A parallel, geometric algorithm for transformed pattern matching in polyphonic music

David Meredith[0000−0002−9601−5017]

Aalborg University, Rendsburggade 14, 9000 Aalborg, Denmark
dave@create.aau.dk
https://vbn.aau.dk/en/persons/119171
https://www.titanmusic.com

Abstract. We present a parallel, geometric, pattern-matching algorithm that can be used for finding transformed matches of a polyphonic musical pattern, P, in a polyphonic musical dataset, D, where neither P nor D are required to contain any information about the voices or parts to which notes belong. We assume that P and D are represented as sets of k-dimensional points, $P, D \subset \mathbb{R}^k$. Given a class, F, of bijections from $\mathbb{R}^k$ to $\mathbb{R}^k$, we define a pattern, Q, to be a *maximal transformed match* of P in D with respect to F, if there exists a transformation, $f \in F$, such that $Q = D \cap f(P)$, where $f(P) = \bigcup_{p \in P}\{f(p)\}$. Our proposed algorithm returns the maximal transformed matches of P in D with respect to F. If $|P| = m$ and $|D| = n$, then the algorithm does $O\left(\beta! + \frac{\tau}{(\beta-1)!}(mn)^\beta \log_2 m\right)$ work, uses $O(\tau(|\alpha_F|+k\beta)(mn)^\beta/(\beta!))$ space and has a span of $O(\beta! + \tau\beta^2 \log_2 m(\log_2 m + \log_2 n - \log_2(\beta!)))$, where τ, $|\alpha_F|$ and β depend on the transformation class and are typically small natural numbers. We evaluated the algorithm on two musicological tasks: (1) discovering occurrences of the "HAYDN" theme in Ravel's *Menuet sur le nom d'Haydn*, on which the algorithm achieved an F_1 score of 0.7; and (2) discovering subject entries in *Contrapunctus VI* from Bach's *Die Kunst der Fuge*, on which the algorithm achieved an F_1 score of 0.93.

Keywords: Music information retrieval · Pattern-matching algorithms · Computational musicology · Geometric algorithms · Parallel algorithms.

1 Introduction

We present a parallel, geometric, pattern-matching algorithm that can be used to find transformed matches of a user-specified, polyphonic, musical query pattern in a polyphonic, symbolic, music encoding, where neither the query nor the encoding being searched are required to contain information about how the music is structured into voices or parts. This problem can arise when one has a symbolic encoding of a score where information about voice structure is absent or unreliable. For example, in piano music, it is often unclear how the music should be parsed into voices, and picking one particular interpretation runs the

risk of failing to find matches of a query pattern that are split across different voices in the particular voicing structure chosen, but that are perhaps contained wholly within single voices in another, equally valid, interpretation.

We use the term *dataset* to refer to the encoding being searched. We are especially interested in discovering matches that are related to a query pattern by transformations within a *user-specified set of transformations*. In Section 4, we present an algorithm for solving this problem when the dataset, D, and query pattern, P, are represented by sets of k-dimensional points, $D, P \subset \mathbb{R}^k$ and when the user-specified set of transformations, or *transformation class*, F, is a set of bijections from $\mathbb{R}^k$ to $\mathbb{R}^k$. We define a pattern, Q, to be a *maximal transformed match* of P in D with respect to F, if there exists a transformation, $f \in F$, such that $Q = D \cap f(P)$, where $f(P) = \bigcup_{p \in P}\{f(p)\}$.

