RECURSIA-RRT: Recursive translatable point-set pattern discovery with removal of redundant translators

David Meredith[0000−0002−9601−5017]

Aalborg University, Denmark
dave@create.aau.dk
http://www.titanmusic.com http://personprofil.aau.dk/119171

Abstract. We introduce two algorithms, RECURSIA and RRT, designed to increase the compression factor achievable using point-set cover algorithms based on the SIA and SIATEC pattern discovery algorithms. SIA computes the maximal translatable patterns (MTPs) in a point set, while SIATEC computes the translational equivalence class (TEC) of every MTP in a point set, where the TEC of an MTP is the set of translationally invariant occurrences of that MTP in the point set. In its output, SIATEC encodes each MTP TEC as a pair, $\langle P, V \rangle$, where P is the first occurrence of the MTP and V is the set of non-zero vectors that map P onto its other occurrences. RECURSIA recursively applies a TEC cover algorithm to the pattern P, in each TEC, $\langle P, V \rangle$, that it discovers. RRT attempts to remove translators from V in each TEC without reducing the total set of points covered by the TEC. When evaluated with COSIATEC, SIATECCompress and Forth's algorithm on the JKU Patterns Development Database, using RECURSIA with or without RRT increased compression factor and recall but reduced precision. Using RRT alone increased compression factor and reduced recall and precision, but had a smaller effect than RECURSIA.

Keywords: Pattern discovery · Point sets · Music analysis · Data compression · SIATEC · COSIATEC · SIATECCompress · Forth's algorithm · Geometric pattern discovery in music.

1 Introduction

The principle of parsimony posits that, when given two models that account equally accurately for a given set of observations (data), then the simpler model is less likely to be an accurate description of the data by chance. That is, the simpler model is more likely to be a faithful representation of the true process that gave rise to the data. This principle, commonly known as "Ockham's razor", has been formalized in various ways in recent times, including Rissanen's minimal description length principle [17] and Kolmogorov's structure function [18]. The principle has been one of the foundational principles of scientific enquiry since antiquity and recent results in information theory [19] have shown that data compression is almost always the best strategy both for model selection and prediction.

In recent years, we have had some success in using compression-based point-set pattern discovery algorithms, such as COSIATEC [13, 10, 14, 16], SIATECCOMPRESS

[13, 11, 14] and Forth's algorithm [4, 5], in conjunction with normalized compression distance, to carry out classification tasks such as folk song tune family detection [8, 13, 12]. Moreover, Louboutin and Meredith [8] found a highly significant correlation between compression factor and performance on the task of automatically discovering fugue subjects and countersubjects [6, 7]. This motivates us to search for ways to improve the compression factor achieved by such algorithms in the hope that improving compression factor may also result in improved performance on a variety of musicological tasks. Our research programme is driven by the hypothesis that shorter encodings of data objects represent better ways of understanding those objects. We therefore strive to devise algorithms that compute encodings of musical data objects that are as parsimonious as possible.

Let D be a set of k–dimensional points, such that $D \subset \mathbb{R}^k$ and $|D| = n$. We call D a *dataset*. For any vector, $v \in \mathbb{R}^k$, the *maximal translatable pattern* (MTP) in D is defined as $\text{MTP}(v, D) = D \cap (D - v)$. The SIA algorithm [15] computes all the non-empty MTPs in such a dataset in $\Theta(n^2 \log_2 n)$ time. Two point sets, P_1, P_2, are *translationally equivalent*, denoted by $P_1 \equiv_\text{T} P_2$, if and only if there exists a vector, v, such that $P_1 = P_2 + v$. The translational equivalence relation partitions the powerset of D exhaustively and exclusively into *translational equivalence classes* (TECs), such that the TEC to which a point set, $P \subseteq D$, belongs is defined to be $\text{TEC}(P) = \{Q \mid Q \subseteq D \wedge Q \equiv_\text{T} P\}$. The SIATEC algorithm [15] computes the TEC of every non-empty MTP in a dataset, D, in $\Theta(n^3)$ time. A TEC, $\text{TEC}(P)$, can be encoded in a compressed form as a pair, $\langle P, V\rangle$, where V is the set of non-zero vectors, $\{v \mid P + v \subseteq D\}$. Each TEC in the output of SIATEC is encoded in this form. Given a TEC, $T = \text{TEC}(P) = \langle P, V\rangle$, we define $P(T) = P$ and $V(T) = V$. $P(T)$ is called the TEC's *pattern* and $V(T)$ is called the TEC's *translator set* or *set of translators*. The *covered set* of a TEC, T, is the union of the point sets in the TEC and is given by $C(T) = P \cup \bigcup_{v \in V(T)} (P(T) + v)$. The *compression factor* of a TEC, $T = \text{TEC}(P) = \langle P, V\rangle$ is defined as $\text{CF}(T) = |C(T)| / (|P(T)| + |V(T)|)$. It is the ratio of $|C(T)|$, the number of points whose coordinates need to be explicitly specified if the covered set of the TEC is described *in extenso*, to $|P(T)| + |V(T)|$, the number of points and vectors whose coordinates need to be specified if the TEC is encoded as a pair, $\langle P, V\rangle$, as defined above.

