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

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

We present a very efficient variant of the (δ, α)-SEQUENTIAL-SAMPLING algorithm, recently introduced by the authors, for the δ-approximate string matching problem with α-bounded gaps, which often arises in many questions on musical information retrieval and musical analysis.

Though it retains the same worst-case $\mathcal{O}(mn)$-time and $\mathcal{O}(m\alpha)$-space complexity of its progenitor to compute the number of distinct δ-approximate α-gapped occurrences of a pattern of length m at each position in a text of length n, our new variant achieves an average $\mathcal{O}(n)$-time complexity in practical cases.

Extensive experimentations indicate that our algorithm is more efficient than existing solutions for the same problem, especially in the case of long patterns.

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, musical analysis, information retrieval, computational biology and chemistry, etc.

In this paper we focus on a variant of the 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 on musical information retrieval and musical analysis.

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. A brief survey of existing algorithms for this problem is given in Section 4, whereas in Section 5 we present a new very efficient variant of the (δ, α)-SEQUENTIAL-SAMPLING algorithm recently introduced by the authors in Cantone et al. (2005). Experimental data obtained by running all the discussed algorithms under different conditions are presented and compared in Section 6. Finally, we draw our conclusions in Section 7.

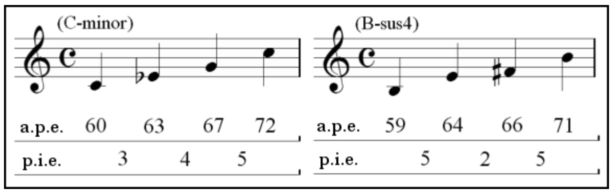

Figure 1: Representation of the *C-minor* and *B-sus4* chords in the absolute pitch encoding (a.p.e.) and in the pitch interval encoding (p.i.e.).

2 APPROXIMATE MATCHING AND MUSICAL SEQUENCES

Musical sequences can be schematically viewed as sequences of integer numbers, representing either the notes in the chromatic or diatonic notation (absolute pitch encoding), or the intervals, in number of semitones, between consecutive notes (pitch interval encoding); see examples in Figure 1.

δ-approximate string matching algorithms are very effective in searching 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.

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 approximate matchings. In the present context, two symbols have an approximate matching if the absolute value

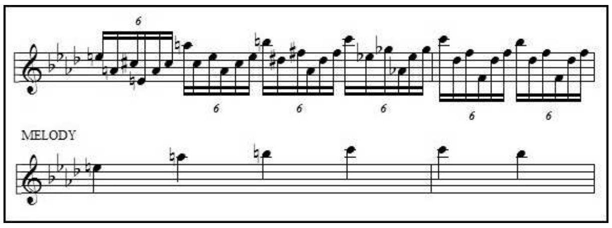

Figure 2: Two bars of the study Op. 25 N. 1 by F. Chopin (first score). The second score represents the melody. If a gap bound of $\alpha \geq 5$ is allowed, an exact occurrence of the melody can be found through the piece.

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. Figure 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).

Arpeggios are not by any means the only musical technicality 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.

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 $\text{length}(P) = m$. In particular, for $m = 0$ we obtain the empty string. By $P[i]$ we denote the $(i+1)$-st character of P, for $0 \leq i < \text{length}(P)$. 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 $0 \leq i \leq j < \text{length}(P)$. The substrings of the form $P[0 \,..\, j]$ (also denoted by P_j), with $0 \leq j < \text{length}(P)$, 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*, 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, in which case we write $P \overset{\delta}{=} Q$, if

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

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) $i_{h+1} - i_h - 1 \leq \alpha$, for $h = 0, 1, \ldots m-2$,[1] and (iii) $P[j] =_\delta T[i_j]$, for $j = 0, 1, \ldots m-1$. We write $P \trianglelefteq^{i}_{\delta,\alpha} T$ 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 $P \trianglelefteq^i T$).

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 $P \trianglelefteq^i T$; (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 $P \trianglelefteq^i T$.

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 $P_k \trianglelefteq^{i}_{\delta,\alpha} T$ if and only if $P[k] =_\delta T[i]$ and either $k = 0$, or $P_{k-1} \trianglelefteq^{i-h}_{\delta,\alpha} T$, for some h such that $1 \leq h \leq \alpha + 1$.*

■

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

In this section we survey the state-of-the-art of the δ-approximate matching problem with α-bounded gaps. In particular, we consider three algorithms, based on 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 variants (a) and (c) of the problem in $\mathcal{O}(mn)$-space, and variant (b) in $\mathcal{O}(n)$-space, requiring in both cases $\mathcal{O}(mn)$-time. The second algorithm, based on bit-parallelism (Baeza-Yates and Gonnet, 1992), solves only variant (b) of the problem in $\mathcal{O}(\lceil mn/\omega \rceil)$-time and $\mathcal{O}(\lceil m\alpha/\omega \rceil)$-space, where ω is the number of bits in the computer word. The third algorithm, named (δ,α)-SEQUENTIAL-SAMPLING (Cantone et al., 2005), computes sequentially occurrences of prefixes of P thus solving variants (b) and (c) of the problem in $\mathcal{O}(mn)$-time and $\mathcal{O}(m\alpha)$-space, and variant (a) in $\mathcal{O}(m^2\alpha)$-space.