Our proposed algorithm returns all maximal transformed matches of P in D with respect to F. The algorithm does $O\left(\beta! + \frac{\tau}{(\beta-1)!}(mn)^\beta \log_2 m\right)$ work, using $O(\tau(|\alpha_F| + k\beta)(mn)^\beta/(\beta!))$ space and has a span of $O(\beta! + \tau\beta^2 \log_2 m(\log_2 m + \log_2 n - \log_2(\beta!)))$, where τ, $|\alpha_F|$ and β depend on the transformation class and are typically small integers. Our implementation of the algorithm allows the user to set minimum thresholds on the size and compactness of the matches returned.[1]

The algorithm we propose is well-suited to finding matches related to a query pattern by any combination of inversion, retrograde, augmentation or diminution, since these transformations can be modelled by simple geometric bijections over appropriate point-set representations. We focus on cases where queries and datasets are 2-dimensional point sets in which each point represents a note and where one dimension represents time and the other represents pitch. In this case, the class of traditional contrapuntal transformations corresponds to the class of transformations over $\mathbb{R}^2$ that consist of combinations of a translation, a reflection in the time axis (inversion) and a scaling parallel to the time axis by either a positive or negative scale factor (augmentation, diminution, retrograde). Note that the algorithm we propose allows us to find matches scaled by *any real scale factor* and does not require the user to provide a finite, fixed set of scale factors over which to search.

In the remainder of this paper, we first review some related previous work (Section 2). In Section 3, we define some theory relating to maximal transformed matches required for understanding the new algorithm, which is presented in Section 4. In Section 5, we present the results of using the algorithm to carry out two musicological tasks.

2 Related work

The algorithm we propose in Section 4 can be considered a generalization of the SIAM algorithm and its variants, SIAME and SIAMESE [10]. SIAM discovers

[1] The *compactness* of a pattern, P, in a dataset, D, is the ratio of the number of points in P to the number of points in D that are contained within the bounding box of P [5, 249–250].

all maximal translationally-invariant matches of a query pattern consisting of a set of m, k-dimensional points in a dataset containing n, k-dimensional points in $O(knm \log n)$ time and $O(kmn)$ space (assuming $n \geq m$). SIAM was developed as a variant of the SIA geometric pattern discovery algorithm [7]. Clifford et al. [1] prove that the maximal subset matching problem solved by SIAM is 3SUM-hard and that therefore any subquadratic solution would necessarily require approximation. They present a randomised approximation algorithm called MSM that uses FFT-based binary wildcard matching to solve the maximal subset matching problem in $O(n \log n)$ time. Building on the approach of SIAM, Ukkonen et al. [9] present sweepline algorithms that find all complete translationally invariant matches of a pattern of size m in a dataset of size n (their "P1" problem) in $O(mn)$ time and $O(m)$ working space. However, the problem considered here is closer to their "P2" problem, which is that of discovering all maximal translationally invariant matches. They present an $O(mn \log m)$-time, $O(m)$-space algorithm for solving P2. Lemstrom et al. [4] introduced fast, index-based filtering algorithms for the P1 and P2 problems that have output-sensitive running times and use only a linear-sized index.

Lemström [3] presents algorithms for finding the transposition- and time-scale-invariant occurrences of a query pattern in a dataset. His methods are capable of finding all complete occurrences in $O(mwn \log n)$ time and all partial such occurrences in $O(mnw^2 \log n)$ time where w is the size of a window within which each occurrence must occur. Laaksonen [2] presents an $O(n^2)$ algorithm for finding time-scaled occurrences of a query pattern of size m in a dataset of size n, assuming that all coordinates are integers and that the difference in time between any two consecutive notes is less than some constant, c. The algorithm (unlike the one presented below) only finds time-scaled occurrences where the scale factor is within some finite set of possible values.

Thus, although there exist a number of efficient methods for finding query patterns that are transformed by certain classes of transformation, to our knowledge there is no previous work that proposes a general pattern-matching algorithm that allows the user to define the class of transformations by which occurrences may be related to a query pattern.