SIATECCOMPRESS and Forth's algorithm use SIATEC to compute the MTP TECs in a dataset, D, and then attempt, using a greedy strategy, to select a subset of these TECs, E, such that $\bigcup_{T \in E} C(T) = D$ and $\sum_{T \in E} (|P(T)| + |V(T)|)$ is minimized. That is, these algorithms attempt to find a minimum-length description of the dataset in terms of a cover constructed from TEC covered sets. The TEC covered sets in the covers computed by SIATECCOMPRESS and Forth's algorithm may share points. However, the COSIATEC algorithm typically achieves better compression than these algorithms by partitioning the input dataset exhaustively and exclusively into non-intersecting TEC covered sets. It does this by incrementally constructing an encoding, E, by (1) running SIATEC, (2) adding the TEC with the best compression factor to E, (3) removing the covered set of this TEC from D and then repeating this three-step process on progressively smaller, unencoded subsets of the dataset until all the points in the dataset have been covered.

```
RECURSIA(𝒜, D)
1   E ← 𝒜(D)
2   if |E| = 1 ∧ |E[0][1]| = 1 return E
3   for i ← 0 to |E| − 1
4     e ← RECURSIA(𝒜, E[i][0])
5     if |e| > 1 ∨ |e[0][1]| > 1
6       E[i][0] ← e
7   return E
```

Fig. 1. The RECURSIA algorithm

In this paper, we introduce two novel techniques for improving the compression factor achieved using TEC cover algorithms. First, an algorithm, RECURSIA, is presented, that recursively applies a TEC cover algorithm to the pattern, P, in each TEC in the cover it generates. Second, an approximation algorithm, RRT, is presented, that aims to remove as many translators from each TEC as possible without removing points from its covered set. The two techniques are evaluated separately and in combination on the effect that they have on compression factor, recall and precision, when used with COSIATEC, SIATECCOMPRESS and Forth's algorithm on the JKU Patterns Development Database [2].

2 The RECURSIA algorithm

Figure 1 gives pseudocode for the RECURSIA algorithm. RECURSIA has two parameters, a TEC cover algorithm, $\mathcal{A}$ (e.g., COSIATEC, SIATECCOMPRESS or Forth's algorithm) and a dataset D. RECURSIA runs $\mathcal{A}$ on D to obtain an *encoding*, $\mathbf{E}$ (line 1 in Fig. 1), which is a list of TECs, $\mathbf{E} = \langle T_1, T_2, \ldots, T_{|\mathbf{E}|} \rangle$. Each TEC, T_i, is encoded as a pair, $\langle P_i, V_i \rangle$, as defined above. If the encoding, $\mathbf{E}$, contains only one TEC and the pattern for this TEC has only one occurrence, then $\mathcal{A}$ failed to find any non-trivial MTPs in D. In this case, $\mathcal{A}$ is not applied to the pattern in this TEC, so RECURSIA returns $\mathbf{E}$ (see line 2 in Fig. 1). If $\mathcal{A}$ finds more than one TEC or at least one TEC whose pattern has more than one occurrence, then RECURSIA is applied recursively to the pattern, $P_i = \mathbf{E}[i][0]$, in each TEC in $\mathbf{E}$ (Fig. 1, lines 3–4). This generates a new encoding, $\mathbf{e}_i$, for each pattern, P_i. If the encoding, $\mathbf{e}_i$, for a pattern, P_i, contains more than one TEC, or a TEC whose pattern occurs more than once, then $\mathbf{e}_i$ is a *compressed* encoding of P_i and $\mathbf{e}_i$ replaces P_i in the TEC, $\mathbf{E}[i]$ (Fig. 1, lines 5–6).

