research **44**(3), 245–270 (2015), http://dx.doi.org/10.1080/09298215.2015.1045003
14. Meredith, D.: RecurSIA-RRT: Recursive translatable point-set pattern discovery with removal of redundant translators. In: ECML PKDD 2019. Communications in Computer and Information Science, vol. 1168, pp. 485–493. Springer (2020), https://doi.org/10.1007/978-3-030-43887-6_42
15. Meredith, D., Lemström, K., Wiggins, G.A.: Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. Journal of New Music Research **31**(4), 321–345 (2002), https://doi.org/10.1076/jnmr.31.4.321.14162
16. Meredith, D., Lemström, K., Wiggins, G.A.: Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. In: Cambridge Music Processing Colloquium (2003), http://www.titanmusic.com/papers/public/cmpc2003.pdf