PATTERN MATCHING IN POLYPHONIC MUSIC AS A WEIGHTED GEOMETRIC TRANSLATION PROBLEM

Anna Lubiw
University of Waterloo
School of Computer Science

Luke Tanur
University of Waterloo
School of Computer Science

ABSTRACT

We consider the music pattern matching problem—to find occurrences of a small fragment of music called the "pattern" in a larger body of music called the "score"—as a problem of translating a set of horizontal line segments in the plane to find the best match in a larger set of horizontal line segments. Our contribution is that we use fairly general weight functions to measure the quality of a match, thus enabling approximate pattern matching. We give an algorithm with running time $O(nm \log m)$, where n is the size of the score and m is the size of the pattern. We show that the problem, in this geometric formulation, is unlikely to have a significantly faster algorithm because it is at least as hard as a basic problem called 3-SUM that is conjectured to have no subquadratic algorithm. We present some examples to show the potential of this method for finding minor variations of a theme, and for finding polyphonic musical patterns in a polyphonic score.

1. INTRODUCTION

Music information retrieval is a rapidly evolving, multidisciplinary research area [7, 4]. One of the problems at its core is the "music pattern matching problem"—to find occurrences of a small fragment of music (the "pattern") in a larger body of music (the "score").

The techniques required for this problem differ depending on whether the music is represented symbolically or as audio. This paper focuses on the former; for literature on audio representations and the pattern matching problem in that context, see [11].

With music represented symbolically, there are still a variety of approaches to the music pattern matching problem. Efforts are underway to compare these approaches on large data sets, see Downie [8]. Techniques based on string matching have been most heavily explored [15]. These include edit distance [19] and n-gram [9] techniques. Since these algorithms work on sequences, polyphonic music poses a great challenge, though there have been attempts to handle it in this framework [16, 6].

For polyphonic music, the pattern matching problem is more tractable when music is represented in a richer, more geometric format than as a 1-dimensional string—when it is represented as line segments in the plane [23], weighted point sets in the plane [22], or multi-dimensional point sets [24].

This paper explores the possibilities of a particular geometric approach to music pattern matching. We model each note as a line segment in the plane—see Figure 1. The vertical axis corresponds to pitch and the horizontal axis corresponds to time; in particular, the length of a line segment indicates the duration of the note. This representation is a natural one, and has been used by many others, for example in the Music Animation Machine [18] and by Brinkman and Mesiti [3].

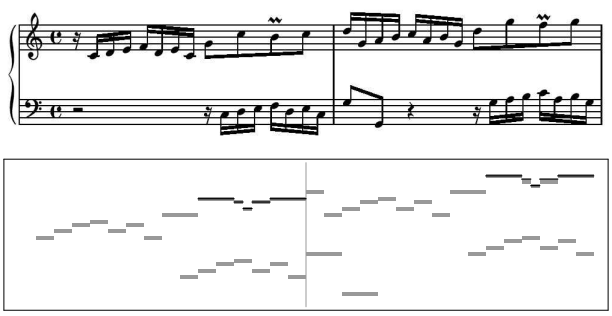

Figure 1. J.S. Bach, Invention 1, BWV 772, bars 1–2, and the representation as line segments. Black line segments indicate an exact match of a pattern (on the left) and an inexact match of the same pattern (on the right).

Matching the pattern into the score means translating the pattern relative to the score, where "translation" is used in its mathematical sense. Imagine the pattern drawn on a transparent sheet that can be shifted horizontally and vertically over the score to find the best position. The vertical shift corresponds to transposing the pattern. The horizontal shift corresponds to locating the pattern in time. Some matches are better than others. An *exact match* is a translation of the pattern so that each line segment of the pattern exactly matches a line segment of the score. See Figure 1. Exact matches have limited applicability—they encompass transposition, but allow no other variation. For a richer set of possibilities, we introduce weight functions and we search for matches of optimum weight.

Algorithms using this approach have been developed

for some specific weight functions. Ukkonen et al. [23] define the weight of a match to be the sum of the lengths of the overlaps of pattern and score line segments. They give an algorithm to find maximum weight matches of a monophonic pattern in a polyphonic score.

A series of papers by Ó Maidín [17], Francu and Neville-Manning [12], and Aloupis et al. [1] give algorithms to minimize a weight function that measures the area between a monophonic pattern and a monophonic score.

We introduce a weighting scheme that encompasses both of these measures, and many more. We can, for example, assign weights depending on the interval between a note of the pattern and a note of the score; for example, matching notes an octave apart could contribute more weight than matching notes an augmented 4^{th} apart. Mongeau and Sankoff [19] used such a weighting scheme in their edit-distance algorithm. We include examples to show a little of the power and flexibility of our approach.