3 The RRT algorithm

Given a TEC, $T = \text{TEC}(P) = \langle P, V \rangle$, the RRT algorithm attempts to replace V with one of the smallest possible subsets of V—let us call it V'—such that $C(\langle P, V' \rangle) = C(T)$, where $C(T)$ denotes the covered set of T, as defined above. Exhaustively testing every subset of V to determine if the resulting covered set is the same as $C(T)$ would take time exponential in the size of V and would therefore only be practical for relatively small translator sets. RRT therefore uses a greedy approximation strategy with a polynomial time complexity instead of carrying out an exhaustive search.

```
RRT(T)
1  F ← ComputePointFreqSet(T)
2  if F[|F| − 1][0] = 1 return T
3  S ← ComputeSIAMVectorTable(T, F)
4  R ← ComputeRemovableVectors(T, S)
5  M ← ComputeMaxPoints(T, R, F)
6  if M = ∅ then T[1] \← R, return T
7  V ← ComputeVectorMaxPointSetPairs(M)
8  Q ← ComputeRetainedVectors(V)
9  return RemoveRedundantVectors(T, Q, R)
```

Fig. 2. The RRT algorithm

Figure 2 provides pseudocode for the RRT algorithm. For convenience, we define the function $V(p,T)$ to be the set of vectors in $V(T)$ that map points in $P(T)$ onto the point p. Formally,

$$V(p,T) = \{p - q \mid p - q \in V(T) \wedge q \in P(T)\} \,. \tag{1}$$

The first step in the algorithm is to compute for each $p \in C(T)$ the ordered pair $\langle f(p,T), p\rangle$, where $f(p,T) = |V(p,T)|$. These ordered pairs are placed in a sequence in lexicographical order and stored in the variable, $\mathbf{F}$ (Fig. 2, line 1). We call $f(p,T)$ the *frequency* of p in T. For example, for the TEC,

$$\langle\{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle\}, \{\langle 0,0\rangle, \langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle, \langle 4,4\rangle\}\rangle \tag{2}$$

the ComputePointFreqSet function would return

$$\langle\langle 1, \langle 1,1\rangle\rangle, \langle 1, \langle 7,7\rangle\rangle, \langle 2, \langle 2,2\rangle\rangle, \langle 2, \langle 6,6\rangle\rangle, \langle 3, \langle 3,3\rangle\rangle, \langle 3, \langle 4,4\rangle\rangle, \langle 3, \langle 5,5\rangle\rangle\rangle.$$

If, for some $p \in C(T)$, $f(p,T) > 1$, then we call p a *multipoint*. If $\mathbf{F}$ contains no multipoints, then none of the translators in $V(T)$ can be removed without also removing points from $C(T)$. This will be the case if and only if the frequency of the last entry in $\mathbf{F}$ is one. We therefore check for this in line 2 of Fig. 2 and return the TEC unchanged if it is the case.

The set of translators that can be removed from $V(T)$ is a subset of those vectors that map the whole pattern, $P(T)$, onto multipoints. That is, if a translator, $v \in V(T)$, maps any point in $P(T)$ onto a point in $C(T)$ that is not a multipoint, then we know that v cannot be removed from $V(T)$ without removing points from $C(T)$. We therefore define a *removable vector* to be a translator that maps the TEC's entire pattern, $P(T)$, onto a set of multipoints. In lines 3–4 of Fig. 2 we compute a list, $\mathbf{R}$, of these removable vectors. This is done by using the initial steps of the SIAM algorithm [10, 20] to compute the set, $S = \{\langle q - p, p\rangle \mid p \in P(T) \wedge q \in C(T) \wedge f(q,T) > 1\}$. This set S or *vector table* is sorted lexicographically to give the list, $\mathbf{S}$, (line 3 in Fig. 2) from which the maximal matches of the TEC pattern, $P(T)$, to the multipoints in $C(T)$ can be obtained. For example, for the TEC in Eq. 2, ComputeSIAMVectorTable returns the following sorted SIAM vector table, where each maximal match is printed

on its own line:

$$\begin{array}{l}\langle\langle\langle -1,-1\rangle,\langle 3,3\rangle\rangle,\\ \ \langle\langle 0,0\rangle,\langle 2,2\rangle\rangle,\langle\langle 0,0\rangle,\langle 3,3\rangle\rangle,\\ \ \langle\langle 1,1\rangle,\langle 1,1\rangle\rangle,\langle\langle 1,1\rangle,\langle 2,2\rangle\rangle,\langle\langle 1,1\rangle,\langle 3,3\rangle\rangle,\\ \ \langle\langle 2,2\rangle,\langle 1,1\rangle\rangle,\langle\langle 2,2\rangle,\langle 2,2\rangle\rangle,\langle\langle 2,2\rangle,\langle 3,3\rangle\rangle,\\ \ \langle\langle 3,3\rangle,\langle 1,1\rangle\rangle,\langle\langle 3,3\rangle,\langle 2,2\rangle\rangle,\langle\langle 3,3\rangle,\langle 3,3\rangle\rangle,\\ \ \langle\langle 4,4\rangle,\langle 1,1\rangle\rangle,\langle\langle 4,4\rangle,\langle 2,2\rangle\rangle,\\ \ \langle\langle 5,5\rangle,\langle 1,1\rangle\rangle\rangle\end{array} \tag{3}$$

The COMPUTEREMOVABLEVECTORS function (Fig. 2, line 4) scans this sorted SIAM vector table to identify the vectors that map the entire pattern onto multipoints (i.e., the ones for which the maximal matches have the same cardinality as the TEC pattern itself). For the TEC in Eq. 2, the list $\mathbf{R}$ returned by COMPUTEREMOVABLEVECTORS would be $\langle\langle 1,1\rangle,\langle 2,2\rangle,\langle 3,3\rangle\rangle$.

We say that $p \in C(T)$ is a *maxpoint* if and only if all the vectors in $V(p,T)$ (as defined in Eq. 1) are removable vectors, i.e., $V(p,T) \subseteq \mathbf{R}$. If $C(T)$ contains any maxpoints, then it will not be possible to remove all the vectors in $\mathbf{R}$ from $V(T)$ without also removing the maxpoints from the covered set. Indeed, we can remove all the vectors in $\mathbf{R}$ from $V(T)$ if and only if $C(T)$ contains no maxpoints. In line 5 of Fig. 2, the maxpoints are computed and then, in line 6, if there are no maxpoints, all the removable vectors, $\mathbf{R}$, are removed from the TEC's translator set and the modified TEC is returned. The COMPUTEMAXPOINTS function, called in line 5 of the RRT algorithm (line 5 in Fig. 2) actually returns a set of ordered pairs, $M = \{\langle p_1, R_1\rangle, \langle p_2, R_2\rangle, \ldots, \langle p_{|M|}, R_{|M|}\rangle\}$, where each $\langle p_i, R_i\rangle$ gives the maxpoint, p_i, and the set of removable vectors, R_i, that map pattern points onto that maxpoint. As an example, the TEC in Eq. 2 has just one maxpoint, so the COMPUTEMAXPOINTS function returns the following: $\{\langle\langle 4,4\rangle, \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle\}\rangle\}$.

If $C(T)$ contains maxpoints, then our goal is to find the smallest subset of $\mathbf{R}$ that contains, for each maxpoint, at least one vector that maps a point in $P(T)$ onto that maxpoint. We first compute a list of $\langle v, P\rangle$ pairs that give, for each removable vector, v, the set of maxpoints, P, onto which v maps points in the TEC pattern, $P(T)$. This is computed by the COMPUTEVECTORMAXPOINTSETPAIRS function in line 7 of the RRT algorithm in Fig. 2. Formally, COMPUTEVECTORMAXPOINTSETPAIRS computes the set, V, defined as follows: $V = \{\langle v, P\rangle \mid v \in \mathbf{R} \wedge P = \{p \mid p \in M \wedge p - v \in P(T)\}\}$. This set is then sorted to give an ordered set, $\mathbf{V}$, so that the $\langle v, P\rangle$ pairs are in decreasing order of maxpoint set size (i.e., pairs in which P is larger appear earlier in the list).

```
COMPUTERETAINEDVECTORS(V)
1   Q ← ∅
2   while V ≠ ⟨⟩
3     Q ← Q ∪ {V[0][0]}
4     for i ← 1 to |V| − 1 do V[i][1] ← V[i][1] \ V[0][1]
5     Y ← ⟨⟩
6     for i ← 1 to |V| − 1
7       if V[i][1] ≠ ∅ then Y ← Y ⊕ ⟨V[i]⟩
8     V ← Y
9   return Q
```

Fig. 3. The COMPUTERETAINEDVECTORS function. ($\mathbf{A} \oplus \mathbf{B}$ concatenates the lists $\mathbf{A}$ and $\mathbf{B}$.)