3 Maximal transformed matches

We adapt the pattern *discovery* framework proposed in [6] to pattern *matching*. Given a query pattern, $P \subset \mathbb{R}^k$, and a dataset, $D \subset \mathbb{R}^k$, we define a *transformation* to be a bijection, $f : \mathbb{R}^k \to \mathbb{R}^k$ and a *transformation class*, F, to be a set of such transformations. Note that a transformation, as we define it here, is a bijective mapping from a *point* in $\mathbb{R}^k$ to a *point* in $\mathbb{R}^k$; it is *not* a mapping from P to D. However, for notational convenience, we define that, if f is a transformation and $S \subset \mathbb{R}^k$, then $f(S) = \bigcup_{p \in S} \{f(p)\}$.

Each transformation, $f \in F$, has a distinct *parameter*, $\alpha_F(f)$, that uniquely identifies f within F. For example, if $F_{2\mathrm{T}}$ is the class of 2-dimensional translations, then $\alpha_{F_{2\mathrm{T}}}(f)$ would simply be the vector by which f translates a

2-dimensional point. 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 $\phi_F(\alpha_F(f), p) = f(p)$, for all $p \in \mathbb{R}^k$ and $f \in F$. We call ϕ_F the *transformation class function* for the class F. For example, $\phi_{F_{2\mathrm{T}}}(\alpha_{F_{2\mathrm{T}}}(f), p) = p + \alpha_{F_{2\mathrm{T}}}(f)$. 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. Given a 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, $\alpha_F(f)$. Given a pattern, $P \subset \mathbb{R}^k$, a dataset, $D \subset \mathbb{R}^k$, and a transformation, $f : \mathbb{R}^k \to \mathbb{R}^k$, we define the *maximal transformed match* (MTM) of P in D for the transformation, f, to be the set of points, $D \cap f(P)$.

As another example, consider the class, $F_{2\mathrm{STR}}$, of 2-dimensional transformations consisting of a scaling parallel to the x-axis with non-zero scale factor, followed by a translation, optionally followed by a reflection in the x-axis. Let $D \subset \mathbb{R}^2$, such that each point, $\langle t, p\rangle \in D$, represents a note, with t representing the mid-duration timepoint of each note, 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. More specifically, transposition corresponds to a translation by a vector with a non-zero y-coordinate, inversion corresponds to a reflection in the x-axis, retrograde corresponds to an x-axis-aligned scaling with a scale factor of -1, and augmentation and diminution correspond to x-axis scalings with scale factors whose absolute values are greater than 1 and less than 1, respectively. We represent the temporal location of each note by its mid-duration timepoint rather than its onset, because, under a retrograde, the onset of a note maps onto the offset of the image note, whereas the mid-duration timepoint of a note maps onto the mid-duration timepoint of the image note. 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 the scale factor of the x-axis-aligned stretch (positive or negative so as to include retrogrades), v is a translation vector and b indicates whether the point is reflected in the x-axis ($b = -1$ if it is and $b = 1$ if it is not). If we set $\alpha = \alpha_{F_{2\mathrm{STR}}}$, then the transformation class function for $F_{2\mathrm{STR}}$ is then

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

4 A parallel algorithm for finding the maximal transformed matches of a query pattern in a dataset

Figure 1 shows an algorithm, MaxMatch, that takes a pattern, $P \subset \mathbb{R}^k$, a dataset, $D \subset \mathbb{R}^k$, and a transformation class, F, and computes the maximal transformed matches (MTMs) of P in D with respect to F of size at least $s_{\min}$.

The final parameter of the algorithm is the size, ℓ, of the hash table used to store the maximal transformed matches when they are computed. Our pseudocode follows the conventions used in a number of related studies (e.g., [5]).

