Computing Approximate Repetitions in Musical Sequences

C. S. Iliopoulos[1] *, T. Lecroq[2] †, L. Mouchard[3] †, Y. J. Pinzon[1] ‡

[1] Dept. Computer Science, King's College London, London WC2R 2LS, England, and School of Computing, Curtin University of Technology, GPO Box 1987 U, WA. Australia
{csi,pinzon}@dcs.kcl.ac.uk,
[2] LIFAR - ABISS, Université de Rouen, 76821 Mont Saint Aignan Cedex, France.
lecroq@dir.univ-rouen.fr
[3] ESA 6037: Dept. of Vegetal Physiology - ABISS, Université de Rouen, 76821 Mont Saint Aignan Cedex, France and School of Computing, Curtin University of Technology, GPO Box 1987 U, WA., Australia
lm@dir.univ-rouen.fr

e-mail:

Abstract. Here we present new algorithms for computing all δ-approximate and (δ, γ)-approximate repetitions in musical sequences. We also present algorithms for computing the longest δ-approximate repeats, (δ, γ)-approximate repeats and minimum-tolerance powers (a "repeat" is a repetition variant).

Key words: String algorithms, approximate string matching, dynamic programming, computer-assisted music analysis.

1 Introduction

The approximate repetition problem has been extensively studied over the last few years. Such problem can be found in computational biology, information retrieval, musical analysis and compression. This paper focuses in one type of repetition that arise especially in musical information retrieval, i.e. δ-approximate repetitions. A musical score can be viewed as a string: at a very rudimentary level, the alphabet could simply be the set of notes in the chromatic or diatonic notation, or the set of intervals that appear between notes (e.g. pitch may be represented as MIDI numbers and pitch intervals as number of semitones). Approximate repetitions in one or more musical works play a crucial role in discovering similarities between different musical entities and may be used for establishing "characteristic signatures" (see [3]).

*Partially supported by the Royal Society Grant CCSLAAR.
†Partially supported by the C.N.R.S. Program "Génomes".
‡Partially supported by the University of London Central Research Fund (CRF).

Furthermore, efficient algorithms for computing the approximate repetitions are also directly applicable to molecular biology (see [4, 5, 7]) and in particular in DNA sequencing by hybridization ([8]), reconstruction of DNA sequences from known DNA fragments (see [10, 11]), in human organ and bone marrow transplantation as well as the determination of evolutionary trees among distinct species ([10]).

The approximate matching problem has been used for a variety of musical applications (see overviews in McGettrick [6]; Crawford et al [3]; Rolland et al [9]; Cambouropoulos et al [1]). It is known that exact matching cannot be used to find occurrences of a particular melody. Approximate matching should be used in order to allow the presence of errors. The number of errors allow will be referred to as δ.

The paper is organised as follows. In the next section we present some basic definitions for strings and background notions for approximate matching. In Section 3 we present algorithms for computing δ-approximate repetitions and in Section 4 for computing (δ, γ)-approximate repetitions. In Section 5 we present algorithms for computing another variant of the above repetitions: longest δ-approximate repeats, (δ, γ)-approximate repeats and minimum-tolerance powers. Finally in Section 6 we present our conclusions and open problems.

2 Background and basic string definitions

A *string* is a sequence of zero or more symbols from an alphabet Σ; the string with zero symbols is denoted by ϵ. The set of all strings over the alphabet Σ is denoted by Σ^*. A string x of length n is represented by $x_1 \dots x_n$, where $x_i \in \Sigma$ for $1 \leq i \leq n$. A string w is a *substring* of x if $x = uwv$ for $u, v \in \Sigma^*$; we equivalently say that the string w occurs at position $|u| + 1$ of the string x. The position $|u| + 1$ is said to be the *starting position* of w in x and the position $|w| + |u|$ the *end position* of w in x. A string w is a *prefix* of x if $x = wu$ for $u \in \Sigma^*$. Similarly, w is a *suffix* of x if $x = uw$ for $u \in \Sigma^*$.