Our purpose in this paper is to examine the *efficiency* and the *efficacy* of this approach to music pattern matching. With respect to *efficiency*, we give an algorithm in Section 2 to solve the music pattern matching problem in time $O(nm \log m)$ where m is the size of the pattern and n is the size of the score. This is the same running time as is achieved by Ukkonen et al. [23] in their algorithm to maximize length of pattern-score overlap, and by Aloupis et al. [1] in their algorithm to minimize the area between pattern and score. The running time of our algorithm is also competitive with other approaches to the music pattern matching problem, such as edit distance techniques. It is, however, disappointing in the sense that string pattern matching can be done much more efficiently, in linear time $O(n+m)$. The quadratic time behavior for music pattern matching is acceptable for small input sizes, but is prohibitively slow for huge ones, such as those envisioned in google-style music query systems.

However, we argue in Section 3 that for this geometric approach, quadratic behaviour is the best that can be achieved without a significant breakthrough in some basic algorithm design problems. In particular, our model of the music pattern matching problem includes as a special case a problem about containment of points in line segments. This latter problem is known to be equivalent, in terms of computational complexity, to other problems for which no one has a subquadratic algorithm, and for which it is conjectured that no such algorithm exists [2]. This is not a proved lower bound, but it is evidence towards a lower bound, which, given the dismal state of lower bound techniques, is something. We know of no previous lower bound arguments in music pattern matching.

With respect to *efficacy*, i.e. whether this approach has anything to offer music theorists and musicologists, we give some examples in Section 4 to show what is possible with our methods. In particular, we consider some of the examples that Selfridge-Field [20] identifies as being problematic for automatic classification systems.

For more detail on this work, see the Master's thesis of the second author [21].

2. ALGORITHM

2.1. Overview

For the music pattern matching problem, we are given a pattern of m notes and a score of n notes, represented as line segments. We are also given a weight function with which to evaluate a translation of the pattern in the score. We wish to find the translation of the pattern in the score that has maximum weight. More generally, we want not only "the best" match, but a number of good matches.

Our algorithm is an efficient version of the most basic approach to this music pattern matching problem: to try all possible translations of the pattern in the score, and compute the weight of each, in order to find the translations that have maximum weight. The algorithm of Ukkonen et al. [23] uses this same approach, and our algorithm can be viewed as an extension of theirs to more general weight functions.

There are two main ingredients to an efficient implementation. One is to identify a bounded-size set of candidate translations that includes all possible optimum solutions to the music pattern matching problem. We show a bound of $O(nm)$ on the number of candidate translations. The other ingredient is to avoid computing the weight of each translation from scratch, but rather to update efficiently from one translation to the next. This is possible for many, though not all, weight functions. We discuss the allowable weight functions in Section 2.3, and show how to preprocess the score in time $O(n)$ to achieve an update time of $O(\log m)$ to find the next translation and $O(1)$ to compute its weight.

Putting these together, we obtain an $O(nm \log m)$ algorithm for the music pattern matching problem.

In the analysis of our algorithm, we make crucial use of the assumption that musical pitches come from a discrete set. Our examples use the 128 MIDI values based on semi-tones, but our algorithm would apply to any discrete set, for example scale degrees, or the base-40 representation of Hewlett [14]. Our running time of $O(nm \log m)$ hides the dependence on 128. To put it more precisely, our running time is actually $O(nm(d + \log m))$ where d is the size of the discrete pitch set. We remark that, although 128 is a constant, it is a rather large constant, and an algorithm whose running time does not depend on d would certainly be desirable. This is possible for specialized weight functions and/or monophonic music [23, 1]. It remains an open problem to achieve this independence from d for polyphonic music and our general weight functions.

2.2. Notation and Input Data

A note s is represented by its starting time, $\sigma(s)$, its ending time, $\tau(s)$, and its pitch, $\pi(s)$. The note s corresponds to the horizontal line segment from the point $(\sigma(s), \pi(s))$ to the point $(\tau(s), \pi(s))$.

We assume that the notes of the score are given sorted by $\sigma(s)$. This is true for data coming from a MIDI file,

but other data may need to be sorted at an extra cost of $O(n \log n)$.

For the purpose of our algorithm, we need an ordered list of all the distinct $\sigma(s)$ and $\tau(s)$ values. These are called the *time points* of the score. There are at most $2n$ time points, and for polyphonic music, generally fewer. We can compute the list of time points from the sorted score by scanning through the score once, keeping a heap of the τ-values of the notes that are playing at the current time. This takes time time $O(n \log l)$, where l is the maximum polyphony of the score—i.e. the maximum number of notes being played at any one time. We do not assume that l is bounded by 128, but we do assume it to be bounded by a constant.

2.3. The Weight Model

We use a weight function to measure deviations of the translated pattern from the score. Note that translating the pattern is "free"; only the differences between the translated pattern and the score count. Our weight functions are additive—i.e. the weight of a particular translation of the pattern is the sum of the weights of its notes.