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 just large enough so that there is a finite maximum number of distinct transformations, $f \in F$, for which this is true. For example, for the transformation class $F_{2\text{STR}}$, if we set β to 2, then there are at most 2 transformations that map $\mathbf{P}$ to $\mathbf{Q}$. We call β the *basis size* and we use the term *object basis* and *image basis*, respectively, to refer to such point sequences, $\mathbf{P}$ and $\mathbf{Q}$, that are just large enough to ensure that there is a finite number of transformations within a transformation class that map $\mathbf{P}$ to $\mathbf{Q}$.

```
MaxMatch(P, D, F, s_min, ℓ)
1   M ← ⟨⟩[ℓ]
2   β ← GetBasisSize(F)
3   N_obj ← NumCombinations(|P|, β)
4   N_img ← NumCombinations(|D|, β)
5   R ← ComputePermutationIndexSequences(β)
6   C ← N_obj N_img |R|
7   spawn ComputeMTMs(P, D, F, M, 0, C, N_obj, N_img, R, β)
8   sync
9   M ← ⟨⟩
10  for i ← 0 to ℓ − 1
11      if |M[i]| > 0
12          for j ← 0 to |M[i]| − 1
13              if |M[i][j][0]| ≥ s_min
14                  M ← M ⊕ ⟨M[i][j]⟩
15  return M
```

Fig. 1: A parallel, geometric algorithm for computing the maximal transformed matches of size at least $s_{\min}$, of a pattern, P, in a dataset, D, with respect to a transformation class F.

The first step in MaxMatch (line 1 in Fig. 1) is to construct a hash table, $\mathcal{M}$, of size ℓ. Each element in this hash table is a list of $\langle f, S \rangle$ pairs in each of which f is a transformation and S is a point set that eventually becomes the MTM of P in D for f. In lines 2–8, the MTMs for the transformation class, F, are computed in parallel using the kernel function, ComputeMTMs (shown in Fig. 2). In order to carry out this computation in parallel, ComputeMTMs needs to be provided with the basis size, β, for F (line 3 of MaxMatch) and the total number of object bases, N_{obj}, and image bases, N_{img}, which are equal to the number of combinations of size β in the pattern, P, and dataset, D, respectively (lines 3 and 4 of MaxMatch). In line 5 of MaxMatch, the set of $\beta!$ permutations of the sequence $\langle 0, \ldots \beta - 1 \rangle$ is computed and stored in lexicographical order in $\mathbf{R}$. β will typically be very small (e.g., in the case of F_{2STR}, $\beta = 2$), so $\beta!$ is also typically small. In line 6 we calculate the total number of computations to be computed in parallel and store this in C. Then in line 7, the kernel function,

ComputeMTMs, is called on a new thread with its start and end indices, i_{start} and i_{end}, set to 0 and C.

```
ComputeMTMs(P, D, F, M, i_start, i_end, N_obj, N_img, R, β)
1    if i_end − i_start = 1
2        N_perms ← |R|
3        j ← i_start/(N_obj N_perms)
4        i ← (i_start mod (N_obj N_perms))/N_perms
5        r ← R[i_start mod N_perms]
6        b_obj ← ComputeBasis(P, β, i)
7        b_img ← ComputeBasis(D, β, j)
8        b'_img ← ⊕_{y=0}^{β−1} ⟨b_img[r[y]]⟩
9        T ← GetTransformations(F, b_obj, b'_img)
10       for each f ∈ T
11           h ← Hash(f, ℓ)
12           synchronized(M[h])
13               Insert(M[h], f, b_obj)
14   else
15       s ← ⌊(i_end + i_start)/2⌋
16       spawn ComputeMTMs(P, D, F, M, i_start, s, N_obj, N_img, R, β)
17       spawn ComputeMTMs(P, D, F, M, s, i_end, N_obj, N_img, R, β)
```

Fig. 2: ComputeMTMs kernel function, called in line 7 of MaxMatch.