4.1 An algorithm based on dynamic programming

The δ-approximate matching problem with α-bounded gaps has been first addressed 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 $P \trianglelefteq^i T$ need to be computed (variant

[1]The notation $x(i,j)$ is also used to denote a gap of at least i characters and at most j characters. Using such notation, in the problem at hand we are admitting gaps of the form $x(0,\alpha)$ at each position.

(b)). To solve also variant (c), one can first solve variant (a) and then trace back and count all approximate matchings with gaps at each position of the text T.

The δ-BOUNDED-GAPS algorithm is presented as an incremental procedure, based on the dynamic programming approach. More specifically, the δ-BOUNDED-GAPS algorithm computes a matrix D of dimension mxn where $D[i,j] = \max\left(\{0 \leq k \leq j : P_i \trianglelefteq^k T \text{ and } j-k \leq \alpha\} \cup \{-1\}\right)$.

Notice that $P_i \trianglelefteq^j T$ if and only if $P[i] =_\delta T[j]$ and $D[i-1, j-1] \geq 0$, for $i = 1, 2, \ldots, m-1$ and $j = 1, 2, \ldots, n-1$.

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

4.2 An algorithm based on bit-parallelism

Baeza-Yates and Gonnet (1992) presented an algorithm for the exact string matching problem, named SHIFT-AND, which uses bit-parallelism to simulate a nondeterministic finite automaton (NFA). The simulation is carried out by representing the automaton as an array of L bits, where $L+1$ is the number of states of the automaton. Bits corresponding to active states are set to 1, whereas bits corresponding to inactive states are set to 0. The initial state is not represented because it is always active.

Note that if $L \leq \omega$ then the entire array fits in a single computer word, whereas if $L > \omega$ we need $\lceil L/\omega \rceil$ computer words to represent the automaton.

For each symbol c of the alphabet Σ, the SHIFT-AND algorithm maintains a bit mask $B[c]$ whose i-th bit is set to 1, provided that $P[i] = c$. The current configuration of the automaton is maintained in a bit mask D, which is initialized to 0^L, since initially all states are inactive. While scanning the text T from left to right, the algorithm simulates automaton transitions by the following basic shift-and operation, for each position j:

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

If the final state is active, a matching is reported at position j.

It turns out that the SHIFT-AND algorithm has an $\mathcal{O}(\lceil mn/\omega \rceil)$ worst-case time and requires $\mathcal{O}(\lceil L/\omega \rceil)$-space.

The SHIFT-AND algorithm can easily be extended to solve also the approximate matching problem of our interest.

This can be done by adapting a forward search algorithm presented in Navarro and Raffinot (2001) for the pattern matching problem with character classes and bounded gaps, with application on protein searching.[2]

To take into account δ-matches, it is enough to set to 1 the i-th bit of the masks from $B[c-\delta]$ to $B[c+\delta]$.

To handle also α-bounded gaps, we modify the automaton as follows. Let $S_0, S_1, \ldots, S_\ell$ be the states of the automaton in the order induced by its transitions, where S_0 is the initial state. Following Navarro and Raffinot (2001), the states $S_1, \ldots, S_{\ell-1}$ are called *gap-initial states*. After each gap-initial state S_i, α new states $S_{i,1}, \ldots, S_{i,\alpha}$, linearly connected by Σ-transitions, are inserted. Moreover, ε-transitions from S_i to each of the new states $S_{i,1}, \ldots, S_{i,\alpha}$ are introduced. The states $S_{i,\alpha}$, for $i = 1, \ldots, \ell-1$, are called *gap-final states*. Plainly, the number L of states of the resulting automaton is $m + (m-1)\cdot\alpha$.

[2]Though Navarro and Raffinot's algorithm is very general, as it allows to specify different gaps $x(i,j)$ at different positions, it requires that $i_1 \geq 1$, for any two consecutive nontrivial gaps $x(i_1, j_1)$ and $x(i_2, j_2)$ at adjacent positions (see footnote 1 for the notation $x(i,j)$).

```
$(\delta,\alpha)$-SHIFT-AND $(T, P, \delta, \alpha)$
    $n$ = length($T$)
    $m$ = length($P$)
    $L = m + (m-1)\cdot\alpha$
    for $c \in \Sigma$ do $B[c] = 0^L$
    $I = F = 0^L$
    $i = 0$
    for $j = 0$ to $m-1$ do
        for $c = P[j]-\delta$ to $P[j]+\delta$ do
            $B[c] = (B[c] \mid (0^{L-1}1 \ll i))$
        $i = i+1$
        if $j < m-1$ then
            $I = I \mid (0^{L-1}1 \ll (i-1))$
            $F = F \mid (0^{L-1}1 \ll (i+\alpha-1))$
            for $c \in \Sigma$ do
                for $k = i$ to $i+\alpha-1$ do
                    $B[c] = (B[c] \mid (0^{L-1}1 \ll k))$
            $i = i+\alpha$
    $M = 0^{L-1}1 \ll L-1$
    $D = 0^L$
    for $j = 0$ to $n-1$ do
        if $D \,\&\, M \neq 0^L$ then output($j$)
        $D = ((D \ll 1) \mid 0^{L-1}1) \,\&\, B[T[j]]$
        $D = D \mid (((F - (D \,\&\, I)) \,\&\, \sim F) \ll 1)$
```

