Soft Computing 8 (2004) 597–603 © Springer-Verlag 2004
DOI 10.1007/s00500-004-0384-5

Approximate string matching for music analysis

R. Clifford, C. Iliopoulos

Abstract Approximate pattern matching algorithms have become an important tool in computer assisted music analysis and information retrieval. The number of different problem formulations has greatly expanded in recent years, not least because of the subjective nature of measuring musical similarity. From an algorithmic perspective, the complexity of each problem depends crucially on the exact definition of the difference between two strings. We present an overview of advances in approximate string matching in this field focusing on new measures of approximation.

Keywords Music analysis, Pattern matching, String matching, Approximate matching

1 Introduction

Music analysis and information retrieval is a relatively new field that has only recently started to take advantage of related techniques in computer science. With the increased availability of large databases of musical data has come the need to devise methods for searching and analysing this information. Not only must any new techniques be sensitive to the requirements of the user but they must also be efficient in order to be practical. By representing a musical score, for example, as a string or a set of strings it is possible in some cases immediately to apply an existing efficient algorithm from the field of string matching. However, in general an existing method may not be sufficient to find the musically relevant passage or discover the type of repeated pattern that is desired. The cause of this challenge lies in the fact that it is rarely the case that one wishes to find an exact copy of the musical passage in hand. More commonly what is required is to find something that is "similar" in some way. This could mean that it has been transposed by some musical interval to be played in a different key. Or it may have a different rhythm or be missing part of the harmony for example. Or it may simply be that one or two notes are played slightly out of tune or have been recorded incorrectly. This great variety of definitions of musical similarity each potentially require a distinct algorithmic solution when translated to a problem in stringology. Although approximate matching has been widely studied in computer science, the forms of approximation that have been devised have been motivated by the applications to which they were be put. These have varied from textual database searching to the relatively new field of computational biology.

The first survey of methods in computational musicology was published in 1998 [10]. Perhaps the most important innovation since that date has been an increase in the sophistication of the way music analysis problems are represented. For example, where previously music was largely assumed to be monophonic, data structures and algorithms for polyphonic music are now commonplace. Furthermore, distinct representations for *voiced* and *unvoiced* polyphonic music allow more appropriate methods to be applied to each. For situations where pitch and rhythm alone are not sufficient, new methods for searching in sets of high dimensional data straddle the border between computational geometry and stringology. Perhaps even more importantly, formal translations of the concept of *musical similarity* have been extended. In [30] a musical measure of distance between notes was added to the *edit* distance between strings. Recently, similar ideas have been extended to the *hamming* distance as well.

In Sect. 2 we discuss the problem of music representation followed by a short discussion of related literature in Sect. 3. In Sect. 4 we describe the algorithms that have been devised for new approximation measures known as δ, γ and (δ, γ) matching. These attempt to measure the distance between two musical passages by looking at the distance between the individual notes. We then give an overview of a selection of other problems that have recently been considered in Sect. 5. Finally, some future research goals are presented in Sect. 6.

2 Music representation

We consider only music representations that in some sense state which "instruments" are to be played at what time and at what pitch. Many other properties of the music may also be included. This definition is deliberately vague but examples include a score of an orchestral piece or a MIDI representation of a popular music track. Digital recordings of sounds are not included and require a different set of methods for their analysis.

The subject of music representation for use in computer applications has been discussed and debated extensively over the past 20 years (see for example, [33, 6, 17, 25, 36,

Published online: 16 July 2004

R. Clifford (✉), C. Iliopoulos
Algorithm Design Group,
Department of Computer Science King's College,
Strand, London, WC2R 2LS, UK
e-mail: raph@dcs.kcl.ac.uk

34]). Often the multi-dimensional quality of musical information is taken into account to some degree. Parameters may involve pitch[1], duration, loudness, timbre, notational and other features. For the present discussion it is assumed that the values of any musical parameters can be mapped to numeric values. As a simple example, they could be MIDI note-numbers in the range 0 through 127 for chromatic pitch. (Rests can be regarded as a special form of note, and treated no differently; if needed, multiple rests contiguous within a voice can be assumed to have been concatenated into a single rest.) A polyphonic score may be treated in a number of ways. These include considering a score as a collection of melodic strings, each of which is labelled as explicitly belonging to a certain 'voice' (double-stopping is not considered in this discussion); or as a sequence of collections of notes lacking voice information each of which occurs simultaneously within a certain 'time-slot'. Musical data that is derived from a score in conventional musical notation can usually be treated as belonging to the first kind, that derived from MIDI performance may or may not lack explicit voice-information and thus will generally need to be treated as the second kind, while note-data originating from raw audio can at best be expected to conform to an error-ridden form of the second kind. Another distinct representation is to map each note to a point in multi-dimensional space. Each dimension is labelled by one of the properties of the note such as pitch and duration. Although not explicitly a string in its common form this representation may maintain much more of the original data.