We then use $\mathbf{V}$ in a greedy strategy to find a small subset of $\mathbf{R}$ that contains, for each maxpoint, at least one vector that maps a point in $P(T)$ onto that maxpoint. This set of *retained vectors* is computed in line 8 of Fig. 2 by the COMPUTERETAINEDVECTORS function (shown in Fig. 3). The first step in this function is to add to the list of retained vectors, $\mathbf{Q}$, the vector associated with the largest set of maxpoints, that is, the first in the list $\mathbf{V}$ (see lines 1–3 of Fig. 3). All the maxpoints mapped to by that vector from points in the TEC pattern can then be removed from the maxpoint sets of the other elements in $\mathbf{V}$ (line 4 in Fig. 3). The effect of lines 5–8 of Fig. 3 is to remove from $\mathbf{V}$ the first element and every other element whose maxpoint set is empty after removing the maxpoint set of the first element. The process is repeated, with the vector of the first pair in the list being selected on each iteration until $\mathbf{V}$ is empty. This results in a list, $\mathbf{Q}$, of retained vectors that constitute a subset of the removable vectors that is sufficient to generate all the maxpoints. Finally, in line 9 of Fig. 2, the REMOVEREDUNDANTVECTORS function removes from the TEC's set of translators all removable vectors that are not retained vectors.

4 Evaluation

Figure 4(a) shows the effect of RECURSIA and RRT on the compression factor achieved using a variety of SIATEC-based TEC cover algorithms, when these algorithms were used to analyse the five pieces in the JKU Patterns Development Database [2]. Three basic algorithms, COSIATEC, SIATECCOMPRESS and Forth's algorithm were run, each with and without compactness trawling [3] (indicated by 'CT') and with or without the SIA algorithm replaced by SIAR [1] (indicated by 'R'). Each of these 12 algorithms was run in its basic form (orange curve), with RECURSIA (blue curve), with RRT (green curve), and with both RECURSIA and RRT (red curve). As expected, using RECURSIA and RRT together nearly always improved compression factor, with particularly large gains being observed on the Beethoven and Mozart sonata movements when Forth's algorithm was used with compactness trawling. Using RRT alone only had a noticeable effect on the Bach fugue and the Beethoven sonata movement. Over all pieces and algorithms, using RECURSIA in combination with RRT improved compression factor by 12.5%, using RECURSIA alone improved it by 9.2% and using RRT alone improved it by 2.1%. Figure 4(b) shows the effect that RECURSIA and RRT had on three-layer precision (TLP) [13], averaged over the pieces in the JKU-PDD and for the same 12 algorithms, each run in "Raw" mode, "BB" mode and "Segment" mode (see [13]). On average, over all pieces, algorithms and modes, using RECURSIA in combination with RRT reduced TLP by 20.3%, using RECURSIA alone reduced it by 21.2% and using RRT alone reduced it by 0.7% (see Fig. 4(b)). On the other hand, on average, over all pieces, algorithms and modes, using RECURSIA and RRT together increased three-layer recall (TLR) [13] by 7.2%, using RECURSIA alone increased it by 10.3%. Using RRT alone reduced TLR by 3.7% (see Fig. 4(c)).

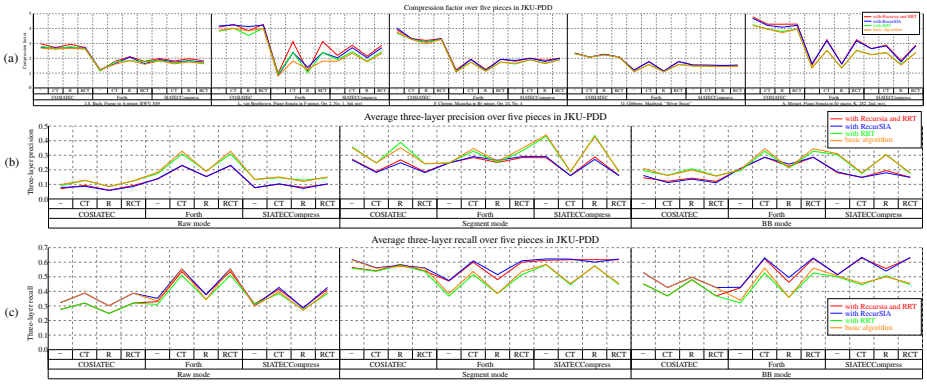

Fig. 4. Effect of RECURSIA and RRT on compression factor (a), three-layer precision (b) and recall (c), over the pieces in the JKU-PDD.