It seems natural that matching a long note should count more than matching a short note. We effect this by setting the weight of a translated note p matching a score note s that occupies the same time interval will be $(\tau(p) - \sigma(p))f(\pi(s), \pi(p))$, where f is a function of the pitches of s and p. More generally, if s and p overlap in time, we use the length of the overlap instead of $(\tau(p) - \sigma(p))$.

When a translated pattern note overlaps in time with several notes of a monophonic score, we allow pieces of the pattern note to match with different notes of the score. This captures what Mongeau and Sankoff [19] call "fragmentation", where one note is replaced by several. The opposite transformation, "consolidation", is captured when several pattern notes match to the same note of the score. A portion of a translated pattern note may match a portion of a note of the score only if they occupy the same time span. See Figure 2(a).

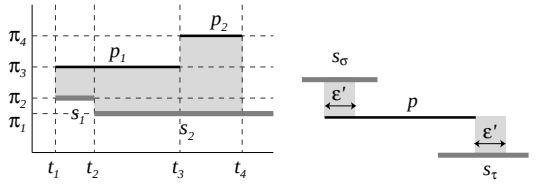

Figure 2. Computing the weight function: (a) weight is $(t_2-t_1)f(\pi_2, \pi_3)+(t_3-t_2)f(\pi_1, \pi_3)+(t_4-t_3)f(\pi_1, \pi_4)$; (b) the effect of a shift by ϵ'.

Polyphonic music may have several pattern notes and several score notes occupying the same time span. In this case we match each (piece of a) pattern note to the single note of the score that gives the best weight.

A very simple version of such a weight function sets $f(\pi(s), \pi(p))$ to be 1 if $\pi(s) = \pi(p)$, and 0 otherwise. In geometric terms, the weight of a translation of the pattern is then the sum of the lengths of the overlap of pattern and score line segments. This is the weight function used by Ukkonen et al. [23].

A more complicated version of such a weight function sets $f(\pi(s), \pi(p))$ to be the difference between $\pi(s)$ and $\pi(p)$. Using MIDI pitches, this is the number of semitones in the interval between the two notes. In geometric terms, this weight function measures the area between the translated pattern and the score, and we want a match of *minimum* weight. This weight function was used for the case of monophonic music by Ó Maidín [17], Francu and Neville-Manning [12], and Aloupis et al. [1].

More generally, we can define $f(\pi(s), \pi(p))$ to depend on the interval between the two notes in a more complicated way. For example, we can assign a better value to an interval of 7 semi-tones (a perfect 5^{th}) than to the smaller interval of 6 semi-tones. Mongeau and Sankoff [19] use a scheme like this in their edit-distance algorithm, assigning weights to intervals in increasing order of dissonance. The particular weighting of intervals that we use in our examples is shown in Table 1. Note that an exact match gets a weight of 1, and we seek a *maximum* weight match. We make no claim about these weights being ideal; further experimentation would be good—see Section 4.

Our method can extend to functions $f(s, p)$ that depend on other properties of the notes s and p than pitch—for example stress, dynamics, relative position in the bar, etc.

2.4. The Set of Candidate Translations

We can think of the score as lying in a grid formed by the 128 MIDI pitches along the vertical axis, and the time points of the score along the horizontal axis. This grid has size at most $2n \times 128$.

Claim 1 *With any weight function as described above, there will be an optimum match of the pattern into the score that has some line segment of the pattern starting or ending at one of these grid points.*

Thus, for any weight function as described above, there are at most $128 \cdot 4 \cdot nm$ candidate translations.

Proof:

Consider an optimum weight translation of the pattern into the score. The translation must leave pitches on the grid, but suppose that no line segment of the translated pattern starts or ends at a time point. Let ϵ be the minimum shift left or right that causes the start or end of a pattern note to reach a time point. Consider a note p of the translated pattern. Shifting p by any ϵ', $-\epsilon \leq \epsilon' \leq \epsilon$, does not alter which notes of the score p matches to; it only alters, by ϵ', the length of the portion of p matching at either end. See Figure 2(b). Thus there is a value $\delta(p)$ such that the change in weight due to the shift of p is

$\epsilon'\delta(p)$. More precisely, if the initial portion of the translated note p matches to score note s_σ and the final portion of p matches to score note s_τ, and p has been translated to pitch π, then $\delta(p) = f(\pi(s_\tau), \pi) - f(\pi(s_\sigma), \pi)$. The change over all notes is $\epsilon' \sum_p \delta(p)$. If $\sum_p \delta(p)$ is positive, then a positive ϵ' would increase the weight; if $\sum_p \delta(p)$ is negative, then a negative ϵ' would increase the weight. Since we assumed the translation to be of optimum weight, $\sum_p \delta(p)$ must be 0, and therefore a shift of ϵ does not change the weight, and lines up the start or end of a pattern note with a time point. Thus there is an optimum match on the grid. □

We remark that the number of candidate translations can be reduced to $4nm$ in some cases because there will be an optimum match in which some line segment of the pattern starts [or ends] exactly where a line segment of the score starts [or ends]. This happens if the weight function only measures exact overlap of pattern and score line segments. With the distance weight function, this property fails, and the number of candidate translations goes up by a factor of d. For the pure distance function, Aloupis et al. [1] are able to avoid looking at all the candidate translations, but we see no way to do this for our more general weight functions and polyphonic music.

