ON TUNING THE (α, δ)-SEQUENTIAL-SAMPLING ALGORITHM FOR δ-APPROXIMATE MATCHING WITH α-BOUNDED GAPS IN MUSICAL SEQUENCE

Domenico Cantone, Salvatore Cristofaro, Simone Faro
Università di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125 Catania, Italy
cantone, cristofaro, faro@dmi.unict.it

ABSTRACT

In this paper we present a new efficient algorithm for the δ-approximate matching problem with α-bounded gaps which arises in many questions concerning musical information retrieval and musical analysis. Our presented algorithm is an efficient variant of the (δ, α)-Sequential-Sampling algorithm (Cantone et al., 2003), recently introduced by the authors.

An extensive comparison with the other solutions existing in literature for the same problem indicates that our algorithm is more efficient, especially in the cases of long patterns. In particular the algorithm solves the problem in $O(mn)$-time and $O(m)$-space, where m is the length of the pattern and n is the length of the text. However it requires only $O(n)$-time on the average for alphabet with a uniform distribution. In addition, our algorithm computes the total number of approximate matchings for each position of the text, requiring only $\mathcal{O}(m\alpha)$-space.

Keywords: approximate string matching, experimental algorithms, musical information retrieval.

1 INTRODUCTION

Given a text T and a pattern P over some alphabet Σ, the *string matching problem* consists in finding *all* occurrences of P in T. It is a very extensively studied problem in computer science, mainly due to its direct applications to such diverse areas as text, image and signal processing, speech analysis and recognition, information retrieval, computational biology and chemistry, etc.

Recently, the classical string matching problem has been generalized with various notions of approximate matching, particularly useful in specific fields such as molecular biology (Karlin et al., 1988), musical applications (Crawford et al., 1998), or image processing (Karhumäki et al., 2000).

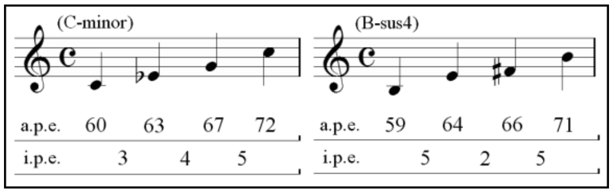

Figure 1: Representation of the C-minor and B-sus4 chords in the absolute pitch encoding (a.p.e.) and in the interval pitch encoding (i.p.e.).

In this paper we focus on a variant of the approximate string matching problem, namely the δ-*approximate string matching problem with* α-*bounded gaps*. Such a problem, which will be given a precise definition later, arises in many questions concerning musical information retrieval and musical analysis. This is especially true in the context of monophonic music, in which one wants to retrieve a given melody from a complex musical score. We mention here that a significant amount of research has been devoted to adapt solutions for exact string matching to δ-approximate matching (see for instance Cambouropoulos et al. (1999), Crochemore et al. (2001), Crochemore et al. (2002b), Cantone et al. (2004)). In this respect, Boyer-Moore-type algorithms are of particular interest, since they are very fast in practice.

The paper is organized as follows. In Section 2 we discuss the applications of approximate matching in the context of musical sequences. Then in Section 3 we introduce some basic notions and give a formal definition of the δ-approximate matching problem with α-bounded gaps. An algorithm based on the dynamic programming approach for the approximate matching problem of our interest is reviewed in Section 4.1. Then, in Section 4.3, we present a new efficient algorithm for the same problem. Experimental data obtained by running under various conditions both our algorithm and the one based on the dynamic programming approach are presented and compared in Section 6. Finally, we draw our conclusions in Section 7.

2 APPROXIMATE MATCHING AND MUSICAL SEQUENCES

Musical sequences can be schematically viewed as sequences of integer numbers, representing either the notes

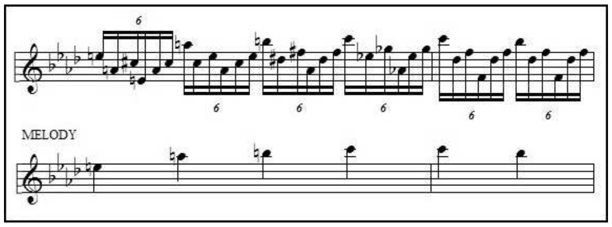

Figure 2: Two bars of the study Op. 25 N. 1 of F. Chopin (first score). The second score represents the melody. If a gap bound of $\alpha \geq 5$ is allowed, an exact occurrence of the melody can be found through the piece.

in the chromatic or diatonic notation (absolute pitch encoding), or the intervals, in number of semitones, between consecutive notes (interval pitch encoding); see the examples in Fig. 1. The second representation is generally of greater interest for applications in tonal music, since absolute pitch encoding disregards tonal qualities of pitches. Note durations and note accents can also be encoded in numeric form, giving rise to richer alphabets whose symbols can really be regarded as sets of parameters. This is the reason why alphabets used for music representation are generally quite large.

δ-approximate string matching algorithms are very effective to search for all similar but not necessarily identical occurrences of given melodies in musical scores. We recall that in the δ-approximate matching problem two integer strings of the same length match if the corresponding integers differ by at most a fixed bound δ. For instance, the chords C-minor and B-sus4 match if a tolerance of $\delta = 1$ is allowed in the absolute pitch encoding (where C-minor$= (60, 63, 67, 72)$ and B-sus4$= (59, 64, 66, 71)$), whereas if we use the interval pitch encoding, a tolerance of $\delta = 2$ is required to get a match (in this case we have C-minor$= (3, 4, 5)$ and B-sus4$= (5, 2, 5)$); see Fig. 1. Notice that for $\delta = 0$, the δ-approximate string matching problem reduces to the exact string matching problem.

Intuitively, we say that a melody (or *pattern*) has a δ-approximate occurrence with α-bounded gaps within a given musical score (or *text*), if the melody has a δ-approximate matching with a subsequence of the musical score, in which it is allowed to skip up to a fixed number α of symbols (the *gap*) between any two consecutive positions. In the present context, two symbols have an approximate matching if the absolute value of their difference is bounded by a fixed number δ.