5 Conclusion

Two algorithms, RECURSIA and RRT, have been presented, designed to increase the compression factor achieved using any TEC cover algorithm. When tested with three basic algorithms and evaluated on the JKU Patterns Development database, using RECURSIA with or without RRT increased compression factor and three-layer recall but reduced three-layer precision. Using RRT alone generally had a smaller effect than using RECURSIA, and, on average, increased compression factor but reduced both recall and precision on the JKU-PDD.

Supplementary materials

The results reported in this paper were obtained using the implementations of the algorithms in the OMNISIA software [9]. The source code for the version of OMNISIA used here is available on GitHub at https://github.com/chromamorph/omnisia-recursia-rrt-mml-2019. An executable JAR file is also available at http://www.titanmusic.com/software/omnisia/201904151348OMNISIA.zip.

Acknowledgements

The author would like to thank Geraint A. Wiggins for suggesting the idea of applying the COSIATEC algorithm recursively to the patterns in TECs.

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)

2. Collins, T.: JKU Patterns Development Database (2013), available at https://dl.dropbox.com/u/11997856/JKU/JKUPDD-Aug2013.zip
3. 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: 11th International Society for Music Information Retrieval Conference (ISMIR 2010), Utrecht, The Netherlands, 9–13 August 2010. pp. 3–8 (2010)
4. 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)
5. Forth, J., Wiggins, G.A.: An approach for identifying salient repetition in multidimensional representations of polyphonic music. In: Chan, J., Daykin, J.W., Rahman, M.S. (eds.) London Algorithmics 2008: Theory and Practice, pp. 44–58. College Publications, London (2009)
6. Giraud, M., Groult, R., Levé, F.: Subject and counter-subject detection for analysis of the well-tempered clavier fugues. In: Aramaki, M., Barthet, M., Kronland-Martinet, R., Ystad, S. (eds.) From Sounds to Music and Emotions: 9th International Symposium, CMMR 2012 London, UK, June 19–22, 2012. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7900), pp. 422–438. Springer-Verlag, Berlin and Heidelberg (2013)
7. Giraud, M., Groult, R., Levé, F.: Truth file for the analysis of Bach and Shostakovich fugues (2013/12/27 version) (2013), available online at http://www.algomus.fr/truth/fugues.truth.2013.12
8. Louboutin, C., Meredith, D.: Using general-purpose compression algorithms for music analysis. Journal of New Music Research **45**(1), 1–16 (2016)
9. Meredith, D.: Omnisia, http://www.titanmusic.com/software/omnisia/OMNISIA.pptx
10. Meredith, D.: Point-set algorithms for pattern discovery and pattern matching in music. In: Dagstuhl Seminar on Content-based Retrieval (No. 06171, 23–28 April, 2006). Schloss Dagstuhl, Germany (2006), http://drops.dagstuhl.de/opus/volltexte/2006/652
11. Meredith, D.: COSIATEC and SIATECCompress: Pattern discovery by geometric compression. In: MIREX 2013 (Competition on Discovery of Repeated Themes & Sections) (2013), available online at http://www.titanmusic.com/papers/public/MeredithMIREX2013.pdf
12. Meredith, D.: Using point-set compression to classify folk songs. In: Fourth International Workshop on Folk Music Analysis (FMA 2014), 12–13 June 2014. Bogazici University, Istanbul, Turkey (2014)
13. Meredith, D.: Music analysis and point-set compression. Journal of New Music Research **44**(3), 245–270 (2015)
14. Meredith, D.: Analysing music with point-set compression algorithms. In: Meredith, D. (ed.) Computational Music Analysis, pp. 335–366. Springer (2016)
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)
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
17. Rissanen, J.: Modeling by shortest data description. Automatica **14**(5), 465–471 (1978)
18. Vereshchagin, N.K., Vitányi, P.M.B.: Kolmogorov's structure functions and model selection. IEEE Transactions on Information Theory **50**(12), 3265–3290 (2004)
19. Vitányi, P.M.B., Li, M.: Minimum description length induction, Bayesianism and Kolmogorov complexity. IEEE Transactions on Information Theory **46**(2), 446–464 (2000)
20. Wiggins, G.A., Lemström, K., Meredith, D.: SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In: Proceedings of the Third International Conference on Music Information Retrieval (ISMIR 2002), Paris, France, 13–17 October 2002. pp. 283–284 (2002)