2.5. Preprocessing

In order to quickly determine the weight of a translated note, we precompute a weight matrix W based on the score and the given weight function. Matrix W has a row for each of the 128 MIDI pitches, and a column for each of the time points of the score, of which there are at most $2n$. Thus it has size at most $128 \times 2n$, which is $O(n)$.

For pitch π and time point t, the corresponding matrix entry, $W(\pi, t)$, contains the weight factor to be applied to a note of the pattern translated to pitch π, and going from time point t to the next time point t'. Thus such a translated pattern note contributes $(t' - t)W(\pi, t)$ to the weight of a match. In terms of the function f described above, $W(\pi, t) = \max\{f(\pi(s), \pi) : s$ is a score note that includes the time interval $(t, t')\}$.

We compute W by iterating through the notes s of the score, and updating $W(\pi, t)$ as π ranges through the 128 pitch values, and t ranges through the time points from $\sigma(s)$ up to, but not including, $\tau(s)$.

Each of the $128 \times 2n$ matrix positions will be updated at most l times, where l is the maximum polyphony of the score. As mentioned above, we assume l to be a constant, so the time to set up W is at most $O(n)$. More precisely, with a pitch set of size d, and maximum polyphony of l, the time to compute W is $O(dln)$.

We can reduce this to $O(dn \log l)$ by computing W across rows, and using a heap to compute the best weight for each matrix entry. We note that if the space to store the matrix is considered prohibitive, we can dispense with the matrix altogether, and simply compute weights as we need them during the main algorithm.

2.6. The Main Matching Algorithm

We try each possible candidate translation (t, π), where t is the translation applied to the time coordinate, and π is the translation applied to the pitch coordinate.

We try each value of t, in order. We call it an *event* when the start or the end of a translated pattern note lines up with a time point of the score. Lining up the start of pattern note p with time point u occurs at translation value $t = u - \sigma(p)$. Lining up the end of the pattern note with time point u occurs at translation value $t = u - \tau(p)$. We go through the events in order of their translation values. Note that several events may occur at the same translation value, but for book-keeping purposes we handle them one at a time. The number of events is at most $4nm$.

To go through the events in order, we keep, for each pattern note p, two pointers p_σ and p_τ into the list of time points. The pointer p_σ gives the time point for the next event involving the start of note p, and p_τ gives the time point for the next event involving the end of note p. We make a heap of the translation values corresponding to these $2m$ forthcoming events. This allows us to find the next event in $O(\log m)$ time. Each of the $2m$ pointers makes one pass through the list of time points.

As we go through the events—i.e. the values of t—we maintain information for each value of π—i.e. each transposition of the pattern. There are at most $2 \cdot 128$ values of π. The information we maintain includes the weight for the candidate translation (t, π), but also other information that allows us to update each of these weights in $O(1)$ time. Specifically, we store, for each value of π, and each pattern note p, a value $\delta_\pi(p)$ with the property that a small change $t \leftarrow t + \epsilon$ changes p's contribution to the weight of the match at transposition value π by the amount $\epsilon\delta_\pi(p)$. Suppose that π translates note p to pitch $\nu = \pi(p) + \pi$. The start of p has just passed time point $prev(p_\sigma)$ and the end of p has just passed time point $prev(p_\tau)$. Then $\delta_\pi(p) = W(\nu, prev(p_\tau)) - W(\nu, prev(p_\sigma))$.

If w_π is the weight of the pattern translated by (t, π) then the change to w_π caused by $t \leftarrow t + \epsilon$ is $\epsilon \sum_p \delta_\pi(p)$. We maintain $\Delta_\pi = \sum_p \delta_\pi(p)$. At an event involving pattern note p, we must update $\delta_\pi(p)$, Δ_π, and w_π for each value of π. If this event is for the start of p, then $\delta_\pi(p)$ changes by $\delta_\pi = W(\nu, prev(p_\sigma)) - W(\nu, p_\sigma)$. If the event is for the end of p then $\delta_\pi(p)$ changes by $\delta_\pi = W(\nu, p_\tau) - W(\nu, prev(p_\tau))$.