The ComputeMTMs function, shown in Fig. 2, is a recursive kernel function that is always run on its own thread. The base case occurs when $i_{\text{end}} - i_{\text{start}} = 1$ in line 1. When the base case occurs, the algorithm selects a specific object basis from P ($\mathbf{b}_{\text{obj}}$ in line 6) and a specific permutation ($\mathbf{b}'_{\text{img}}$ in line 8) of a specific image basis ($\mathbf{b}_{\text{img}}$ in line 7) and then computes the transformations in F that map $\mathbf{b}_{\text{obj}}$ onto $\mathbf{b}'_{\text{img}}$ and stores these in the list $\mathbf{T}$ (line 9). The algorithm then iterates over the transformations f in $\mathbf{T}$, computing a hash value, h for each f (line 11) and then inserting a pair, $\langle f, \mathbf{b}_{\text{obj}} \rangle$, into the slot with index h in the hash table, $\mathcal{M}$ (line 13). If a pair $\langle f, S \rangle$ already exists in $\mathcal{M}[h]$ whose first element is equal to f, then the points in $\mathbf{b}_{\text{obj}}$ are added to S; otherwise, a new pair, $\langle f, \mathbf{b}_{\text{obj}} \rangle$, is added to $\mathcal{M}[h]$. When ComputeMTMs is called with a range between i_{start} and i_{end} that is greater than 1, the function splits this range in half and spawns two new calls to ComputeMTMs over each of these two new ranges (lines 14–17). When all parallel calls to ComputeMTMs have terminated in line 8 in MaxMatch (see Fig. 1), the MTMs are harvested from the hash table, $\mathcal{M}$, and stored in a list, $\mathbf{M}$ which is returned (lines 9–15 in Fig. 1).

4.1 Analysis of the algorithm

To analyse the work, span and space used by the algorithm, we first note that lines 1, 2 and 6 of MaxMatch take $O(1)$ time, that lines 3 and 4 take $O(\beta)$ time and that line 5 takes $O(\beta!)$ time. The overall worst-case time complexity of lines 1–6 of MaxMatch is therefore $O(\beta!)$. If $|P| = m$ and $|D| = n$, then the value of C in line 6 of MaxMatch is equal to

$$C = \beta! \binom{m}{\beta} \binom{n}{\beta} = \beta! \frac{m!}{\beta!(m-\beta)!} \frac{n!}{\beta!(n-\beta)!} = O\left(\frac{(mn)^\beta}{\beta!}\right).$$

The recursion tree created by the calls to ComputeMTMs is $O(\log_2 C)$ deep and contains $O(C)$ non-base-case calls (which simply spawn new calls) and $O(C)$ base-case calls that execute lines 2–13 in ComputeMTMs.

In ComputeMTMs, lines 2–7 run in $O(1)$ time and line 8 runs in $O(\beta)$ time. Line 9 runs in $O(\tau)$ time, where τ is the maximum number of transformations returned by the GetTransformations function (typically a small natural number). The 'for' loop in lines 10–13 iterates $O(\tau)$ times. Line 11 runs in $O(1)$ time, whereas line 13 can involve computing the union of $\mathbf{b}_{\text{obj}}$ and a point set already stored in $\mathcal{M}[h]$. This will take $O(\beta \log_2 m)$ worst-case time, implying that lines 10–13 take a total of $O(\tau\beta \log_2 m)$ time in the worst case.

This implies that the span of that part of MaxMatch that computes the maximal transformed matches (lines 1–8 of MaxMatch) is

$$O(\beta! + \tau\beta \log_2 m \log_2 C) = O(\beta! + \tau\beta^2 \log_2 m(\log_2 m + \log_2 n - \log_2(\beta!)))$$

and the work done by this part of the algorithm is

$$O(\beta! + C\tau\beta \log_2 m) = O\left(\beta! + \frac{\tau}{(\beta-1)!}(mn)^\beta \log_2 m\right).$$

The space used by the algorithm is dominated by the hash table, $\mathcal{M}$. A maximum of τ $\langle f, \mathbf{b}_{\text{obj}}\rangle$ pairs are inserted into $\mathcal{M}$ on each of the $O(C)$ base-case calls to ComputeMTMs. Each of these pairs has length $O(|\alpha_F| + k\beta)$, where $P, D \subset \mathbb{R}^k$, giving an upper bound of $O(\tau C(|\alpha_F| + k\beta)) = O(\tau(|\alpha_F| + k\beta)(mn)^\beta/(\beta!))$ for the total space used by the algorithm. The running time of lines 9–14 of MaxMatch, which simply gather the MTMs into a suitable form for output, is equal to $O(C)$.