In classical music compositions, and in particular in compositions for *Piano Solo*, it is quite common to find musical pieces based on a sweet ground melody, whose notes are interspaced by rapidly executed arpeggios. Fig. 2 shows two bars of the study *Op. 25 N. 1 for Piano Solo* by F. Chopin illustrating such a point. The notes of the melody are the first of each group of six notes (sextuplet). If we use the standard MIDI representation of the pitches, then the melody corresponds to the sequence of integer numbers $P = [76, 81, 83, 84, 84, 83, 86, 77]$. Then, if a gap bound of $\alpha \geq 5$ is allowed, an exact occurrence of the melody can be found through the piece.

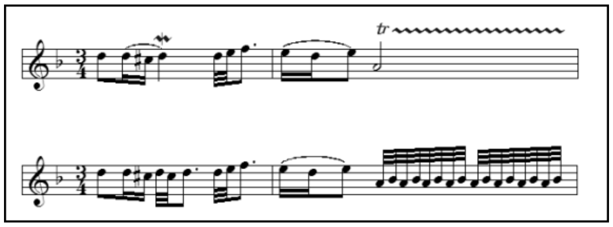

Figure 3: An excerpt of a piece of J.S. Bach (first score). The second score shows how the musical ornaments must be played. Two musical ornaments are present: a *mordent*, attached to the 4^{th} note; a *trill*, attached to the 11^{th} note.

The above musical technicality is not by any means the only one for which approximate string matching with bounded gaps turns out to be very useful. Other examples are given by *musical ornaments*, which are common practice in classical music, and especially in the music of the baroque period. Musical ornaments are groups of notes, played at a very fast tempo, which generally are “attached” to the notes of a given melody, in order to emphasize or adorn certain dynamical passages. Some of the most common musical ornaments are the *acciaccatura*, the *appoggiatura*, the *mordent*, the *turn*, and the *trill*.

Fig. 3 shows an excerpt of a *Minuet* by J.S. Bach, which makes use of musical ornaments. We provide both the actual score, in which ornaments are represented as special symbols marked above notes or as groups of small notes, and the corresponding score showing how these notations translate into real musical execution. Note that in Fig. 3 the mordent corresponds to a group of three notes, whereas the trill corresponds to a group of 16 notes. In general, to take care of musical ornaments in δ-matching problem with gaps, one needs gap values in the range between 4 and 16.

3 BASIC DEFINITIONS AND PROPERTIES

Before entering into details, we need a bit of notations and terminology. A string P is represented as a finite array $P[0 \,..\, m-1]$, with $m \geq 0$. In such a case we say that P has length m and write $\mathrm{length}(P) = m$. In particular, for $m = 0$ we obtain the empty string. By $P[i]$ we denote the $(i+1)$-st character of P, for $0 \leq i < \mathrm{length}(P)$. Likewise, by $P[i \,..\, j]$ we denote the substring of P contained between the $(i+1)$-st and the $(j+1)$-st characters of P, for $0 \leq i \leq j < \mathrm{length}(P)$. The substrings of the form $P[0 \,..\, j]$ (also denoted by P_j), with $0 \leq j < \mathrm{length}(P)$, are the nonempty *prefixes* of P.

Let Σ be an alphabet of integer numbers and let $\delta \geq 0$ be an integer. Two symbols a and b of Σ are said to be *δ-approximate* (or we say that a and b *δ-match*), in which case we write $a =_\delta b$, if $|a - b| \leq \delta$. Two strings P and Q over the alphabet Σ are said to be δ-approximate (or we say that P and Q δ-match), in which case we write $P \overset{\delta}{=} Q$, if

$$\mathrm{length}(P) = \mathrm{length}(Q), \quad \text{and}$$

$$P[i] =_\delta Q[i], \ \text{ for } i = 0, ..., \mathrm{length}(P) - 1\,.$$

Given a text T of length n and a pattern P of length m, a *δ-occurrence with α-bounded gaps of P in T at position i* is an increasing sequence of indices $(i_0, i_1, \ldots, i_{m-1})$ such that (i) $0 \leq i_0$ and $i_{m-1} = i \leq n-1$, (ii) $i_{h+1} - i_h \leq \alpha + 1$, for $h = 0, 1, \ldots m-2$, and (iii) $P[j] =_\delta T[i_j]$, for $j = 0, 1, \ldots m-1$. We write $P \trianglelefteq_{\delta,\alpha} T_i$ to mean that P has a δ-occurrence with α-bounded gaps in T at position i (in fact, when the bounds δ and α are well understood from the context, we will simply write $P \trianglelefteq T_i$).

The *δ-approximate string matching problem with α-bounded gaps* admits the following variants: (a) find all δ-occurrences with α-bounded gaps of P in T; (b) find all positions i in T such that $P \trianglelefteq T_i$; (c) for each position i in T, find the number of distinct δ-occurrences of P with α-bounded gaps at position i.

In Section 4.3 we will describe an efficient $\mathcal{O}(mn)$-time solution for the variants (b) and (c) above which uses only $\mathcal{O}(m\alpha)$ extra space. Variant (a) can then be solved by running an $\mathcal{O}(m^2\alpha)$-time and -space local search at each position i such that $P \trianglelefteq T_i$.

The following very elementary fact will be used later.