Suppose that the current event occurs at translation value t', and that the previous event occurred at translation value t (possibly $t = t'$). The changes are as follows: $w_\pi \leftarrow w_\pi + (t' - t)\Delta_\pi$; $\delta_\pi(p) \leftarrow \delta_\pi(p) + \delta_\pi$; $\Delta_\pi \leftarrow \Delta_\pi + \delta_\pi$. Each event involves a single pattern note p, but we make changes for each of the $O(d)$ settings of π, for a total time of $O(d)$.

Altogether, we have $O(nm)$ events, time $O(\log m)$ to find the next event, and time $O(d)$ to update at each event—for a total running time of $O(nm(d + \log m))$, and space of $O(dn)$. Finding the k best matches adds $O(k)$ to the time and space. (We use linear time median finding.)

3. BARRIERS TO A FASTER ALGORITHM

Our algorithm for the music pattern matching problem takes time $O(nm \log m)$ for a score of size n and a pattern of size m. A subquadratic algorithm with running time $O(n+m)$ (achievable for string pattern matching) or even $O((n+m)\log n)$, would be vastly preferable, and would make the algorithm practical for use in large music databases. In this section we show that such an efficient algorithm will be a major challenge.

More specifically, we show that the music pattern matching problem, in this geometric formulation, includes as a very special case a problem called "Segments Containing Points" which is at least as hard as the 3-SUM problem—and *that* problem is conjectured to have no algorithm with a subquadratic running time. We expand on these points in the remainder of this section. For background on $O(\cdot)$ and $o(\cdot)$ notation, see [5]. The two problems are as follows:

3-SUM: Given a set of n integers, does it contain numbers a, b, c with $a + b + c = 0$?

Segments Containing Points (SCP): Given a set P of m numbers and a set Q of n pairwise-disjoint intervals, is there a translation u such that $P + u \subseteq Q$?

The geometric formulation of the music pattern matching problem includes SCP as the very special case where the pattern and the score have notes only on one pitch, the pattern notes are very short, and the 0-1 weight function is used.

Barequet and Har-Peled [2] prove that an algorithm with running time $o(nm)$ for the SCP problem would imply an algorithm with running time $o(n^2)$ for the 3-SUM problem. Thus SCP is "3-SUM hard". The class of 3-SUM hard problems was introduced by Gajentaan and Overmars [13], who show that a number of different problems are 3-SUM hard. Although this is not a proved lower bound (see [10]) a subquadratic algorithm for any of these problems would be a major breakthrough.

Thus a subquadratic algorithm for the geometric version of the music pattern matching problem would have implications far beyond music information retrieval. From a practical point of view, at least for small patterns, the factor of 128 in our algorithm is probably more prohibitive than the factor of m. Certainly, given the above, it is a more tractable challenge to attempt to reduce the dependence on the size of the pitch set.

4. EXAMPLES AND DISCUSSION

For all our examples we use the weights shown in Table 1, which are somewhat similar to those used by Mongeau and Sankoff [19]. We assign a very good weight to a semitone, because we model pitches on a scale of 12 semitones, rather than as scale degrees. Thus a pattern repeated at a different scale degree will in general be off by semitones. We also assign a good weight to an octave, and to semi-tones on either side of an octave.

Our first example is the Bach Two-Part Invention, Number 1, BWV 772, the first two bars of which are shown in Figure 1. Our first pattern consists of the first 8 notes of the piece, but with the final G replaced by a 16^{th} note D, as it appears in inverted form. Figure 3 shows the first and last four bars of the results of a search for the best 20 matches. There are 18 "correct" matches in the piece. Our algorithm assigns them weights between 90% and 100%, and nicely separates them from the extraneous matches—except that it finds a good match (92.5%) of the pattern into the final chord. Since our pattern has so few distinct pitches, it matches quite well against the two notes of a minor 3^{rd} in a chord. It might be possible to avoid such anomalies by assigning extra weight when the start of a pattern note matches the start of a score note.

When we search for the inversion of our first pattern, the match into the final chord gets a lower weight than the 19 valid matches, so our algorithm correctly separates the good matches from the extraneous ones. See Figure 4.

Our second example is the Andante movement of Mozart's Piano Sonata K311, with the theme of the first 2 bars as the pattern. See Figure 5. This is one of the examples

Interval apart (in semi-tones)	Weight
perfect unison (0)	1
minor 2^{nd} (1)	0.9
major 2^{nd} (2)	0.6
minor 3^{rd} (3)	0.4
major 3^{rd} (4)	0.4
perfect 4^{th} (5)	0.2
perfect 5^{th} (7)	0.6
minor 6^{th} (8)	0.3
major 6^{th} (9)	0.3
major 7^{th} (11)	0.7
perfect octave (12)	0.8
minor 9^{th} (13)	0.7
all other intervals	0