The string xy is a *concatenation* of two strings x and y. The concatenations of k copies of x is denoted by x^k. For two strings $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$ such that $x_{n-i+1} \dots x_n = y_1 \dots y_i$ for some $i \geq 1$, the string $x_1 \dots x_n y_{i+1} \dots y_m$ is a *superposition* of x and y. We say that x and y *overlap*.

Let x be a string of length n. The integer p is said to be a *period* of x, if $x_i = x_{i+p}$ for all $1 \leq i \leq n - p$. *The period* of a string x is the smallest period of x. A string y is a *border* of x if y is a prefix and a suffix of x.

Let Σ be an alphabet of integers and δ an integer. Two symbols a, b of Σ are said to be δ-approximate, denoted $a \stackrel{\delta}{=} b$ if and only if

$$|a - b| \leq \delta$$

We say that two strings x, y are δ-approximate, denoted $x \stackrel{\delta}{=} y$ if and only if

$$|x| = |y|, \text{ and } x_i \stackrel{\delta}{=} y_i, \ \forall i \in \{1, \dots, |x|\} \tag{2.1}$$

For a given integer γ we say that two strings x, y are γ-approximate, denoted $x \overset{\gamma}{=} y$ if and only if

$$|x| = |y|, \text{ and } \sum_{1}^{|x|} |x_i - y_i| < \gamma \tag{2.2}$$

Furthermore, we say that two strings x, y are $\{\gamma, \delta\}$-approximate, denoted $x \overset{\gamma,\delta}{=} y$, if and only if x and y satisfy conditions (2.1) and (2.2).

3 Computing δ-Approximate Repetitions

The problem of computing all *δ-approximate repetitions* is formally defined as follows: given a string $t{=}t_1 \ldots t_n$ and integers δ and m, compute all positions j of t, that there exists a string $\hat{t}$ of length m such that

$$\begin{gathered} t[j..j+m-1] \overset{\delta}{=} \hat{t} \\ t[j+m..j+2m-1] \overset{\delta}{=} \hat{t} \\ \vdots \\ t[j+(\ell-1)m..j+\ell m-1] \overset{\delta}{=} \hat{t} \end{gathered}$$

where $\hat{t}$ and ℓ are said to be the *root* and the *power* of the repetition respectively.

When we look for a repetition we will run into two possibilities: the root does or does not occur necessarily in the text. In this section we study the first case, that is, $\hat{t} = t[j..j+m-1]$ (exact matching) for some $j \in \{1, \ldots, n-m-1\}$. We state as an open problem the second case when the root does not occur necessarily in the text.

Let $D[0..n, 0..n]$ be the δ-matrix such that

$$D(i,j) \leftarrow \sum_{k=1}^{m} \delta(t_{i-m+k}, t_{j-m+k}) \;\; \forall (i,j) \in [0..n] \times [0..n]$$

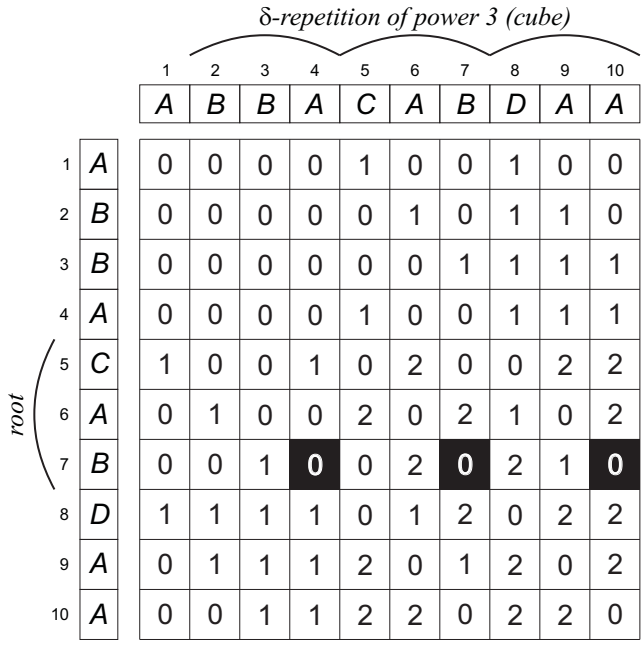
where $\delta(t_i, t_j)$ is 0 if and only if $t_i \overset{\delta}{=} t_j$ and 1 otherwise.