Lemma 1 *Let T and P be a text of length n and a pattern of length m, respectively. Also, let $\delta, \alpha \geq 0$. Then, for each $0 \leq i < n$ and $0 \leq k < m$, we have that $P_k \trianglelefteq_{\delta,\alpha} T_i$ if and only if $P[k] =_\delta T[i]$ and either $k = 0$, or $P_{k-1} \trianglelefteq_{\delta,\alpha} T_{i-h}$, for some h such that $1 \leq h \leq \alpha + 1$.* ■

4 ALGORITHMS FOR δ-APPROXIMATE MATCHING WITH α-BOUNDED GAPS

In this section we survey the state of the art concerning the δ-approximate matching problem with α-bounded gaps. In particular we present three algorithms based on three different strategies. Given a pattern P of length m and a text T of length n, the first algorithm, based on dynamic-programming (Crochemore et al., 2000, 2002a), solves variant (a) and (c) in $\mathcal{O}(mn)$-space, and variant (b) in $\mathcal{O}(n)$-space. In both case it requires $O(nm)$-time. The second algorithm is based on bit-parallelism (Baeza-Yates and Gonnet, 1992) and solves only variant (b) in $\mathcal{O}(\lceil mn/w \rceil)$-time requiring $\mathcal{O}(\lceil m\alpha/w \rceil)$-space, where w is the number of bits in the computer word. The third algorithm, named (δ, α)-Sequential-Sampling (Cantone et al., 2003), sequentially computes occurrences of prefixes of P solving variant (b) and (c) in $O(mn)$-time and $O(m\alpha)$-space, and variant (a) in $O(m^2\alpha)$-space.

4.1 An algorithm based on dynamic programming

The δ-approximate matching problem with α-bounded gaps has been first addressed by Crochemore *et al.* in Crochemore et al. (2000), where an algorithm based on the dynamic programming approach, named δ-Bounded-Gaps, has been proposed. In our review, we follow the presentation given later in Crochemore et al. (2002a), which considers also several new versions of the approximate matching problem with gaps.

Given as usual a text T of length n, a pattern P of length m, and two integers $\delta, \alpha \geq 0$, the algorithm δ-Bounded-Gaps runs in $\mathcal{O}(mn)$-time and -space, at least in the case in which one is interested in finding all δ-occurrences with α-bounded gaps of P in T (variant (a)). Space requirements can be reduced to $\mathcal{O}(n)$, if only positions i in T such that $P \trianglelefteq T_i$ need to be computed (variant (b)). To solve also variant (c) with δ-Bounded-Gaps, one needs to first solve variant (a) and then trace back and count all approximate matchings with gaps at each position of the text T.

```
δ-Bounded-Gaps (P, T, δ, α)
    $n = \text{length}(T)$
    $m = \text{length}(P)$
    for $j = 1$ to $n-1$ do
        $D[0,j] = -1$
        if ($P[0] =_\delta T[0]$) then $D[0,j] = j$
        else if $D[0,j-1] \geq j - \alpha$ then
            $D[0,j] = D[0,j-1]$
    for $i = 1$ to $m-2$ do
        $D[i,0] = -1$
        for $j = 1$ to $n-1$ do
            $D[i,j] = -1$
            if ($P[i] =_\delta T[j]$ and $D[i-1,j-1] \geq 0$)
                then $D[i,j] = j$
            else if $D[i,j-1] \geq j - \alpha$ then
                $D[i,j] = D[i,j-1]$
    for $j = m-1$ to $n-1$ do
        if $P[m-1] =_\delta T[j]$ and $D[m-2,j-1] \geq 0$
            then output($j$)
```

Figure 4: The algorithm δ-Bounded-Gaps for the δ-approximate matching problem with α-bounded gaps.

The algorithm δ-Bounded-Gaps is presented as an incremental procedure, based on the dynamic programming approach. More specifically, it starts by first computing all the δ-occurrences with α-bounded gaps in T of the prefix of P of length 1, i.e. P_0. Then, during the i-th iteration, it looks for all the δ-occurrences with α-bounded gaps in T of the prefix P_{i-1}. At the end of the last iteration, the δ-occurrences of the whole pattern P have been computed.

To give a more formal description of the algorithm, let us put *LastOccur*$_j(P_i) = \max\left(\{0 \leq k \leq j : P_i \trianglelefteq T_k \text{ and } j - k \leq \alpha\} \cup \{-1\}\right)$. Notice that if *LastOccur*$_j(P_i) = -1$, then $P_i \ntrianglelefteq T_k$ for $k = j - \alpha, j - \alpha + 1, \ldots, j$. Otherwise, *LastOccur*$_j(P_i)$ is the largest value $k \in \{j - \alpha, j - \alpha + 1, \ldots, j\}$ such that $P_i \trianglelefteq T_k$.

The values *LastOccur*$_j(P_i)$ can be computed incrementally, for $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$. More specifically, the algorithm δ-Bounded-Gaps fills a matrix D of dimension $m \times n$, where $D[i,j]$ corresponds to *LastOccur*$_j(P_i)$, according to the following recursive relation:

$$D[i,j] = \begin{cases} j & \text{if } T[j] =_\delta P[i] \text{ and} \\ & \quad \text{- } i = 0\,, \text{ or} \\ & \quad \text{- } i, j \geq 1 \text{ and } D[i-1,j-1] \geq 0 \\ & \\ D[i,j-1] & \text{if } j \geq 1\,,\ D[i,j-1] \geq j - \alpha\,, \text{ and} \\ & \quad \text{- } T[j] \neq_\delta P[i]\,, \text{ or} \\ & \quad \text{- } T[j] =_\delta P[i]\,,\ i \geq 1\,, \\ & \quad\ \ \text{and } D[i-1,j-1] < 0 \\ & \\ -1 & \text{otherwise} \end{cases}$$

where $0 \leq i < m$ and $0 \leq j < n$.

Using a *trace-back* procedure, as described in Crochemore et al. (2002a), the values $LastOccur_j(P_i)$ can be used to retrieve the approximate matchings at a given position in time $\mathcal{O}(m\alpha)$.

Fig. 4 presents the pseudo-code of the algorithm δ-Bounded-Gaps. Its running time is easily seen to be $\mathcal{O}(mn)$. Also, $\mathcal{O}(mn)$-space is needed to store the matrix D. However, if one is only interested in the positions i of T at which $P \trianglelefteq T_i$, space requirements reduce to $\mathcal{O}(n)$, since the computation of each row depends only on the values stored in the previous row.

