

# Computing Approximate Repetitions in Musical Sequences

C. S. Iliopoulos<sup>1 \*</sup>, T. Lecroq<sup>2 \*\*</sup>, L. Mouchard<sup>3 \*\*\*</sup>, and Y. J. Pinzon<sup>1 †</sup>

<sup>1</sup> 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,`

`www.dcs.kcl.ac.uk/staff/csi,` `www.dcs.kcl.ac.uk/pg/pinzon`

<sup>2</sup> LIFAR - ABISS, Université de Rouen, 76821 Mont Saint Aignan Cedex, France.

`lecroq@dir.univ-rouen.fr`

`perso.normandnet.fr/lecroq`

<sup>3</sup> 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`

`www.dir.univ-rouen.fr/~lm`

**Abstract.** Here we present new algorithms for computing all  $\delta$ -approximate and  $(\delta, \gamma)$ -approximate repetitions in musical sequences. We also present algorithms for computing the longest  $\delta$ -approximate repeats,  $(\delta, \gamma)$ -approximate repeats and minimum-tolerance powers (a “repeat” is a repetition variant).

**Keywords:** 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.  $\delta$ -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

---

\* Partially supported by the Royal Society Grant CCSLAAR.

\*\* Partially supported by the C.N.R.S. Program “Génomes”

\*\*\* Partially supported by the C.N.R.S. Program “Génomes”

† Partially supported by the University of London Central Research Fund (CRF)

discovering similarities between different musical entities and may be used for establishing “characteristic signatures” (see [3]).

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 allowed will be referred to as  $\delta$ .

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  $\delta$ -approximate repetitions and in Section 4 for computing  $(\delta, \gamma)$ -approximate repetitions. In Section 5 we present algorithms for computing another variant of the above repetitions: longest  $\delta$ -approximate repeats,  $(\delta, \gamma)$ -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  $\Sigma$ ; the string with zero symbols is denoted by  $\epsilon$ . The set of all strings over the alphabet  $\Sigma$  is denoted by  $\Sigma^*$ . 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  $x = uvw$  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  $\Sigma$  be an alphabet of integers and  $\delta$  an integer. Two symbols  $a, b$  of  $\Sigma$  are said to be  $\delta$ -approximate, denoted  $a \stackrel{\delta}{=} b$  if and only if

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

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

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

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

$$|x| = |y|, \text{ and } \sum_1^{|x|} |x_i - y_i| \leq \gamma \quad (2.2)$$

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

### 3 Computing $\delta$ -Approximate Repetitions

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

$$\begin{aligned} t[j..j+m-1] &\stackrel{\delta}{\approx} \hat{t} \\ t[j+m..j+2m-1] &\stackrel{\delta}{\approx} \hat{t} \\ &\vdots \\ t[j+(\ell-1)m..j+\ell m-1] &\stackrel{\delta}{\approx} \hat{t} \end{aligned}$$

where  $\hat{t}$  and  $\ell$  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  $j \in \{1, \dots, n-m-1\}$ . 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  $\delta$ -matrix such that

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

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

*Example.* Table 1 shows the  $\delta$ -matrix for  $t = ABBACABDAA$ ,  $\delta=1$  and  $m=3$ . Row 7 shows that  $D(7, 4) = D(7, 7) = D(7, 10) = 0$ , which means that there is a  $\delta$ -repetition of power 3 ( $\delta$ -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  $u_i \stackrel{\delta}{\approx} u_j$  and therefore candidates for belonging to a repetition. The basic steps of the algorithm are as follows:

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

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

-repetition of power 3 (cube)

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

  

root	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  $\delta$ -matrix  $D$  for  $t = ABBACABDAA$ ,  $\delta=1$  and  $m=3$ .

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

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

3. We say that there is a  $\delta$ -approximate repetition of power  $\ell$  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  $k \in \{1, \dots, \ell\}$ . In other words, we are looking for runs of zeros in each row but considering the index computed in step 2.

Table 2 shows all the  $\delta$ -approximate repetitions after considering all rows.

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

**Table 2.**  $\delta$ -repetitions for  $t=ABBACABDAA$ ,  $\delta=1$  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 shows the pseudo-code for computing all  $\delta$ -approximate repetitions. The algorithm was optimized to use  $\mathcal{O}(n)$  space instead of  $\mathcal{O}(n^2)$ . 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.

```

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

```

Fig. 1. The  $\delta$ -REPETITIONS algorithm.

### 3.2 Running time

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

## 4 Computing $(\delta, \gamma)$ -Approximate Repetitions

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

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

If we know where the  $\delta$ -approximate repetitions are, then next we need to discard somehow those repetitions that are not  $(\delta, \gamma)$ -approximate repetitions. We can extend the  $\delta$ -approximate repetition algorithm to the  $(\delta, \gamma)$ -approximate repetition problem by adding some information about  $\gamma$ . This information will be stored in the  $\gamma$ -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  $\gamma$ -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  $(\delta, \gamma)$ -approximate repetition of power  $\ell$  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  $k \in \{1, \dots, \ell\}$ .

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

		( $\delta, \gamma$ )-repetition of power 3 (cube)										
		1	2	3	4	5	6	7	8	9	10	
		A	B	B	A	C	A	B	D	A	A	
root	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  $\gamma$ -matrix  $G$  for  $t = ABBACABDAA$ ,  $\delta=1$ ,  $\gamma=2$  and  $m=3$ .