Example. Table 1 shows the δ-matrix for $t = ABBACABDAA$, δ=1 and m=3. Row 7 shows that $D(7,4) = D(7,7) = D(7,10) = 0$, which means that there is a δ-repetition of power 3 (δ-cube) starting at position 2 ($BBA.CAB.DAA$) with root (CAB) starting at position 5.

Note that our algorithm will be seeking those cells with $D(i,j) = 0$ since $u_i \overset{\delta}{=} u_j$ and therefore candidates for belonging to a repetition. The basic steps of the algorithm are as follows:

1. Computation of the δ-matrix $D[0..n, 0..n]$

$$D(i,j) \leftarrow \begin{cases} 0 & , \text{ if } i,j = 0 \\ D(i-1,j-1) + \delta(t_i, t_j) & , \text{ if } 0 < i,j < m \\ D(i-1,j-1) + \delta(t_i, t_j) - \delta(t_{i-m}, t_{j-m}), & \text{otherwise} \end{cases}$$



		1	2	3	4	5	6	7	8	9	10
		A	B	B	A	C	A	B	D	A	A
1	A	0	0	0	0	1	0	0	1	0	0
2	B	0	0	0	0	0	1	0	1	1	0
3	B	0	0	0	0	0	0	1	1	1	1
4	A	0	0	0	0	1	0	0	1	1	1
5	C	1	0	0	1	0	2	0	0	2	2
6	A	0	1	0	0	2	0	2	1	0	2
7	B	0	0	1	0	0	2	0	2	1	0
8	D	1	1	1	1	0	1	2	0	2	2
9	A	0	1	1	1	2	0	1	2	0	2
10	A	0	0	1	1	2	2	0	2	2	0

Table 1: The δ-matrix D for $t = ABBACABDAA$, δ=1 and m=3.

2. Computation of the index matrix $I[0..n, 0..n]$ defined as follows:

$$I(i,j) \leftarrow \mathrm{mod}(|j - \mathrm{mod}(i,m)|, m) \;\; \forall (i,j) \in [1..n] \times [1..n]$$

3. We say that there is a δ-approximate repetition of power ℓ starting at position j with root $\hat{t}$ starting at position i if and only if $D(i+m-1, j+km-1) = 0$ for $k \in \{1, \ldots, \ell\}$. In other words, we are looking for runs of zeros (in each row) with the same index value.

Table 2 shows all the δ-approximate repetitions after considering all rows.

s	r	Root	Repetition	Power
1	1	ABB	$ABB.ACA$	2
1	2	BBA	$ABB.ACA$	2
2	2	BBA	$BBA.CAB$	2
3	3	BAC	$BAC.ABD$	2
3	6	ABD	$BAC.ABD$	2
4	7	BDA	$ACA.BDA$	2
5	8	DAA	$CAB.DAA$	2
1	4	ACA	$ABB.ACA.BDA$	3
2	5	CAB	$BBA.CAB.DAA$	3

Table 2: δ-repetitions for t=$ABBACABDAA$, δ=1 and m=3. Note that s denotes the starting position of the repetition and r denotes the starting position of the root.