4.2 An algorithm based on bit-parallelism

In this section we present a simple algorithm to search gapped occurrence of a pattern in a text which makes use of *bit-parallelism* (Baeza-Yates and Gonnet, 1992). This technique consists in taking advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor of at most w, where w is the number of bits in the computer word. However algorithm based on bit-parallelism generally work well only on patterns of moderate length and are not able to retrieve information about matches.

The string-matching problem can be solved in $O(n)$-time by simulating the behavior of an automata (Cormen et al., 1990) which recognizes the pattern P. The Shift-And algorithm (Baeza-Yates and Gonnet, 1992) for exact string matching, uses bit-parallelism to simulate the automata in its nondeterministic form. This simulation is performed by representing the automata as a list of L bits, where L is the number of states of the automata, and each state corresponds to a bit in the list. In this context, bits corresponding to active states are set to 1, while bits corresponding to inactive states are set to 0.The initial state is not represented because it is always active.

Note that if $L \leq w$ the entire list fits in a single computer word, whereas if $L > w$ we need $\lceil L/w \rceil$ computer words to represent the automata.

Figure 4.2(A) shows the non-deterministic automata of the pattern $P = [2, 5, 3, 2]$ for the δ-matching problem.

For each character, c, of the alphabet Σ the algorithm maintains a bit mask $B[c]$ where the i-th bit is set to 1 if $P[i] = c$. To take into account δ-matches, we set to 1 the i-th bit of the masks from $B[c - \delta]$ to $B[c + \delta]$.

The current configuration of the automa is maintained in a bit mask D, which is initialized to O^L. The algorithm scans the text from the first character to the last one and, for each position j, it performs the following basic shift-and operation:

$$D = ((D << 1) \mid 0^{L-1}1) \ \& \ B[T[j]]$$

If the final state is active we report a match at position j. The Shift-And algorithm achieves $O(\lceil mn/w \rceil)$ worst-case time and require $O(\lceil L/w \rceil)$ extra-space.

If we want extend the Shift-And algorithm to α-bounded gaps we need to modify the automata to allow the presence of gaps between any two consecutive positions. Thus, between each character of the pattern, we

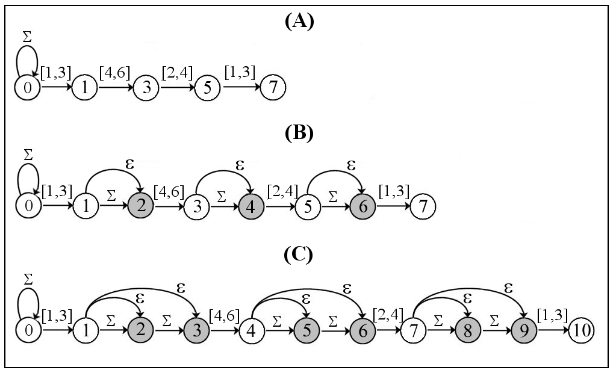

Figure 5: Three non-deterministic automata of the numeric pattern $P = 2, 5, 3, 2$ for approximate matching with $\delta = 1$. **(A)** non-deterministic automata with $\alpha = 0$ **(B)** non-deterministic automata with $\alpha = 1$ **(C)** non-deterministic automata with $\alpha = 2$

insert α transitions that can be followed by any character of the alphabet. Then α ε-transitions leave the state where a character has been recognized and skip from one to α subsequent edges, respectively. A self-loop in the initial state allows the match to begin at any text position. Whit this representation the number, L, of states needed to simulate the automa is equal to $m + (m - 1)\alpha$. Figures 4.2(B),(C) show the non-deterministic automata for the pattern $P = [2, 5, 3, 2]$, with $\alpha = 1$ and $\alpha = 2$.

In this context we call *gap-initial states* those states S_i from where an ε-transition leaves. For each gap-initial state S_i we define its corresponding *gap-final state* to be $S_{i+\alpha}$, i.e., the last state reached by an ε-transition leaving S_i. Then we create a bit mask I which has 1 in the gap-initial states, and another mask F that has 1 in the gap-final states. After performing the normal shift-and step, we simulate all the ε-moves with the operation

$$D = D \mid (((F \ - \ (D \ \& \ I)) \ \& \ \sim F) \ << \ 1)$$

Figure 6 shows the complete algorithm. The preprocessing takes $O(m\alpha|\Sigma|)$ time, while the scanning needs $O(\lceil nm/w \rceil)$ time.

4.3 The (δ, α)-Sequential-Sampling algorithm

The (δ, α)-Sequential-Sampling algorithm (Cantone et al., 2003) is characterized by an $\mathcal{O}(mn)$-time and an $\mathcal{O}(m\alpha)$-space complexity. In addition, this algorithm solves variant (c) (and therefore also variant (b)) of the approximate matching problem with gaps, as stated in Section 3. If one is also interested in retrieving the actual approximate matching occurrences at position i of a text T, a possibility would be to compute the submatrix $D[k, j]$, for $\max(0, (m-1)\cdot(\alpha+1)) \leq k \leq i$ and $0 \leq j \leq m-1$, where, as before, m is the length of the pattern, and then trace back through all possible approximate matchings. The submatrix $D[k, j]$ can be computed in time and space $\mathcal{O}(m^2\alpha)$ by the algorithm δ-Bounded-Gaps.