The focus of this survey is on algorithms that require string representations of music. In this case it is common to only represent the pitch and order of the notes involved. It is clear that mapping music to strings in this way is not reversible (many different pieces of music will be represented by the same string) and in general it is an important feature of music representation to consider the information lost in translation. Where voicing data is available, polyphonic music will be regarded as a set of monophonic melodies each of which is represented as a single string and so the same questions relating to information loss must be considered. For example, consider a monophonic score translated to a string of integer values each of which represents the pitch of a note. Not only is information about loudness etc. discarded but so is the duration of each note. In some cases, where certain forms of approximate matching are required this may be beneficial; allowing one to find a melody that occurs more than once with different rhythms, for example. In other cases it may result in false positives that obscure the true picture. However, it is important to note that there are no theoretical restrictions on the amount of information that can be represented by a string or set of strings. It is the particular representation chosen that will determine how much information is lost, making it all the more important to consider each application separately.

[1]The pitch of a note may be represented in a number of ways. For example, one may use diatonic or chromatic pitch, a combination of the two or other quantities derived from these values [27, 5]. Alternatively, one could use a fuller representation such as Cambouropoulos's GPIR [7] or a more compact and space-efficient one such as Hewlett's base-40 representation [16].

Two simple alternatives (out of many) for encoding the phrase in Fig. 1 using MIDI pitches alone are as follows:

(1) Three separate strings:
(a) `Voice1: 71,74,72,71,69,71,72,74,72,71,69,67`
(b) `Voice2: 62,62`
(c) `Voice3: 55,54,55,57,54,55`
(2) A single string consisting of a list of sets of notes that begin at the same time:
[(55,62,71), (74), (72), (71), (54,69), (55,71), (57,62,72), (54,74), (55,72), (71), (69), (67)]

3 Related literature

We do not attempt to cover the whole topic of music analysis in this survey. By concentrating on new forms of approximate string matching we are able to give a flavour of one corner of the subject which we believe to be particularly exciting. Amongst the many other areas of current research in computational musicology are the use of probabilistic harmonic models [31, 24] and the application of techniques from computational geometry to music analysis (see [28, 29] for a geometric approach to music analysis). The general topic of measuring musical similarity, which is crucial to the development of more musically relevant algorithms, is surveyed in [32]. Another area of great importance in music analysis is that of perception and cognition of music. In this survey we consider music to be made up of "notes" without discussing how they are heard, perceived or understood. See e.g. [15] for an in depth discussion on the psychology of music.

4 Approximate string matching for music analysis

When the pattern matching task is to find whether one string occurs (approximately or not) as a substring of another string or set of strings, the string to be searched for is called the *pattern* and the string (or set of strings) that is being searched is called the *text*. This terminology will also sometimes be extended to non-string data. When there is no pattern, such as in repetition finding (see Section 5.3), the term *text* will refer to the whole input. We use the convention throughout that p represents the pattern and the length of p is m. Similarly, t represents the text and has length n. The ith element of a pattern (text) will be written p_i (t_i). $t[i \ldots j]$ represents the substring of the t starting at position i and ending at position j, inclusively. We also assume throughout that the elements

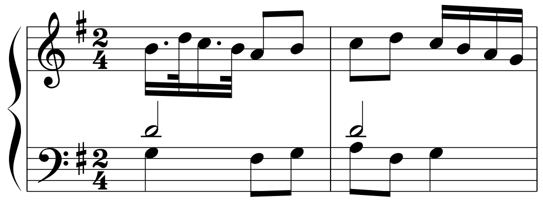

Fig. 1. Simple musical phrase

of all strings are integers. The alphabet is written as Σ and its size σ. For music analysis σ will typically be a constant as Σ may represent either all the notes in the chromatic scale or those in one octave.

The diagrams below that show circles connected by straight lines show only the pitch and order of the notes involved. Where pattern matching is being performed, the black circles represent the pattern and the white circles the text. For repetition finding the black circles represent the substring that is found repeated. The lines joining the circles indicate which notes belong to the same voice.