Table 1. The weighting scheme used for our experiments

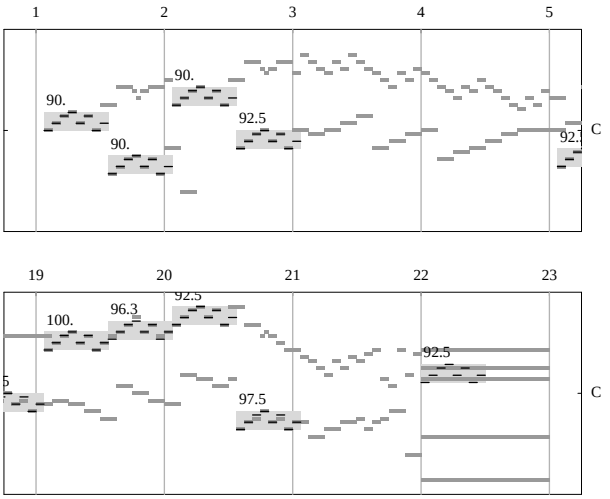

Figure 3. Selected bars showing matches of the first pattern in Bach's Invention.

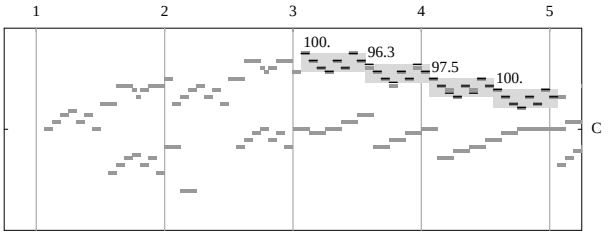

Figure 4. Matches of the inversion of the first pattern in bars 1–4 of Bach's Invention.

discussed by Selfridge-Field [20]. She points out that the recurrences of the theme differ slightly from each other, making automatic classification difficult. Some 6-bar segments of the result of our search are shown in Figure 7. Bars are numbered as though the repeat were explicitly written out. The first match does not receive 100% because the pattern omits the grace note, and because the MIDI file, produced from a human performance, has the note durations shortened. The top 9 matches found by our algorithm are good ones, and include all the variants listed by Selfridge-Field. The final variant she lists is the syncopated one shown in Figure 6 (top), which occurs in bar 86. Our algorithm detects this as the 9^{th} match, at weight 84.4%. See the last pane of Figure 7. The 10^{th} match, starting just before bar 54, is extraneous, but the 11^{th}, at bar 50, is good. See the third pane of Figure 7.

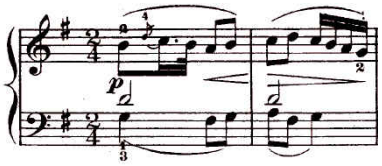

Figure 5. W.A. Mozart, Piano Sonata in D major K311, 2^{nd} movement, bars 1–2, with the theme in the right hand.

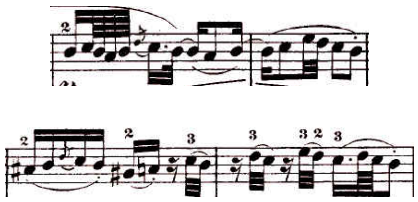

Figure 6. Two occurrences of the theme in the right hand: bars 86–87 (top) and bars 90–91 (bottom).

Unlike in the Bach example, there are occurrences of the theme that are ranked poorly by our algorithm and do not make it into even the top 20 matches. We have identified 5 such occurrences. Four of them are just artifacts of the particular MIDI file we used: in bars 38–39 and 74–75, the theme occurs in the left hand, in thirds, but the performer played these notes staccato, so the lengths of the notes in the MIDI file are roughly half of what the score indicates (this can be seen in the 2^{nd} pane of Figure 7), and the weight computed by our algorithm is commensurately reduced. (Actual values are about 60%.) When we change the lengths of these notes to accurately reflect the written score, these matches receive weights of above 90%, and would rank among the "good" matches. The fifth occurrence of the theme that our algorithm misses is more interesting. See Figure 6 (bottom) and bars 90–91 of Figure 7. To human eyes and ears, this is a variant of the theme, but it poses quite a challenge for automatic pattern detection!