The algorithm computes the occurrences of all prefixes of the pattern in continuously increasing prefixes of the text. That is, for a text T of length n, pattern P of

```
($\alpha$, $\delta$)-Shift-And ($T$, $P$, $\delta$, $\alpha$)
1.  $n = \text{length}(T)$
2.  $m = \text{length}(P)$
3.  $L = m + (m-1) \times \alpha$
4.  for $c \in \Sigma$ do $B[c] = 0^L$
5.  $I = 0^L$
6.  $i = 0$
7.  for $j = 0$ to $m-1$ do
8.      for $c \in \{P[j]-\delta..P[j]+\delta\}$ do
9.          $B[c] = (B[c] \mid (1 << i))$
10.         $i = i + 1$
11.     if $j < m-1$ then
12.         $I = I|(1 << (i-1))$
13.         for $c \in \Sigma$ do
14.             for $k = i$ to $i + \alpha - 1$ do
15.                 $B[c] = (B[c] \mid (1 << k))$
16.         $i = i + \alpha$
17. $M = 1 << (L-1)$
18. $D = 0^L$
19. for $j = 0$ to $n-1$ do
20.     if $D$ & $M \neq 0^L$ then print($j$)
21.     $D = ((D << 1) \mid 0^{L-1}1)$ & $B[T[j]]$
22.     $D = D \mid (((F - (D \,\&\, I)) \,\&\, \sim F) << 1)$
```

Figure 6: The algorithm based on Bit-Parallelism for the δ-approximate matching problem with α-bounded gaps.

```
($\delta, \alpha$)-Sequential-Sampling ($T$, $P$, $\delta$, $\alpha$)
1.  $n = \text{length}(T)$
2.  $m = \text{length}(P)$
3.  for $i = 0$ to $\alpha + 1$ do
4.      for $j = 0$ to $m - 2$ do
5.          $\mathcal{M}[i, j] = 0$
6.  for $i = 0$ to $m - 2$ do $\mathcal{C}[i] = 0$
7.  for $i = 0$ to $n - 1$ do
8.      $j = i \; mod(\alpha + 2)$
9.      for $k = 0$ to $m - 2$ do
10.         $\mathcal{C}[k] = \mathcal{C}[k] - \mathcal{M}[j, k]$
11.         $\mathcal{M}[j, k] = 0;$
12.     if $P[m-1] =_\delta T[i]$ and $\mathcal{C}[m-2] > 0$ then
13.         output($i$)
14.     for $k = m - 2$ downto $1$ do
15.         if $P[k] =_\delta T[i]$ and $\mathcal{C}[k-1] > 0$ then
16.             $\mathcal{M}[j, k] = \mathcal{C}[k-1]$
17.             $\mathcal{C}[k] = \mathcal{C}[k] + \mathcal{C}[k-1]$
18.     if $P[0] =_\delta T[i]$ then
19.         $\mathcal{M}[j, 0] = 1$
20.         $\mathcal{C}[0] = \mathcal{C}[0] + 1$
```

Figure 7: The (δ, α)-Sequential-Sampling algorithm for the δ-approximate matching problem with α-bounded gaps.

length m, and nonnegative integers δ, α, during its first iteration the algorithm (δ, α)-Sequential-Sampling computes the (number of) occurrences of all prefixes P_k of P such that $P_k \trianglelefteq T_0$. Then, during the i-th iteration, it computes (the number of) all occurrences of prefixes P_k of P such that $P_k \trianglelefteq T_i$, using information gathered during previous iterations.

To be more precise, let $\mathcal{S}_i$ denote the collection of all pairs (j, k) such that $P_k \trianglelefteq T_j$, for $0 \leq i \leq n$, $0 \leq j < i$, and $0 \leq k < m$. Notice that $\mathcal{S}_0 = \varnothing$. If we put $\mathcal{S} = \mathcal{S}_n$, then the problem of finding the positions i in T such that $P \trianglelefteq T_i$ translates to the problem of finding all values i such that $(i, m-1) \in \mathcal{S}$.

To begin with, notice that Lemma 1 justifies the following recursive definition of the set $\mathcal{S}_{i+1}$ in terms of $\mathcal{S}_i$, for $i < n$:

$$\mathcal{S}_{i+1} = \mathcal{S}_i \cup \{(i,k) : \begin{array}{l} P[k] =_\delta T[i] \text{ and} \\ (k = 0 \text{ or } (i-h, k-1) \in \mathcal{S}_i, \\ \text{ for some } h \in \{1, \ldots, \alpha+1\})\}. \end{array}$$

Such relation, coupled with the initial condition $\mathcal{S}_0 = \varnothing$, allows one to compute the set $\mathcal{S}$ in an iterative fashion.

From a practical point of view, the set $\mathcal{S}$ is represented by its characteristic $n \times m$ matrix $\mathcal{M}$, where $\mathcal{M}[i, k]$ is 1 or 0, provided that the pair (i, k) belongs or does not belong to $\mathcal{S}$, for $0 \leq i < n$ and $0 \leq k < m$.

Moreover, since during the i-th iteration at most $\alpha + 1$ rows of $\mathcal{M}$ need to be scanned —more precisely the ones having index $j \in \{\max(0, i - \alpha - 1), i - 1\}$,— it would be enough to store only $\alpha + 1$ rows of $\mathcal{M}$ at each step of the computation, plus another one as working area.

In addition, by maintaining an extra array $\mathcal{C}$ of length m such that the following invariant holds:

$$\mathcal{C}[k] = \sum_{j=\max(0,i-\alpha-1)}^{i-1} \mathcal{M}[j,k]\,, \qquad \text{for } 0 \leq k < m\,,$$

the test of the conditional instruction at line 6 can be performed in constant time, rather than in $\mathcal{O}(\alpha)$-time.

Such observations allow to reduce the space requirement to $\mathcal{O}(m\alpha)$ and the running time to $\mathcal{O}(mn)$.

Finally, rather than maintaining in $\mathcal{M}[j, k]$ the Boolean value of the test $(j, k) \in \mathcal{S}$, it is more convenient to let $\mathcal{M}[j, k]$ count the number of *distinct* δ-occurrences with α-gaps of P_k at position j of T. With this change, when the i-th iteration starts, the item $\mathcal{C}[k]$ will contain the total number of *distinct* δ-occurrences with α-gaps of P_k at positions $\max(0, i - \alpha - 1)$ through $i - 1$, provided that the above invariant holds. Such values can then be used to maintain the invariant itself.

Plainly, at the end of the computation one can retrieve in constant time the number of approximate matchings at each position of the text.

The resulting algorithm is presented in detail in Fig. 7.