Approximate matching. Given a sequence of notes, t, find whether a monophonic pattern p occurs approximately in t. Some basic definitions are given here.

Definition. *Two strings p and t are said to be δ-approximate if and only if they are the same length and each value of p is within δ of its corresponding value in t. δ is assumed to be an integer.*

Example. String 99,27,43,12 is δ-approximate to 90,33,47,6 if $\delta = 9$. Clearly if two strings are δ-approximate they will be δ'- approximate for any $\delta' > \delta$.

Definition. *Two strings p and t are said to be γ-approximate if and only if they are the same length and the sum of the absolute differences of the corresponding values of p and t is less than or equal to γ. γ is assumed to be an integer. Note that γ-approximation is a cumulative measure of distance where δ-approximation is applied pointwise.*

Example. String 99,27,43,12 is γ-approximate to 98,27,41,10 if $\gamma = 5$. Clearly if two strings are γ-approximate they will be γ'- approximate for any $\gamma' > \gamma$.

Definition. *Two strings p and t are said to be (δ, γ)-approximate if and only if they are both δ and γ approximate.*

We can now define that problems of δ, γ and (δ, γ) matching.

Definition. *Given a pattern $p = p_1 \dots p_m$ and a text $t = t_1 \dots t_n$, find all indices j such that p* is *δ-approximate to $t[j \dots j+m-1]$. This is known as the δ matching problem.*

γ and (δ, γ) matching are similarly defined. The different approaches for solving these types of approximate matching can be split into three category which we describe here.

4.1 Practical $O(nm)$ methods for δ, γ and (δ, γ) matching

In [11, 8] solutions for δ, γ and (δ, γ) matching are given that run in $O(nm)$ time but are fast in practice. Two different approaches are taken. The first employs bitwise methods known as Shift-And and Shift-Or [3]. By using the fact that calculations on the bits in a single *word* can be performed in parallel, considerable speed-

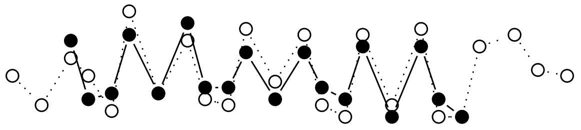
Fig. 2. Approximate matching using δ-matching

ups are possible. In general the time complexity of the best of these algorithms is $O(nm/\omega)$, where ω is the number of bits in a computer word. The second approach adapts classic exact matching algorithms to give the δ-Tuned-Boyer-Moore, δ-Skip-Search and δ-Maximal-Shift algorithms. It has been shown that this approach is faster in practice [11] and so we focus our description on the δ-Tuned-Boyer-Moore algorithm. The modifications required for Skip-Search and Maximal-Shift are similar and are given in detail in [11].

One general approach to exact pattern matching uses a window whose size is equal to the length of the pattern. First the left end of the window and the text are aligned. Then the pattern is checked to see if it matches the text in the window. If there is a mismatch or if more matches are required the window is shifted to the right. This procedure is then repeated until the right end of the window goes beyond the right end of the text. The Tuned-Boyer-Moore algorithm is a very fast practical variant of the famous Boyer-Moore algorithm which uses this approach [4,18]. The shifts are performed using the occurrence shift function. The occurrence shift function is defined for each symbol a in the alphabet Σ as follows:

$$shift(a) = min(\{m - i | p_i = a\} \cup \{m\})$$

The Tuned-Boyer-Moore algorithm gains its efficiency by unrolling three shifts in a very fast skip loop to locate the occurrences of the rightmost symbol of the pattern in the text. Once an occurrence of p_m is found, it checks naively if the whole pattern occurs in the text. Then the shift consists in aligning the rightmost symbol of the window with the rightmost recurrence of p_m in $p_1 \dots p_m$, if any. The length$s$ of this shift is defined as follows:

$$s = min(\{m - i | p_i = p_m \text{ and } i > 0\} \cup \{m\})$$

To perform δ-approximate pattern matching, the shift function can be defined for each symbol a in the alphabet Σ to be the distance from the right end of the pattern to the closest symbol p_i such that $p_i \overset{\delta}{=} a$:

$$shift(a) = min(\{m - i | p_i \overset{\delta}{=} a\} \cup \{m\})$$

Then the length of the shift s becomes:

$$s = min(\{m - i | p_i \overset{2\delta}{=} p_m \text{ and } i > 0\} \cup \{m\})$$

Pseudo code for the δ-Tuned-Boyer-Moore algorithm is given in Figure 4.1. Although the worst case time complexity of all three modified algorithms is $O(nm)$, experimental results comparing the Shift-And and Shift-Or algorithms to those based on modifying fast exact matching algorithms showed that the latter approaches are efficient in practice and considerably faster for small or moderate δ and γ [11].