3.1 Pseudo-code

Fig. 1 show the pseudo-code for computing all δ-approximate repetitions. The algorithm was optimized to use $O(n)$ space instead of $O(n^2)$. This is possible because the computation of each row only depends on the previous one. The array a is of length n and it stores the current row of D. Also, $r[i].start$ holds the starting position of the repetition for those cells with index i in the current row (array a). In a similar way, $r[i].power$ holds the power and $r[i].root$ holds the starting position of the root.

```
δ-REPETITIONS(t, δ, m)   ▷ n = |t|
1   for i ← 0 until n do
2     for j ← n − 1 until 0 step -1 do
3       k ←   mod (|j −   mod (i, m)|, m)
4       if |t_i − t_j| > δ then a[j] ← a[j − 1] + 1
5       if i − m ≥ 0 and |t_{i−m} − t_{j−m}| > δ then a[j] ← a[j] − m
6       if j ≥ n − m then
7         r[k].root ← i − m + 2
8         r[k].power ← 0
9       if a[j] = 0 then
10        r[k].power ← r[k].power + 1
11        r[k].start ← j − m + 2
12        if j < 2m − 1 and r[k].power > 1 then
13          write "Repetition power", r[k].power, "at", r[k].start,
              "with root at", r[k].root
14      else
15        if r[k].power > 1 then
16          write "Repetition power", r[k].power, "at", r[k].start,
              "with root at", r[k].root
17        r[k].power ← 0
```

Figure 1: The δ-REPETITIONS algorithm.

3.2 Running time

The time complexity of the algorithm is easily seen to be $O(n^2)$ and the space complexity is $O(n)$.

4 Computing (δ, γ)-Approximate Repetitions

The problem of computing all (δ, γ)-*approximate repetitions* is formally defined as follows: given a string $t{=}t_1 \ldots t_n$ and integers δ, γ and m, compute all positions j of t, that there exists a string $\hat{t}$ such that

$$\begin{gathered} t[j..j+m-1] \stackrel{\delta,\gamma}{=} \hat{t} \\ t[j+m..j+2m-1] \stackrel{\delta,\gamma}{=} \hat{t} \\ \vdots \\ t[j+(\ell-1)m..j+\ell m-1] \stackrel{\delta,\gamma}{=} \hat{t} \end{gathered}$$

If we know where the δ-approximate repetitions are, then next we need to discard somehow those repetitions that are not (δ,γ)-approximate repetitions. We can extend the δ-approximate repetition algorithm to the (δ,γ)-approximate repetition problem by adding some information about γ. This information will be stored in the γ-matrix $G[0..n, 0..n]$ so that

$$G(i,j) \leftarrow \sum_{k=1}^{m} |t_{i-m+k} - t_{j-m+k}| \quad \forall (i,j) \in [0..n] \times [0..n]$$

We say that $u_i \stackrel{\delta,\gamma}{=} u_j$ when $D(i,j) = 0$ and $G(i,j) \leq \gamma$.

The additional steps of the algorithm are as follows:

1. Computation of the γ-matrix $G[0..n, 0..n]$
$$G(i,j) \leftarrow \begin{cases} 0 & , \text{ if } i,j = 0 \\ G(i-1,j-1) + |t_i - t_j| & , \text{ if } 0 < i,j < m \\ G(i-1,j-1) + |t_i - t_j| - |t_{i-m} - t_{j-m}|, & \text{otherwise} \end{cases}$$

2. We say that there is a (δ,γ)-approximate repetition of power ℓ starting at position j with root $\hat{t}$ starting at position i iff $D(i+m-1, j+km-1)$=0 and $G(i+m-1, j+km-1) \leq \gamma$ for $k \in \{1, \ldots, \ell\}$.

Example. Table 3 shows the γ-matrix G for $t = ABBACABDAA$, δ=1, γ=2 and m=3. We know there is a δ-cube starting at position 2 ($BBA.CAB.DAA$) with root starting at position 5 (CAB). This δ-cube can be a (δ,γ)-cube only if $G(7,4) \leq 2$, $G(7,7) \leq 2$ and $G(7,10) \leq 2$. However $G(7,4) = 3$ and we conclude that this δ-cube is not a (δ,γ)-cube. But if we look at row 6, we see that there is a (δ,γ)-approximate repetition of power 3 ((δ,γ)-cube) starting at position 1 ($ABB.ACA.BDA$) with root (ACA) starting at position 4.