5 AN IMPROVED VERSION OF (δ,α)-SEQUENTIAL-SAMPLING

Here we present a new efficient variant of the (δ, α)-Sequential-Sampling algorithm for δ-approximate matching problem with α-bounded gaps, name (δ, α)-Tuned-Sequential-Sampling algorithm (TSS for short). As its progenitor the TSS algorithm solves variant (c) of the problem in $O(mn)$-time and $O(m\alpha)$-space but, practically, it requires only $O(n)$-time on the average for alphabet with a uniform distribution. In addition, the TSS algorithm solves variant (b) of the problem requiring only $\mathcal{O}(m)$-space.

5.1 The average number of matched prefixes

In the (δ, α)-Sequential-Sampling algorithm (Figure 7), for each position i in the text, the **for**-loop of line 14 iterates on k from $m-2$ down to 1. However this operation has effect only for those values of k such that $C[k-1] > 0$, i.e. P_{k-1} occurs at a position h of the text, with $i-\alpha-1 \leq h \leq i-1$.

In this section we compute the expected number of matched prefixes P_k of the pattern for each position of the text, and show that such number can be considered a constant if the alphabet has a uniform distribution of characters. Thus the time complexity of the (δ, α)-Sequential-Sampling algorithm can be reduced to $O(n)$ on average. Then in the following section we present a variant of the (δ, α)-Sequential-Sampling algorithm that, basing on such observation, achieves better performances.

In our analysis, we suppose a uniform distribution of characters of the alphabet Σ of dimension σ. In addition we suppose the independence of characters of text and pattern. In this context the probability, Δ, that two characters of Σ effect a δ-match is given, with a good approximation, by

$$\Delta = \frac{2\delta+1}{\sigma}$$

Given a prefix of the pattern of length k, with $k \leq m$, we now attempt to calculate the probability that $P_{k-1} \trianglelefteq T_i$. Observe that the probability that $P[k-1] =_\delta T[i]$ is equal to Δ. Suppose now that $P[k-j..k-1] \trianglelefteq T_i$, then the probability that $P[k-j-1..k-1] \trianglelefteq T_i$ is the probability that character $P[k-j-1]$ has a gapped match after $P[k-j]$. This is given by

$$\begin{array}{lll}
\Delta & + & \textit{(match at 1-th character)} \\
(1-\Delta)\Delta & + & \textit{(match at 2-th character)} \\
(1-\Delta)^2\Delta & + & \textit{(match at 3-th character)} \\
.. & + & \\
(1-\Delta)^\alpha\Delta & = & \textit{(match at } (\alpha+1)\textit{-th char)} \\
\Delta\sum_{k=0}^{\alpha}(1-\Delta)^k & = & \\
\Delta\frac{(1-\Delta)^{\alpha+1}-1}{(1-\Delta)-1} & = & 1-(1-\Delta)^{\alpha+1}
\end{array}$$

Thus the probability that $P_k \trianglelefteq T_i$ is given by

$$\begin{aligned}
\Pr\{P_k \trianglelefteq T_i\} &= \Delta \prod_{j=1}^{k-1}\left(1-(1-\Delta)^{\alpha+1}\right) \\
&= \Delta\left(1-(1-\Delta)^{\alpha+1}\right)^{k-1}
\end{aligned}$$

And the expected number, φ, of prefixes P_k such that $P_k \trianglelefteq T_i$, for each position i of the text is given by

$$\begin{aligned}
\varphi &= \Delta \sum_{k=0}^{m-1}\left(1-(1-\Delta)^{\alpha+1}\right)^k \\
&= \Delta\frac{1-\left(1-(1-\Delta)^{\alpha+1}\right)^m}{(1-\Delta)^{\alpha+1}}
\end{aligned}$$

For $m \to \infty$ we have that

$$\varphi \to \frac{\Delta}{(1-\Delta)^{\alpha+1}}$$

In Figure 5.1 is shown a graphical representation of the function φ with different values of α and δ. It is possible to observe that for values of σ greater than 10, the correspondent values of φ are less than 1.

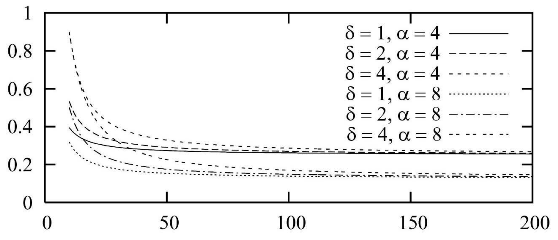

Figure 8: The expected number, φ, of matched prefixes of the pattern for each position of the text and for different values of α and δ.

5.2 A new efficient algorithm

The extra work done by the (δ, α)-Sequential-Sampling algorithm in the **for**-loop of line 14, can be avoided by maintaining an ordered list L, such that during each iteration of the **for**-loop of line 7, L contains exactly those values of k, with $0 \leq k < m-1$, such that $C[k] > 0$. Then, when the algorithm try to match the prefix P_{k+1} at position i of the text T, where k is in the list L, it needs only to know if $P[k+1] =_\delta T[i]$ (in fact, since $\mathcal{C}[k] > 0$, we know that the prefix P_k match in some position $i-h$ of the text, where $1 \leq h \leq \alpha+1$).

The TSS algorithm acts as follows. For each position i of the text T, it scans the list L, in decreasing order, and for each value of k in L it performs the following operations.
After updating the entries $\mathcal{C}[k]$ and $\mathcal{M}[j][k]$, as the (δ, α)-Sequential-Sampling algorithm does in lines 10 and 11, TSS checks if $\mathcal{C}[k] = 0$, and in such a case, the current value of k will be removed from the list L, and the algorithm will proceeds to the next value of k contained in L. If, on the contrary, we have $\mathcal{C}[k] > 0$, then the algorithm try to match the prefix P_{k+1} at position i in the text T, by looking only if $P[k+1] =_\delta T[i]$. If so then P_{k+1} will match and, if $k+1 = m-1$, the algorithm will report a match at position i. If $k+1 < m-1$, then, after updating the entries $\mathcal{M}[j][k+1]$ and $\mathcal{C}[k+1]$, with $\mathcal{M}[j][k+1]$ now containing the correct number of matches of P_{k+1} at position i in T, the value $k+1$ will be inserted at the appropriate position in the list L (if it is not already there), just before k. Then the algorithm proceeds with the next value of k contained in L, and iterates until the list L has been completely scanned. As last operation, the algorithm checks if the characters $P[0]$ and $T[i]$ effect a δ-match, and in such a case it inserts the value 0 in the list L.