4.2 Fast fourier transform methods

A quite different approach to δ-matching was taken in [9] where it was shown that the problem can be reduced to several instances of exact matching using wildcards. As exact matching using wildcards can be solved in $O(n \log m)$ time using fast Fourier transforms (FFTs) the overall time complexity is $O(ln \log m)$ time, where l is the number of instances required. The reduction is somewhat complicated and so we describe a simpler approach to δ-matching using FFTs that nonetheless achieves a worst case running time of $O(\delta n \log m)$.

Fact 4.1. *For a pattern $p = p_1 \dots p_m$ and a text $t = t_1 \dots t_n$, it is possible to compute*

$$\sum_{i=1}^{m} p_i t_{i+j-1}$$

for all j, in $O(n \log m)$ time using FFTs. The resulting array is termed the correlation vector of p and t and is written $p \oplus t$.

In order to perform δ-matching we first make the following observation.

Lemma 4.2. For integers x and y, $|x - y| \leq \delta$ if and only if $\prod_{l=-\delta}^{\delta}(x - y + l)^2 = 0$. Therefore, there is a δ-match between p and $t[j \dots j + m - 1]$ if and only if

$$\sum_{i=1}^{m} \prod_{l=-\delta}^{\delta} (p_i - t_{i+j-1} + l)^2 = 0$$

Example 4.3. Consider a pattern p, a text t and $\delta = 1$. Ignoring terms in p_i and t_j alone, $\prod_{l=-1}^{1}(p_i - t_j + l)^2$ evaluates to

$$\begin{aligned} &-6p_i t_j^5 + 15p_i^2 t_j^4 + (8p_i - 20p_i^3)t_j^3 \\ &\quad + (-12p_i^2 + 15p_i^4)t_j^2 + (8p_i^3 - 2p_i - 6p_i^5)t_j \end{aligned}$$

```
δ-Tuned-Boyer-Moore($p, m, t, n, \delta$)
▷ Preprocessing
for all $a \in \Sigma$
    do $shift(a) \leftarrow \min(\{m - i \mid p_i \overset{\delta}{=} a\} \cup \{m\})$
$s \leftarrow \min(\{m - i \mid p_i \overset{2\delta}{=} p_m\} \cup \{m\})$
$t_n \dots t_{n+m-1} \leftarrow (p_m)^m$
▷ Searching
$j \leftarrow m$
while $j \leq n$
    do $k \leftarrow shift(t_j)$
       while $k \neq 0$
           do $j \leftarrow j + k$
              $k \leftarrow shift(t_j)$
              $j \leftarrow j + k$
              $k \leftarrow shift(t_j)$
              $j \leftarrow j + k$
              $k \leftarrow shift(t_j)$
       if $p_1 \dots p_{m-1} \overset{\delta}{=} t_{j-m+1} \dots t_{j-1}$ and $j \leq n$
         then Report($j - m + 1$)
       $j \leftarrow j + s$
```

Fig. 3. δ-Tuned-Boyer-Moore algorithm

We can see from the expression above that in order to find all δ matches between p and t we only require 5 separate correlation vector computations (if $\delta = 1$). Let us call $f(t, n)$ the string formed from t by raising each element to the power n. In other words $f(t, n)_i = t_i^n$ for all i. To compute the correlation vector corresponding to the first term in the sum, for example, we calculate $p \oplus f(t, 5)$. The terms in p_i and t_j alone do not require FFTs as they can be precomputed from each string independently in linear time.

For arbitrary δ it can be shown that the number of different correlation vector computations required is $4\delta + 1$. Therefore, the overall running time for δ-matching using FFTs is $O(\delta n \log m)$.

γ-matching using fast Fourier transforms. Whereas it is possible to solve δ-matching in $O(\delta n \log m)$ time we can solve γ-matching in $O(\sigma n \log m)$ time, where σ is the alphabet size. If σ is $\Omega(\frac{m}{\log m})$ then we have no advantage over the methods of Section 4.1. However, for music analysis it is common practice to reduce the alphabet size by calculating all pitches modulo 12 (the number of semitones in an octave). In this special case the time complexity is $O(n \log m)$ (as $\sigma \leq 12$), albeit with a high constant factor overhead.