Table 4 shows all the (δ,γ)-approximate repetitions after considering all rows in G.

4.1 Pseudo-code

Fig. 2 shows the pseudo-code for computing all (δ,γ)-approximate repetitions. The algorithm was also optimized to use $O(n)$ space instead of $O(n^2)$. This is possible because the computation of each row only depends on the previous one.

The array a is of length n and it stores the current rows of D and g ($a.delta$ for D and $a.gamma$ for G in Fig. 2). Moreover as in δ-approximate repetitions , $r[i].start$ holds the starting position of the repetition for those cells with index i in the current row (array a), $r[i].power$ holds the power and $r[i].root$ the holds the starting position of the root.

(δ,γ)-repetition of power 3 (cube)

root

		1	2	3	4	5	6	7	8	9	10
		A	B	B	A	C	A	B	D	A	A
1	A	0	1	1	0	2	0	1	3	0	0
2	B	1	0	1	2	1	3	0	3	4	1
3	B	1	1	0	2	3	2	3	2	4	5
4	A	0	2	2	0	3	2	3	4	2	3
5	C	2	1	3	3	0	5	2	3	5	4
6	A	0	3	**2**	2	5	**0**	5	4	**2**	5
7	B	1	0	3	3	2	5	0	5	5	2
8	D	3	3	2	4	3	4	5	0	6	7
9	A	0	4	4	2	5	2	5	6	0	5
10	A	0	1	5	3	4	5	2	7	5	0

Table 3: The γ-matrix G for $t = ABBACABDAA$, δ=1, γ=2 and m=3.

s	r	Root	Repetition	Power
1	1	ABB	$ABB.ACA$	2
1	2	BBA	$ABB.ACA$	2
4	7	BDA	$ACA.BDA$	2
5	8	DAA	$CAB.DAA$	2
1	4	ACA	$ABB.ACA.BDA$	3

Table 4: (δ, γ)-approximate repetitions for t=$ABBACABDAA$, δ=1, γ=2 and m=3. Note that s denotes the starting position of the repetition and r denotes the starting position of the root.

4.2 Running time

The time complexity of the algorithm is easily seen to be $O(n^2)$ and the space complexity is $O(n)$.

5 Computing the Longest δ-Approximate and (δ, γ)-Approximate Repeat

The problem of computing the *longest δ-approximate repeats* (LDAR) is defined as follows: given a text t of length n, and integers m and δ, find whether there exist a sequence of substrings $u_1, u_2, \ldots, u_\ell$ of t that satisfy the following conditions:

1. $u_i \overset{\delta}{=} u_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$

```
(δ, γ)-Repetitions(t, δ, γ, m) ▷ n = |t|
1   for i ← 0 until n do
2     for j ← n − 1 until 0 step -1 do
3       k ←  mod (|j −  mod (i, m)|, m)
4       a[j].delta ← a[j − 1].delta + δ(t_i, t_j)
5       a[j].gamma ← a[j − 1].gamma + |t_i − t_j|
6       if i − m ≥ 0 then
7         a[j].delta ← a[j].delta − δ(t_{i−m}, t_{j−m})
8         a[j].gamma ← a[j].gamma − |t_{i−m}, t_{j−m}|
9       if j ≥ n − m then
10        r[k].root ← i − m + 2
11        r[k].power ← 0
12      if a[j].delta = 0 and a[j].gamma ≤ γ then
13        r[k].power ← r[k].power + 1
14        r[k].start ← j − m + 2
15        if j < 2m − 1 and r[k].power > 1 then
16          write "Repetition power", r[k].power, "at", r[k].start,
            "with root at", r[k].root
17      else
18        if r[k].power > 1 then
19          write "Repetition power", r[k].power, "at", r[k].start,
            "with root at", r[k].root
20        r[k].power ← 0
```

Figure 2: The (δ, γ)-Repetitions algorithm.