Fig. 9 shows the complete code of the TSS algorithm, where the list L is implemented, in a circular fashion, by using an array, $next$, of size m. The entry $next[m-1]$ will be used as an extra sentinel which will always point to the first (highest) value of k contained in the list L. In addition, in order to manage efficiently insertions and deletions over L, we maintain also a pointer to the predecessor of the current value of k in L, by using an extra variable, p, which is moved across L during the scanning phase. The

```
$(\delta, \alpha)$-Tuned-Sequential-Sampling $(T, P, \delta, \alpha)$
    $n = $ length$(T)$
    $m = $ length$(P)$
    for $i = 0$ to $\alpha + 1$ do
        for $j = 0$ to $m - 2$ do
            $\mathcal{M}[i, j] = 0$
    for $i = 0$ to $m - 2$ do $\mathcal{C}[i] = 0$
    $next[0] = next[m - 1] = m - 1$
    for $i = 0$ to $n - 1$ do
        $j = i\ mod(\alpha + 2)$
        $p = m - 1$
        $k = next[m - 1]$
        while $k < m - 1$ to
            $\mathcal{C}[k] = \mathcal{C}[k] - \mathcal{M}[j, k]$
            $\mathcal{M}[j, k] = 0$
            if $(\mathcal{C}[k] == 0)$ then
                $next[p] = next[k]$
            else
                if $(P[k + 1] =_\delta T[i])$ then
                    if $(k == m - 2)$ then
                        output$(i)$
                    else
                        $\mathcal{M}[j][k + 1] = \mathcal{C}[k]$
                        $\mathcal{C}[k + 1] = \mathcal{C}[k + 1] + \mathcal{C}[k]$
                        if $p > k + 1$ then
                            $next[p] = k + 1$
                            $next[k + 1] = k$
                $p = k$
            $k = next[k]$
        if $(P[0] =_\delta T[i])$ then
            $\mathcal{M}[j][0] = 1$
            $\mathcal{C}[0] = \mathcal{C}[0] + 1$
            if $p > 0$ then
                $next[p] = 0$
```

Figure 9: The (δ, α)-Tuned-Sequential-Sampling algorithm for the δ-approximate matching problem with α-bounded gaps.

TSS algorithm achieves $O(mn)$-time and $O(m\alpha)$-space. However, since the number of matched prefixes of the pattern can be considered fixed for each position of the text (see Section 5.1), we aspect a linear average case complexity in practical cases.

Note that, if we are interested only in variant (b) of the approximate matching problem with bounded gaps, we can still improve the efficiency of the TSS algorithm, reducing also the space required to $O(m)$, as follows. Suppose that $\alpha + 2 \leq w$ where w is the length of a computer word. This assumption is realistic since the length of the gap is usually not greater than 16 in practical cases. Then we can represent the columns of the table $\mathcal{M}$, each with a computer word w stored in the table $\mathcal{C}$, avoiding the use of the table $\mathcal{M}$ itself. That is, for $k = 0, 1, \ldots, m - 2$, the $(k + 1)$-st column of $\mathcal{M}$ will correspond to the computer word stored in $\mathcal{C}[k]$. However, in this representation we can not allow the entries $\mathcal{M}[k][j]$ of the table $\mathcal{M}$, where $j = i\ mod(\alpha + 2)$, to contain the actual number of distinct matches of P_k at position i, but only the boolean value indicating that the prefix P_k of P as an occurrence ending at position i of the text T. Moreover, the test if the $(k + 1)$-st column of $\mathcal{M}$ is not null, i.e, the prefix P_k occurs at some positions between $i - \alpha - 1$ and $i - 1$ in T, reduces to the test if $\mathcal{C}[k] \neq 0$.

6 EXPERIMENTAL RESULTS

In this section we report experimental data relative to an extensive comparison of the running times of our algorithm (δ, α)-Tuned-Sequential-Sampling (TSS), against the algorithms δ-Bounded-Gaps (DP), (δ, α)-Shift-And (SA) and (δ, α)-Sequential-Sampling (SS).

All algorithms have been implemented in the C programming language and were used to search for the same patterns in large fixed text sequences on a PC with a Pentium IV processor at 2.66GHz. In particular, they have been tested on two Randσ problems, for $\sigma = 60, 120$ and on a real music text buffer.

In particular, each Randσ problem consisted in searching a set of 250 random patterns of length 10, 20, 40, 60, 80, 100, 120, and 140 in a 5Mb random text sequence over a common alphabet of size σ. For each Randσ problem, the approximation bound δ and the gap bound α have been set to 2 and to $4, 8$, respectively.

Concerning the tests on the real music text buffer, these have been performed on a 4.8Mb file obtained by combining a set of classical pieces, in MIDI format, by C. Debussy. The resulting text buffer has been translated in the pitch interval encoding with an alphabet of 101 symbols. For each $m = 10, 20, 40, 60, 80, 100, 120$, 140, we have randomly selected in the file 250 substrings of length m which subsequently have been searched for in the same file. All running times have been expressed in tenths of second.

Experimental results show that the TSS algorithm is faster than the existing ones and its superiority is more noticeable as the size of the pattern increases. Moreover it turns out from experimental results that the execution time does not depend on the length of the pattern.