5 Evaluation

We evaluated MaxMatch on two musicological tasks with F set to $F_{2\text{STR}}$. In the first task, the goal was to find occurrences of the "HAYDN" theme (B-A-G-G-D) in Ravel's *Menuet sur le nom d'Haydn*. In the second task, the goal was to find all entries of the subject in *Contrapunctus VI* from Bach's *Die Kunst der Fuge* (BWV 1080). On the task of discovering occurrences of the "HAYDN" theme in Ravel's *Menuet sur le nom d'Haydn*, the output of MaxMatch was compared against two different ground truths, the first consisting of patterns identified by Soucy [8], and the second consisting of patterns identified by the author. In both cases, the best results were obtained when the minimum pattern size was set to 4 (1 less than the size of the query), the minimum compactness was set to 0.5, and the onset of each note was replaced with its mid-duration timepoint to allow for retrogrades to be found. When the output generated with these settings was compared with the author's ground truth, MaxMatch achieved a precision of 0.91, recall of 0.56 and an F_1 score of 0.70. When compared with Soucy's ground truth, the precision, recall and F_1 score were 0.89, 0.57 and 0.70, respectively. On the task of discovering subject entries in *Contrapunctus VI* from

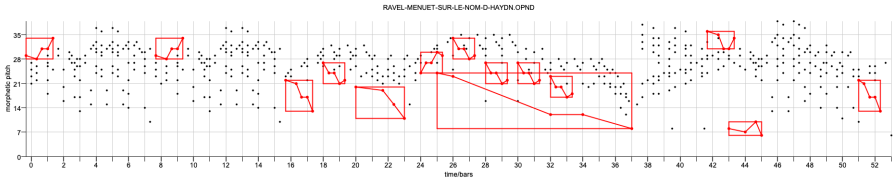
Bach's *Die Kunst der Fuge*, the best results were obtained when the minimum pattern size was set to 11 (3 less than the size of the query), no constraints were set on pattern compactness and onset times were used unchanged. With these settings, MaxMatch achieved a precision of 0.95, recall of 0.92 and an F_1 score of 0.93. Figure 3 shows, for each task, the best output generated by the algorithm compared with the ground-truth analyses.[2]