2. $t = ru_1u_2 \dots u_\ell s$, for some strings r, s.

3. Maximizes ℓ

Note that in case of δ-approximate repetitions *every* repetition is δ-approximate to the root. But in the case of repeats, each repeat is guaranteed to be δ-approximate only to its neighbour.

The method for finding the LDAR is based on the construction of the matrix D presented in the previous section. We can construct m lists L_q, $q = 1, 2, ..., m$ such that

$$L_q[i] := D((i-1)m + q, im + q) \quad i = 1, 2, ...$$

It is not difficult to see that the longest repeats corresponds to the longest subsequence of 0's in one of the L_q's.

Example. Let $t = DCCADCADCBEDCAA$, m=3 and $\delta = 2$. Table 5 shows the δ-matrix needed to compute to lists $L_1 := \{1, 0, 0, 2\}$, $L_2 := \{0, 0, 0\}$ and $L_3 := \{0, 0, 1\}$. The longest subsequence of 0's occurs in L_2 and corresponds to the repeat $(CCA.DCA.DCB.EDC)$.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		D	C	C	A	D	C	A	D	C	B	E	D	C	A	A
1	D	0	0	0	1	0	0	1	0	0	1	0	0	0	1	1
2	C		0	0	1	1	0	1	1	0	0	2	0	0	1	2
3	C			0	1	1	**1**	1	1	1	0	1	2	0	1	2
4	A				0	2	2	**0**	2	2	0	1	2	3	0	1
5	D					0	2	2	**0**	2	2	0	1	2	4	1
6	C						0	2	2	**0**	1	3	0	1	3	5
7	A							0	2	2	**0**	2	4	1	1	3
8	D								0	2	2	**0**	2	4	2	2
9	C									0	1	3	**0**	2	5	3
10	B										0	2	4	**0**	2	5
11	E											0	2	5	**1**	3
12	D												0	2	6	**2**
13	C													0	3	7
14	A														0	3
15	A															0

Table 5: The δ-matrix D for $t = DCCADCADCBEDCAA$, δ=2 and m=3.

The problem of computing the *longest* (δ, γ)*-approximate repeats* (LDGAR) is defined as follows: given a text t of length n, and integers m. δ and γ, find whether there exist a sequence of substrings $u_1, u_2, \ldots, u_\ell$ of t that satisfy the following conditions:

1. $u_i \overset{\delta,\gamma}{=} u_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$
2. $t = r u_1 u_2 \ldots u_\ell s$, for some strings r, s.
3. Maximizes ℓ

The method for finding the LDGAR is based on the construction of the matrix G presented in the previous section. We can construct m lists F_q, $q = 1, 2, ..., m$ such that

$$F_q[i] := \begin{cases} 0, & \text{if } G((i-1)m + q, im + q) \leq \gamma \\ 1, & \text{otherwise} \end{cases} \qquad i = 1, 2, ...$$

The longest repeats is the longest subsequence of 0's in $L_q + F_q$.

Example. Table 5 shows the γ-matrix for the above example. Let γ =2 so that $F_1 := \{1, 0, 1, 1\}$, $F_2 := \{0, 0, 1\}$ and $F_3 := \{0, 0, 1\}$. Now, considering both lists (L and F) we see that the longest repeat is in either L_2/F_2 or L_3/F_3 and they are ($CCA.DCA.DCB$) and ($CAD.CAD.CBE$) respectively.