Suppose that there are σ distinct symbols in the two strings p and t. Call these symbols $v_1, \dots v_\sigma$. The main stages of the algorithm are as follows:

(1) Create a new string t' of length σn with a 1 at position $\sigma(k - 1) + j$ if $t_k = v_j$.
(2) Now construct a new string p' of length σm with $|v_i - v_j|$ at position $\sigma(k - 1) + j$ for each $j = 1, \dots \sigma$, if $p_k = v_i$.
(3) Now use the FFT to compute $A = p' \oplus t'$.

It is clear that $A_{\sigma j} = \sum_{i=1}^{m} |p_i - t_{i+\sigma j}|$. This solves the γ-matching problem as we need only make a linear time pass over A to find all positions where $A_{\sigma j} \leq \gamma$. If $A_{\sigma j} \leq \gamma$ then there is a γ-match between p and $t[j \dots j + m - 1]$. The overall time complexity is therefore $O(\sigma n \log m)$ as required.

If the problem formulation is changed so that we calculate the *total squared difference* between p and $t[j \dots m]$ then a much simpler $O(n \log m)$ solution can be found.

Definition. Consider a pattern $p = p_1 \dots p_m$ and a text $t = t_1 \dots t_n$ and a bound γ. The *total squared difference* problem is to find all indices j such that

$$\sum_{i=1}^{m} (p_i - t_{i+j-1})^2 \leq \gamma$$

This inequality can be rewritten as $\sum_{i=1}^{m}(p_i^2 - 2p_i t_{i+j-1} + t_{i+j-1}^2)$ which can be computed for all j by preprocessing p and t in linear time and computing $p \oplus t$ in $O(n \log m)$ time using FFTs. Therefore the overall time complexity is $O(n \log m)$.

4.3 Reduction to less-than matching

A third approach to δ-matching is to reduce it to a problem known as the *less-than matching* problem. Less-than matching was introduced in [2] under the name of the *smaller matching problem*. It has been observed recently that any instance of δ-matching can be solved using two instances of less-than matching [1].

Definition. Given a pattern $p = p_1 \ldots p_m$ and a text $t = t_1 \ldots t_n$, find all position j in t such that $p_i \leq t_{i+j-1}$ for all $i \in \{1, \ldots m\}$. In other words, we require all symbols in p to be less than or equal to their corresponding symbols in t. This is known as the *less-than matching* problem.

Lemma 4.4. *Any instance of δ-matching can be reduced to two instances of less-than matching in linear time.*

Proof. Consider a new string p' such that $p'_i = p_i + \delta$ and a string p'' such that $p'' = \delta - p_i$. Compute every index in t where there is a less-than match between p' and t. Now consider a string t' such that $t'_i = -t_i$. Compute every index in t' where there is a less-than match between p'' and t'. The intersection of the two sets of indices gives you all the positions in t for which there is a δ-match with p. The reduction takes linear time as it involves two linear time passes of the input. □

Lemma 4.5. *Less-than matching can be solved in $O(n\sqrt{m}\log m)$ time* [2].

It follows immediately that δ-matching can be solved in $O(\sqrt{m}n \log m)$ time overall. For unbounded δ, (specifically if δ is $\Omega(\frac{\sqrt{m \log m}}{\log n})$) then this is currently the fastest known time complexity for δ-matching.

5 Further problems and techniques related to music analysis

We give an overview of some further problems that have been considered in computational musicology and have been tackled using techniques from string matching. For each one we describe the problem informally and then discuss its current algorithmic status.

5.1 Approximate matching with gaps

Problem. Given a sequence of notes, t, find whether a monophonic pattern p occurs approximately in t, possibly skipping some of the notes in t.

A musical score may contain many notes that are only incidental to the melody. It can be very useful to have a search method that does not require all the notes in the pattern to appear consecutively in the musical piece. This can be achieved by allowing *gaps* to be inserted in the pattern where necessary. The question of how long these gaps should be or whether they should have some other form of constraint on them has led to a number of different formulations of this problem. One example is where the size of the gaps is bounded by a constant integer.