Table 4 shows all the  $(\delta, \gamma)$ -approximate repetitions after considering all rows in  $G$ .

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

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

#### 4.1 Pseudo-code

Fig. 2 shows the pseudo-code for computing all  $(\delta, \gamma)$ -approximate repetitions. The algorithm was also optimized to use  $\mathcal{O}(n)$  space instead of  $\mathcal{O}(n^2)$ . 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  $\delta$ -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.

```

( $\delta, \gamma$ )-REPETITIONS( $t, \delta, \gamma, m$ )  $\triangleright n = |t|$ 
1  for  $i \leftarrow 0$  until  $n$  do
2      for  $j \leftarrow n - 1$  until  $0$  step  $-1$  do
3           $k \leftarrow \text{mod}(|j - \text{mod}(i, m)|, m)$ 
4           $a[j].delta \leftarrow a[j - 1].delta + \delta(t_i, t_j)$ 
5           $a[j].gamma \leftarrow a[j - 1].gamma + |t_i - t_j|$ 
6          if  $i - m \geq 0$  then
7               $a[j].delta \leftarrow a[j].delta - \delta(t_{i-m}, t_{j-m})$ 
8               $a[j].gamma \leftarrow a[j].gamma - |t_{i-m}, t_{j-m}|$ 
9          if  $j \geq n - m$  then
10              $r[k].root \leftarrow i - m + 2$ 
11              $r[k].power \leftarrow 0$ 
12             if  $a[j].delta = 0$  and  $a[j].gamma \leq \gamma$  then
13                  $r[k].power \leftarrow r[k].power + 1$ 
14                  $r[k].start \leftarrow j - m + 2$ 
15                 if  $j < 2m - 1$  and  $r[k].power > 1$  then
16                     write "Repetition power",  $r[k].power$ , "at",  $r[k].start$ ,
17                     "with root at",  $r[k].root$ 
18             else
19                 if  $r[k].power > 1$  then
20                     write "Repetition power",  $r[k].power$ , "at",  $r[k].start$ ,
21                     "with root at",  $r[k].root$ 
22              $r[k].power \leftarrow 0$ 

```

**Fig. 2.** The  $(\delta, \gamma)$ -REPETITIONS algorithm.

## 4.2 Running time

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

## 5 Computing the Longest $\delta$ -Approximate and $(\delta, \gamma)$ -Approximate Repeats

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

1.  $u_i \stackrel{\delta}{=} u_{i+1}$  for all  $i \in \{1, \dots, \ell - 1\}$
2.  $t = ru_1u_2 \dots u_\ell s$ , for some strings  $r, s$ .
3. Maximizes  $\ell$

Note that in case of  $\delta$ -approximate repetitions every repetition is  $\delta$ -approximate to the root. But in the case of repeats, each repeat is guaranteed to be  $\delta$ -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  $L_q$ ,  $q = 1, 2, \dots, m$  such that

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

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  $\delta$ -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  $(CCA.DCA.DCB.EDC)$ .

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

1.  $u_i \stackrel{\delta, \gamma}{=} u_{i+1}$  for all  $i \in \{1, \dots, \ell - 1\}$
2.  $t = ru_1u_2 \dots u_\ell s$ , for some strings  $r, s$ .
3. Maximizes  $\ell$

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

		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	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  $\delta$ -matrix  $D$  for  $t = DCCADCADCBEDCAA$ ,  $\delta=2$  and  $m=3$ .

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

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

*Example.* Table 6 shows the  $\gamma$ -matrix for the above example. Let  $\gamma = 2$  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  $\delta$ , find whether there exists a sequence of substrings  $u_1, u_2, \dots, u_p$  of  $t$  that satisfy the conditions:

1.  $u_i \stackrel{\delta, \gamma}{=} u_{i+1}$  for all  $i \in \{1, \dots, p-1\}$
2.  $t = ru_1u_2 \dots u_\ell s$ , for some strings  $r, s$ .
3. Minimizes

$$\sum_{i=1}^{p-1} \gamma(u_i, u_{i+1})$$

The computation of the MTP can be easily done using 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

		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  $\gamma$ -matrix  $G$  for  $t = DCCADCADCBEDCAA$ ,  $\delta=2$  and  $m=3$ .

the matrices  $D$  and  $G$ . Hence, the overall complexity for the both problems will be  $\mathcal{O}(n^2)$ .

## 6 Conclusion and Open problems

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

An interesting open problem is to compute the  $\delta$ -approximate and the  $(\delta, \gamma)$ -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 occurrences of a template with applications to protein structure, In A. Apostolico, M. Crochemore, Z. Galil and U. Manber, editors, *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science, vol. 644, pages 111–120, Tucson, AZ, 1992. Springer-Verlag, Berlin.
5. S. Karlin, M. Morris, G. Ghandour, and M.-Y. Leung, Efficient 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 E. Miranda, editor, *Readings in Music and Artificial Intelligence*. 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, *SIAM Journal on Computing* 27, 4 (1998), 972-992.
11. S. S. Skiena and G. Sundaram, Reconstructing strings from substrings, *J. Computational Biol.* 2 (1995) 333–353.