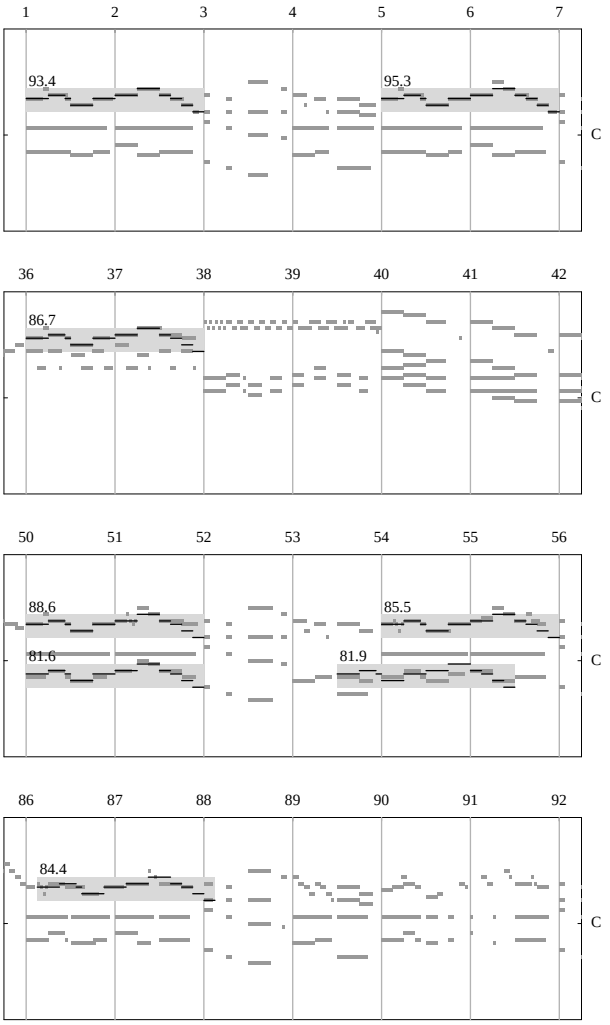

Figure 7. Selected bars showing matches of the theme in Mozart's Piano Sonata.

Our third example is the C minor fugue from Bach's Well-Tempered Clavier, Book I, the first two bars of which are shown in Figure 8. Our polyphonic pattern comes from bar 11—see Figure 9. The top voice of the pattern is the first part of the fugal subject transposed into the relative major key (E♭ major); the bottom voice of the pattern is the corresponding countersubject, which starts a minor 6^{th} and an additional octave below the top voice.

Figure 10 shows selected bars of the result of a search for the best 15 matches. The top 7 matches, with weights above 94%, all appear in these bars. They are very good matches, differing in at most a few semi-tones from the pattern. These matches highlight the occurrences of the pattern in two of the fugal episodes, which modulate to

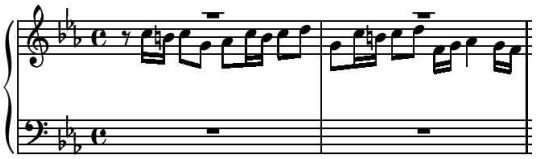

Figure 8. J.S. Bach, Fugue in C minor, Well-Tempered Clavier Book I, bars 1–2.

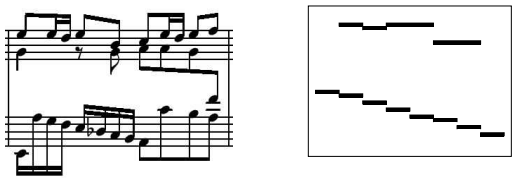

Figure 9. Bar 11 of Bach's Fugue, the source of our pattern, shown at right.

different keys.

Our weight function is lenient with translated notes that fall an octave away from notes of the score. Consequently, we find matches of the pattern where the fugal subject is in the top voice, and the countersubject starts only a minor 6^{th} below the subject in the bottom voice. In Figure 10 these appear in bar 3 and in the second halves of bars 22 and 23. Observe that the latter matches are found twice over, which is logical. (The "phantom" partner of the match in bar 3 has lower weight.) Apart from phantoms, our algorithm properly separates the valid matches from the extraneous ones.

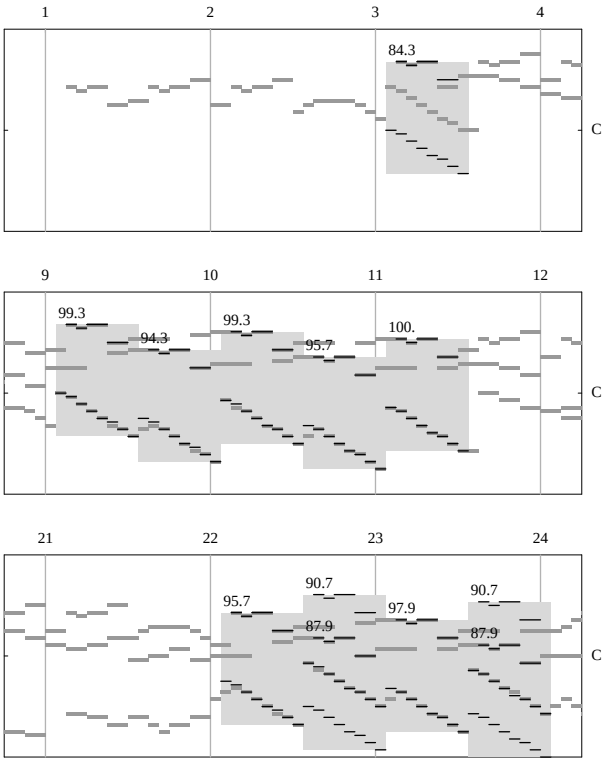

Figure 10. Selected bars showing matches of the pattern in Bach's Fugue.