Definition 5.1 *We say that a pattern p occurs approximately in t with α-bounded gaps if and only if p can be found approximately in t by allowing gaps of size no greater than α to be inserted into the pattern.*

It is shown in [12] that α-bounded approximate δ-occurrence matching can be solved in $O(nm)$ time. This bound also holds for a number of other variants including α-bounded (δ, γ)-occurrences and minimising the total sum of the size of the gaps. Exact matching with gaps can be solved using classical dynamic programming methods in $O(nm)$ time.

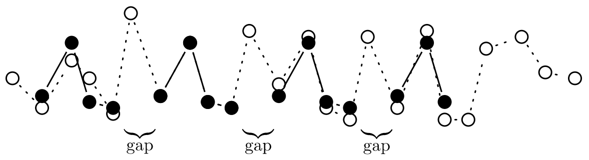

Fig. 4. Approximate matching with gaps using a shorter pattern. The whole pattern is found but gaps are inserted in it to improve the alignment with the text

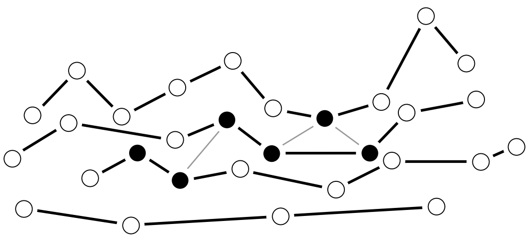
Fig. 5. Polyphonic matching. The pattern is found distributed across the voices

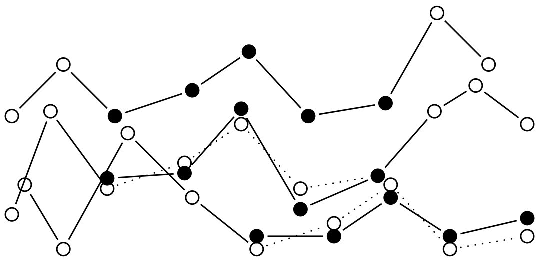
Fig. 6. Approximate repetitions. A pattern in the top voice occurs approximately in the lower two

5.2 Polyphonic matching

Given a set of sequences of notes (one for each voice) and a pattern, find whether the pattern occurs distributed horizontally, either in one voice or across several other voices.

For exact polyphonic matching the problem can be solved using the Modified Shift-Or algorithm in $O(|\Sigma| + m) + O(N)$ time, where $|\Sigma|$ is the number of distinct pitches that can be represented, m is the size of the modified pattern and N is the length of the original score . For approximate distributed matching the time complexity using the Modified Wu-Manber algorithm is $O(|\Sigma| + m + k) + O(n(k + v))$, where v is the number of voices, n is the length of each modified voice, m is the length of the modified pattern and N is the length of the original score [20]. Approximate distance in this case is defined using the *edit distance* or the number of single character insertions, deletions and insertions required to transform the pattern into a substring of the text.

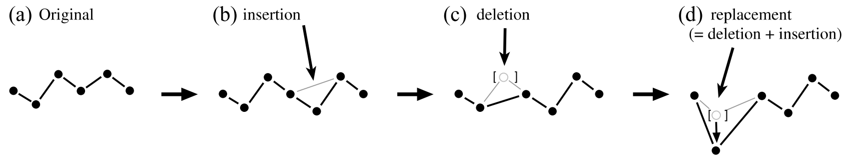

Fig. 7. Different stages of an evolutionary chain

We know of no existing efficient algorithms for δ type approximation measures in polyphonic matching.

5.3 Approximate repetitions

Given a set of sequences of notes (one for each voice), identify repeated patterns that are similar to each other. This is sometimes called the problem of *pattern induction* or *pattern discovery* [2].

The approximate repetition problem has been extensively studied. Algorithms for the *edit* or *hamming distance* [3] may not be appropriate however for music analysis. Solutions for finding approximate repetitions in musical sequences using δ, γ and (δ, γ) matching are given in [8, 21]. For each problem considered the overall running time is $O(n^2)$.

The exact repetition problem can be solved in $O(n \log n)$ time if the positions in the text of the all the repetitions are to be found (see e.g. [23]). If only the identity of the repeated patterns is required (not their locations) then an $O(n)$ solution comes directly from the suffix tree of the text [26, 35].