The problem of the *minimum-tolerance power* (MTP) is as follows: given a text t of length n, and integers m, p and δ, find whether there exist the sequence of substrings $u_1, u_2, \ldots, u_p$ of t that satisfy the conditions:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		D	C	C	A	D	C	A	D	C	B	E	D	C	A	A
1	D	0	1	1	3	0	1	3	0	1	2	1	0	1	3	3
2	C		0	1	3	4	0	3	4	0	2	4	2	0	3	5
3	C			0	3	4	**4**	2	4	4	1	4	5	2	2	5
4	A				0	5	5	**1**	5	5	2	5	6	5	1	2
5	D					0	6	6	**0**	6	5	2	5	6	6	3
6	C						0	6	6	**0**	5	6	3	4	6	7
7	A							0	6	6	**1**	6	7	4	0	3
8	D								0	6	5	**2**	5	6	6	3
9	C									0	5	6	**3**	4	6	7
10	B										0	5	6	**3**	1	4
11	E											0	5	6	**6**	5
12	D												0	5	7	**8**
13	C													0	4	7
14	A														0	3
15	A															0

Table 6: The γ-matrix G for $t = DCCADCADCBEDCAA$, δ=2 and m=3.

1. $u_i \overset{\delta,\gamma}{=} u_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$
2. $t = ru_1u_2 \ldots u_\ell s$, for some strings r, s.
3. Minimizes
$$\sum_{i=1}^{p-1} \gamma(u_i, u_{i+1})$$

The computation of the MTP is based on the list F.

5.1 Running time

The time complexity of the algorithms for computing the LDAR, LDGAR and MTP is easily seen to be dominated by the complexity of the computation of the matrices D and G. Hence, the overall complexity for the both problems will be $O(n^2)$.

6 Conclusion and Open problems

Here we have presented new essentially quadratic algorithms for computing δ-approximate and (δ, γ)-approximate repetitions, the longest δ-approximate and (δ, γ)-approximate repeat and the longest minimum-tolerance repeats.

An interesting open problem is to compute δ-approximate and (δ, γ)-approximate repetitions where the root does not belongs to the text.

References

[1] E. Cambouropoulos, T. Crawford and C.S. Iliopoulos, (1999) Pattern Processing in Melodic Sequences: Challenges, Caveats and Prospects. In Proceedings of the AISB'99 Convention (Artificial Intelligence and Simulation of Behaviour), Edinburgh, U.K., pp. 42–47 (1999).

[2] E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, L. Mouchard, and Y. J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. In R. Raman and J. Simpson, editors, *Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms*, pages 129–144, Perth, WA, Australia, 1999.

[3] T. Crawford, C. S. Iliopoulos and R. Raman, String Matching Techniques for Musical Similarity and Melodic Recognition, *Computing in Musicology*, Vol 11 (1998) 73–100.

[4] V. Fischetti, G. Landau, J. Schmidt and P. Sellers, Identifying periodic occurences of a template with applications to protein structure, *Proc. 3rd Combinatorial Pattern Matching*, Lecture Notes in Computer Science, vol. 644, 1992, pp. 111–120.

[5] S. Karlin, M. Morris, G. Ghandour, and M. Y. Leung, Efficients algorithms for molecular sequences analysis, *Proc. Natl. Acad. Sci.*, USA (1988) 85:841–845

[6] P. McGettrick, MIDIMatch: Musical Pattern Matching in Real Time. MSc Dissertation, York University, U.K. (1997).

[7] A. Milosavljevic and J. Jurka, Discovering simple DNA sequences by the algorithmic significance method, *Comput. Appl. Biosci.* (1993) 9:407–411

[8] P. A. Pevzner and W. Feldman, Gray Code Masks for DNA Sequencing by Hybridization, *Genomics*, 23, 233–235 (1993).

[9] P.Y. Rolland and J.G. Ganascia, Musical Pattern Extraction and Similarity Assessment. *In Readings in Music and Artificial Intelligence*. E. Miranda. (ed.). Harwood Academic Publishers (forthcoming) (1999).

[10] J. P. Schmidt, All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings, in *Proc. of the Fifth Symposium on Combinatorial Pattern Matching* CPM'94, Lecture Notes in Computer Science (1994).

[11] S. S. Skiena and G. Sundaram, Reconstructing strings from substrings, *J. Computational Biol.* 2 (1995) 333–353.