4.1. Discussion

These examples show that our algorithm can be useful in finding occurrences of a motif in polyphonic music, even when there are melodic variations, and even when the motif is polyphonic. Although our algorithm has the limitation that it does not permit changes to the durations of notes, an example of minor rhythmic variations of a short motif was still detected.

On the other hand, our algorithm misses matches in which the notes of the pattern have been shortened in the score. Our algorithm also produces false positives when a pattern with a very small range of pitches matches into long chord notes of the score. Both issues might be addressed by giving greater weight to the starts of notes. This would alleviate the first problem, since it would diminish the penalty when a pattern note matches to a note of the score of shorter duration. The second problem would also be mitigated, since pattern notes matching into the middle of a note of the score would receive less weight.

Finally, we mention that our implementation was done in the high-level language Mathematica, so it is not sensible to report the computation time for these examples.

5. CONCLUSIONS

In this paper we have explored the possibilities and limitations of an approach to music pattern matching that uses a very natural geometric representation of music, and turns the problem into that of finding a "best" translation of a small set of line segments into a larger set. This geometric approach has been used before, but has not been explored as thoroughly as string matching techniques, even though it successfully deals with polyphony.

Our contribution is to show how this approach can be used together with fairly general weight functions to measure the quality of a match. This opens up the possibility of a rich range of approximate music pattern matching techniques. Our experiments only scratch the surface of what may be possible. It is easy to imagine a variety of enhancements, for example: incorporating information about the key of tonal music; adding information about dynamics and stress; weighting more heavily those matches that occur at the beginnings of notes; allowing the user to specify which notes of the pattern are more important, etc.

Our algorithm runs in time $O(nm(d + \log m))$ where d is the size of the pitch set, n is the size of the score, and m is the size of the pattern. This is fine for small patterns, but too expensive for larger ones. We have argued that the factor $O(nm)$ is likely to be very hard to improve. Improving the dependence on d is perhaps more tractable, and also more relevant for reasonably sized patterns.

6. ACKNOWLEDGEMENTS

We thank Erna Van Daele for musical discussions and advice; Ian Munro for the idea of how to find the best k

matches; and Therese Biedl, Dan Brown, and Anne-Marie Donovan for useful suggestions.

7. REFERENCES

[1] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, D. Rappaport, and G. Toussaint. Computing the similarity of two melodies. In *Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 81–84, 2003.

[2] G. Barequet and S. Har-Peled. Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *International Journal of Computational Geometry and Applications*, 11(4):465–474, 2001.

[3] A. Brinkman and M. Mesiti. Graphic modeling of musical structure. *Computers in Music Research*, 3:1–42, 1991.

[4] D. Byrd and T. Crawford. Problems of music information retrieval in the real world. *Information Processing and Management*, 38:249–272, 2002.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition*. McGraw Hill, 2001.

[6] M.J. Dovey. A technique for "regular expression" style searching in polyphonic music. In *Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR 2001)*, pages 179–185, 2001.

[7] J.S. Downie. Music information retrieval. *Annual Review of Information Science and Technology*, 37:295–340, 2003.

[8] J.S. Downie. Toward the scientific evaluation of music information retrieval systems. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 25–32, 2003.

[9] S. Downie and M. Nelson. Evaluation of a simple and effective music information retrieval method. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 73–80, 2000.

[10] J. Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999(8), 1999.

[11] J. Foote. An overview of audio information retrieval. *Multimedia Systems*, 7:2–11, 1999.

[12] C. Francu and C.G. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *Proc. IEEE International Conference on Multimedia and EXPO (II)*, pages 889–894, 2000.

[13] A. Gajentaan and M.H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry:Theory and Applications*, 5(3):165–185, 1995.

[14] W.B. Hewlett. A base-40 number line representation of musical pitch notation. *Musikometrika*, 50:1–14, 1992.

[15] K. Lemstrom. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Department of Computer Science, 2000.

[16] K. Lemstrom and J. Tarhio. Transposition invariant pattern matching for multi-track strings. *Nordic Journal of Computing*, 10:185–205, 2003.

[17] D. Ó Maidín. A geometrical algorithm for melodic difference. In W.B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*, pages 65–72. MIT Press, 1997–1998.

[18] S. Malinowski. Music animation machine. http://www.well.com/user/smalin/mam.html.

[19] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[20] E. Selfridge-Field. Conceptual and representational issues in melodic comparision. In W.B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*, pages 3–64. MIT Press, 1997–1998.

[21] L. Tanur. A geometric approach to pattern matching in polyphonic music. Master's thesis, School of Computer Science, University of Waterloo, 2004. to appear.

[22] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. van Oostrum. Using transportation distances for measuring melodic similarity. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 107–114, 2003.

[23] E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content-based music retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 193–199, 2003.

[24] G. A. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, pages 283–284, 2002.