5.4 Evolutionary chain detection

Given a monophonic sequence of notes t and a monophonic musical pattern p and an integer $k < m/2$, find a sequence $u_1, u_2, \ldots, u_l$ of substrings of t which satisfy the following conditions and for which l is maximal:

1. $u_1 = p$
2. $\delta(u_i, u_{i+1}) \leq k$ for all $i \in \{1, \ldots l-1\}$
3. The start position of u_{i+1} (in t) is to the right of the end position of u_i

Approximate pattern matching will detect an occurrence of an evolving pattern in its early stages. However, once the pattern becomes too distinct from its original form it will no longer be found. By looking for evolutionary chains the whole history of an evolving musical motif can be traced. The best known running time for the longest evolutionary chain problem is $O(n^2 m/\omega)$ (where is ω is the number of bits in a computer word) using a method based on dynamic programming. Several algorithms have also now been developed for variants of this problem including applications to polyphonic music [13, 14, 19, 22]. However, they all use the edit distance and it is an open question whether they can be extended to other distance measures.

[2]It is characteristic of music that typically many repeated patterns may be discovered by these methods (especially in the approximate case). Deciding which of these is perceptually or cognitively pertinent to any music analyst's taste is beyond the scope of this review.

[3]The hamming distance is a commonly used misnomer which refers to the number of mismatches between a pattern and some substring of the text.

6 Future research goals

The main challenges for the future fall broadly into two categories. The first is to improve the time complexity of the problems as currently formulated. The second is to extend these problem definitions to more musically sophisticated form. Amongst the many different avenues for algorithmic research in computational musicology are:

1. Currently the fastest algorithms for δ and γ matching run in $O(n\sqrt{m \log m})$ and $O(nm)$ time in the worst case respectively. However, the only known lower bounds are reductions to FFT computation which takes $O(n \log m)$ time. It is an open question whether any of these time complexities can be improved.
2. Transposition invariant matching and repetition finding allows one to find patterns irrespective of the pitch they are played at. Efficient solutions under different approximation measures is an important new topic of research.
3. Currently the pattern matching algorithms require the whole pattern to be found in the text. Efficient algorithms for partial pattern matching would greatly extend the applicability of the methods devised.

Applying more refined concepts of musical similarity is arguably the most important and difficult step forward that needs to take place to ensure that efficient methods correspond to practical tools.

References

[1] **Amir A** (2004) Private communication
[2] **Amir A, Farach M** (1995) Efficient 2-dimensional approximate matching of half-rectangular figures. Information and Computation **118**(1):1–11
[3] **Baeza-Yates R, Gonnet GH** (1992) A new approach to text searching. Communications of the ACM **35**(10): 74–82
[4] **Boyer RS, Moore JS** (1977) A fast string matching algorithm. Communications of the ACM **20**: 762–772
[5] **Brinkman AR** (1986) A binomial representation of pitch for computer processing of musical data. Music Theory Spectrum **8**: 44–57
[6] **Brinkman AR** (1990) PASCAL Programming for Music Research. The University of Chicago Press, Chicago and London
[7] **Cambouropoulos E** (1996) A general pitch interval representation: Theory and applications. J New Music Research **25**: 231–251
[8] **Cambouropoulos E, Crochemore M, Iliopoulos CS, Mouchard L, Pinzon YJ** (2002) Computing approximate repetitions in musical sequences. International Journal of Computer Mathematics **79**(11): 1135–1148
[9] **Cole R, Iliopoulos C, Lecroq T, Plandowski W, Rytter W** (2003) On special families of morphisms related to

δ-matching and don't care symbols. Information Processing Letters **85**(5): 227–233

[10] **Crawford T, Iliopoulos CS, Raman R** (1998) String-Matching Techniques for Musical Similarity and Melodic Recognition. chapter 3. MIT Press

[11] **Crochemore M, Iliopoulos CS, Lecroq T, Pinzon YJ** (2001) Approximate string matching in musical sequences. In: Balik M, Simanek M (eds) Proceedings of Prague Stringology Club Workshop (PSCW 01), p. 26–36, Czech Technical University, Prague, Czech Republic

[12] **Crochemore M, Iliopoulos CS, Makris C, Rytter W, Tsakalidis A, Tsichlas K** (2002) Approximate string matching with gaps. Nordic J Computing **9**: 54–65

[13] **Crochemore M, Iliopoulos CS, Pinzon YJ** (2000) Fast evolutionary chains. In: Hlavac V, Jeffery KG, Wiedermann J (eds) Proceedings 27–th Software Seminar (Sofsem 2000)—Theory and Practice of Informatics, pp. 306–317, Czech Technical University, Prague, Czech Republic, Springer-Verlag