7 CONCLUSIONS

We have presented a new efficient $\mathcal{O}(mn)$-time variant of the (δ, α)-Sequential-Sampling algorithm, named (δ, α)-Tuned-Sequential-Sampling, for the δ-approximate string matching problem with α-bounded gaps, with a linear average case time complexity. The algorithm has been compared against various existing solutions for the problem. Our experimentation has shown that our algorithm is faster, especially in the case of large alphabet. The performances of our algorithm become more remarkable as the size of the pattern increases. In addition, our algorithm uses only $\mathcal{O}(m\alpha)$-space for computing the number of all distinct approximate matchings of the pattern at each position of the text, while uses only $\mathcal{O}(m)$-space to find all approximate matchings of the pattern in the text.

References

R. A. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

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.

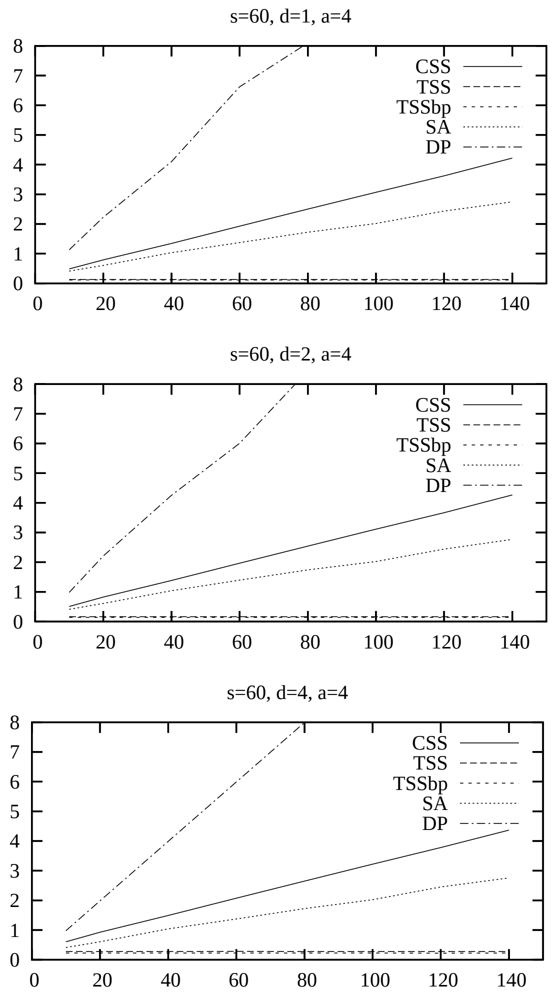

Figure 10: Experimental results with $\sigma = 60$ and $\alpha = 4$

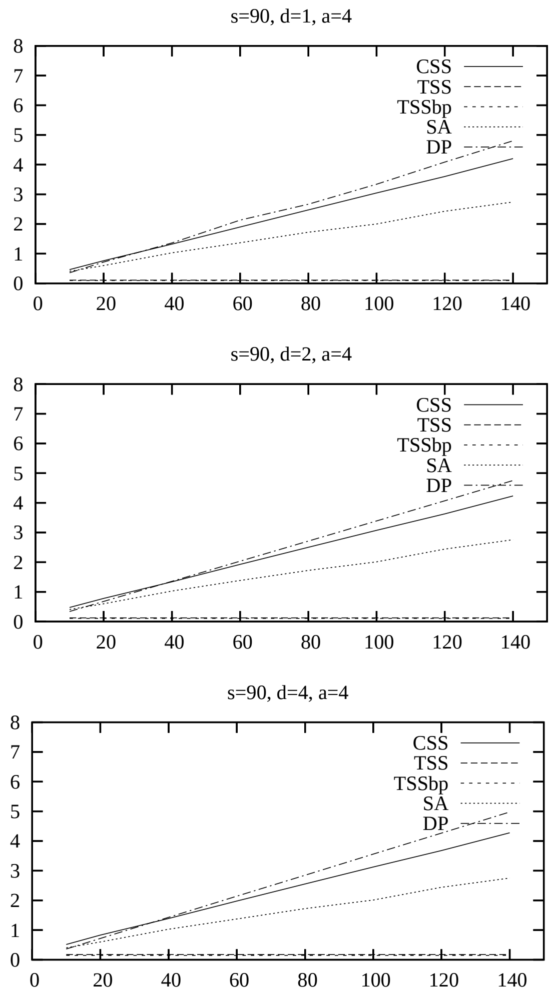

Figure 11: Experimental results with $\sigma = 90$ and $\alpha = 4$

D. Cantone, S. Cristofaro, and S. Faro. An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences. In J.D.P. Rolim (Eds.) M. Margraf, M. Mastrolli, editor, *LNCS 2647*, pages 47–58, 2003. Proc. of WEA 2005.

D. Cantone, S. Cristofaro, and S. Faro. Efficient algorithms for the δ-approximate string matching problem in musical sequences. pages 69–82, Czech Technical University, Prague, Czech Republic, 2004. Proc. of the Prague Stringology Conference '04.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

T. Crawford, C. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:71–100, 1998.

M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, and J. F. Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, pages 75–86, Hunter Valley, Australia, 2000.

M. Crochemore, C. S. Iliopoulos, T. Lecroq, and Y. J. Pinzon. Approximate string matching in musical sequences. In M. Balík and M. Šimánek, editors, *Proceedings of the Prague Stringology Conference'01*, pages 26–36, Prague, Czech Republic, 2001. Annual Report DC–2001–06.

M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis, and K. Tsichlas. Approximate string matching with gaps, 2002a.

M. Crochemore, C. S. Iliopoulos, T. Lecroq, W. Plandowski, and W. Rytter. Three heuristics for δ-matching: δ-BM algorithms. In A. Apostolico and M. Takeda, editors, *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching*, number 2373 in Lecture Notes in Computer Science, pages 178–189, Fukuoka, Japan, 2002b. Springer-Verlag, Berlin.

J. Karhumäki, W. Plandowski, and W. Rytter. Pattern-matching problems for two-dimensional images described by finite automata. *Nordic J. Comput.*, 7(1):1–13, 2000.

S. Karlin, M. Morris, G. Ghandour, and M. Y. Leung. Efficient algorithms for molecular sequence analysis. *Proceedings of the National Academy of Science*, 85:841–845, 1988.