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

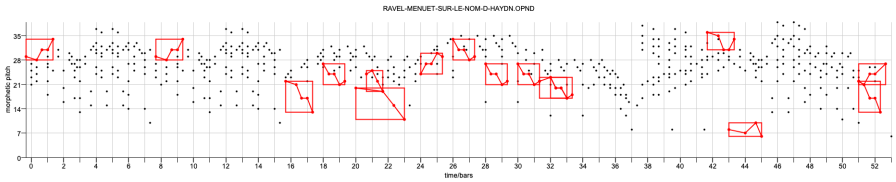
References

1. Clifford, R., Christodoulakis, M., Crawford, T., Meredith, D., Wiggins, G.A.: A fast, randomised, maximum subset matching algorithm for document-level music retrieval. In: ISMIR 2006 (2006), https://ismir2006.ismir.net/PAPERS/ISMIR06120_Paper.pdf
2. Laaksonen, A.: Two-dimensional point set pattern matching with horizontal scaling. In: Proceedings of the Sixth BCS-IRSG Symposium on Future Directions in Information Access (FDIA 2015) (2015), http://dx.doi.org/10.14236/ewic/FDIA2015.9
3. Lemström, K.: Towards more robust geometric content-based music retrieval. In: ISMIR 2010. pp. 577–582 (2010), https://archives.ismir.net/ismir2010/paper/000099.pdf
4. Lemström, K., Mikkilä, N., Mäkinen, V.: Fast index based filters for music retrieval. In: ISMIR 2008. pp. 677–682 (2008), https://archives.ismir.net/ismir2008/paper/000132.pdf
5. 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
6. Meredith, D.: Understanding and compressing music with maximal transformable patterns. LNCS **14035**, 309–325 (2023), https://link.springer.com/chapter/10.1007/978-3-031-34732-0_24
7. 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
8. Soucy, J.P.: Six French composers' homage to Haydn: an analytical comparison enlightening their conception of tombeau. Master's thesis, McGill University, Montréal (2009), masters thesis, https://escholarship.mcgill.ca/downloads/rb68xh19n
9. Ukkonen, E., Lemström, K., Mäkinen, V.: Sweepline the music! LNCS **2598**, 330–342 (2003), https://doi.org/10.1007/3-540-36477-3_25
10. Wiggins, G.A., Lemström, K., Meredith, D.: SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In: ISMIR 2002. pp. 283–284 (2002), https://ismir2002.ismir.net/proceedings/03-SP03-4.pdf

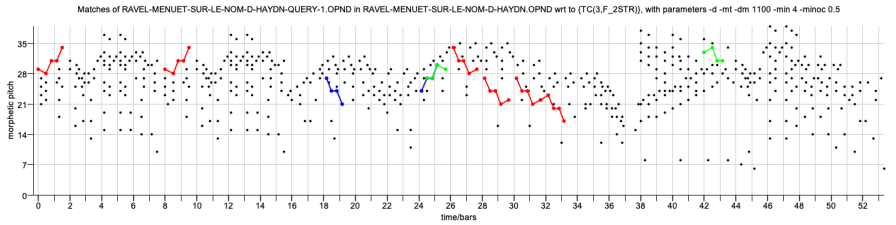
[2] These results were produced using a Java implementation of the algorithm, which is available on GitHub at https://github.com/chromamorph/omnisia/releases/tag/2024-09-17-OMNISIA.



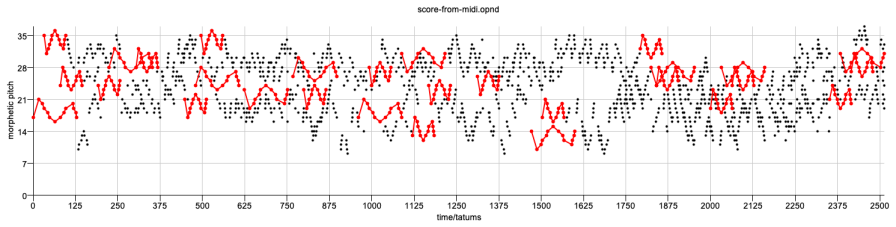
(a) Soucy's [8] ground truth for Ravel's *Menuet*.



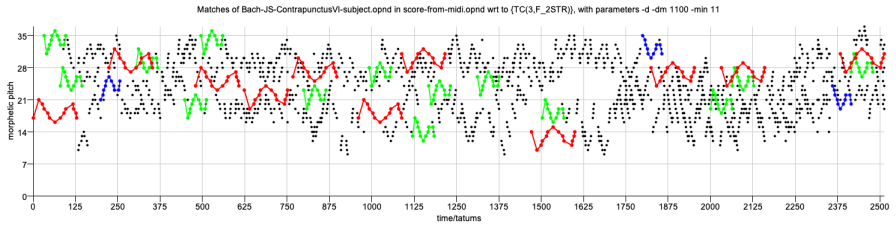
(b) Author's ground truth for Ravel's *Menuet*.



(c) Matches retrieved by MaxMatch for Ravel's *Menuet*.



(d) Ground-truth subject entries in Bach's *Contrapunctus VI*.



(e) Matches retrieved by MaxMatch for Bach's *Contrapunctus VI*.

Fig. 3: Ground-truth analyses for Ravel's *Menuet* and Bach's *Contrapunctus VI* along with results generated by MaxMatch.