[14] **Crochemore M, Iliopoulos CS, Pinzon YJ** (2001) Computing evolutionary chains in musical sequences. Electronic Journal of Combinatorics. **8**(2)

[15] **Deutsch D** (ed) (1999) The Psychology of Music. Academic Press, San Diego, 2 edition

[16] **Hewlett WB** (1992) A base-40 number-line representation of musical pitch notation. Musikometrika **4**: 1–14

[17] **Howell P, West R, Cross I** (eds) (1991) Representing Musical Structure. Academic Press, London

[18] **Hume A, Sunday DM** (1991) Fast string searching. Software Practice and Experience **21**(11): 1221–1248

[19] **Iliopoulos CS, Kumar M, Mouchard L, Venkatesh S** (2000) Motif evolution in polyphonic musical sequences. In: Brankovic L, Ryan J (eds) Proceedings 11–th Australasian Workshop on Combinatorial Algorithms, pp. 53–66, University of Newcastle, NSW, Australia

[20] **Iliopoulos CS, Kurokawa M** (2002) Exact and approximate distributed matching for musical melodic recognition. In: Wiggins G (ed) Symposium on AI and Creativity in Arts and Science, Proceedings of AISB 2002, p. 49–56

[21] **Iliopoulos CS, Lecroq T, Mouchard L, Pinzon YJ** (2000) Computing approximate repetitions in musical sequences. In: Balik M, Simanek M (eds) Proceedings of Prague Stringology Club Workshop (PSCW00), pp. 49–59, Czech Technical University, Prague

[22] **Iliopoulos CS, Lemstrom K, Niyad M, Pinzon YJ** (2002) Evolution of musical motifs in polyphonic passages. In: Wiggins G (ed) Symposium on AI and Creativity in Arts and Science, Proceedings of AISB 2002, pp. 67–76

[23] **Karp RM, Miller RE, Rosenberg AL** (1972) Rapid identification of repeated patterns in strings, trees and arrays. In: Proceedings of the 4th ACM Symposium on the Theory of Computing, pp. 125–136, Denver, CO, ACM Press

[24] **Lavrenko V, Pickens J** (2003) Polyphonic music modeling with random fields. In: Proceedings of ACM Multimedia

[25] **Marsden A, Pople A** (eds) (1992) Computer Representations and Models in Music. Academic Press, London

[26] **McCreight EM** (1976) A space–economical suffix tree construction algorithm. Journal of the Association for Computing Machinery **23**: 262–272

[27] **Meredith D** (2001) MIPS: A formal language for the mathematical investigation of pitch systems (version 2001-09-10) Available online at `http://www.titanmusic.com/papers.html`

[28] **Meredith D, Lemström K, Wiggins GA** (2002) Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. Journal of New Music Research, **31**(4): 321–345 Draft available online at `http://www.titanmusic.com/papers/public/siajnmr_submit_2.pdf`

[29] **Meredith D, Wiggins GA, Lemström K** (2001) Pattern induction and matching in polyphonic music and other multidimensional datasets. In: Callaos N, Zong X, Vergez C, Pelaez JR (eds) (2001) Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), July 22-25, volume X, pp. 61–66, Orlando, FL

[30] **Mongeau M, Sankoff D** (1990) Comparison of musical structures. Computers and the Humanities **24**: 161–175

[31] **Pickens J, Juan Pablo Bello GM, Crawford T, Sandler M, Dovey M, Byrd D** (2002) Polyphonic score retrieval using polyphonic audio queries: A harmonic modeling approach. In: ISMIR '02

[32] **Selfridge-Field E, Hewlett W** (1998) Melodic similarity: Concepts, procedures and applications. Computing in Musicology, pp. 11

[33] **Selfridge-Field E** (ed) (1997) Beyond MIDI: The Handbook of Musical Codes. MIT Press, Cambridge, MA

[34] **Smaill A, Wiggins GA, Harris M** (1993) Hierarchical music representation for analysis and composition. Computers and the Humanities, **27**: 7–17. Also from Edinburgh as DAI Research Paper No. 511

[35] **Weiner P** (1973) Linear pattern matching algorithm. In: Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11, Washington, DC

[36] **Wiggins GA, Miranda E, Smaill A, Harris M** (1993) A framework for the evaluation of music representation systems. The Computer Music Journal (CMJ) **17**(3): 31–42. Machine Tongues series, number XVII; Also from Edinburgh as DAI Research Paper No. 658