Figure 3: The algorithm based on bit-parallelism for the δ-approximate matching problem with α-bounded gaps.

It can easily be seen that ε-transitions can be simulated by the following operation

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

where I is a bit mask containing 1 in the gap-initial states, and F is a bit mask containing 1 in the gap-final states.

Figure 3 shows the complete algorithm, which it is natural to call (δ,α)-SHIFT-AND. The preprocessing phase takes $\mathcal{O}(m\alpha|\Sigma|)$-time, whereas the scanning phase takes $\mathcal{O}(\lceil nm/\omega \rceil)$-time.

4.3 The (δ,α)-SEQUENTIAL-SAMPLING algorithm

The (δ,α)-SEQUENTIAL-SAMPLING algorithm (Cantone et al., 2005) 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 $\mathcal{O}(m^2\alpha)$-time and -space by the δ-BOUNDED-GAPS algorithm.

The (δ,α)-SEQUENTIAL-SAMPLING algorithm computes the occurrences of all prefixes of the pattern in continuously increasing prefixes of the text.

To be more precise, let $\mathcal{S}_i$ denote the collection of all pairs (j,k) such that $P_k \trianglelefteq^j T$, 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 $P \trianglelefteq^i T$ translates into 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 $i < n$:

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

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 $\mathcal{S}$ can be represented by its characteristic nxm matrix $\mathcal{M}$, where $\mathcal{M}[i,k]$ is 1 or 0, according to whether 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 $j \in \{\max(0, i-\alpha-1), \ldots, i-1\}$,— it is 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 in such a way that the following invariant holds:

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

one can test whether a pair (i,k) must be added to the set $\mathcal{S}_{i+1}$ in constant time, rather than in $\mathcal{O}(\alpha)$-time, thus allowing to reduce 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, $\mathcal{C}[k]$ will contain the total number of *distinct* δ-occurrences with α-gaps of P_k at positions $\max(0, i-\alpha-1)$ through $i-1$.

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 Figure 4.

```
(δ, α)-SEQUENTIAL-SAMPLING (T, P, δ, α)
 1.   $n$ = length($T$)
 2.   $m$ = 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 4: The (δ,α)-SEQUENTIAL-SAMPLING algorithm.

5 AN IMPROVED VARIANT OF THE (δ,α)-SEQUENTIAL-SAMPLING ALGORITHM

In this section we present a new efficient variant of the (δ,α)-SEQUENTIAL-SAMPLING algorithm named (δ,α)-TUNED-SEQUENTIAL-SAMPLING algorithm ((δ,α)-TSS for short). As its progenitor, the (δ,α)-TSS algorithm solves variant (c) of the problem in $\mathcal{O}(mn)$-time and $\mathcal{O}(m\alpha)$-space but, practically, it requires only $\mathcal{O}(n)$-time on the average, at least in the case of alphabets with a uniform distribution.

5.1 An estimate of the average number of matched prefixes

In the (δ,α)-SEQUENTIAL-SAMPLING algorithm in Figure 4, the **for**-loop in line 14 iterates on k from $m-2$ down to 1, for each position i in the text, yielding an $\mathcal{O}(mn)$-time complexity. However, an iteration relative to a value k can have some effect only if $\mathcal{C}[k-1] > 0$, i.e. only if $P_{k-1} \trianglelefteq^h T$ for some $i - \alpha - 1 \leq h \leq i - 1$.

As we will see below, in practical cases, at each position of the text the average number of approximate matching prefixes of the pattern P is constant. Hence, by maintaining in a linked list the nonnull positions of the array $\mathcal{C}$, we can attain an average time complexity of $\mathcal{O}(n)$, rather than $\mathcal{O}(mn)$.

In our analysis, we will assume that the pattern and text are independent random strings over an alphabet with uniform distribution. Moreover, to simplify the analysis, we will overlook some further dependence problems. Nevertheless, we will see that our theoretical results agree perfectly with the experimental results.

To begin with, we observe that the probability p_δ that two random characters of an alphabet Σ have a δ-match is given with a good approximation by

$$p_\delta \approx \frac{2\delta + 1}{\sigma}\,,$$

where σ is the size of the alphabet.

Next we estimate the probability that $P_k \trianglelefteq^i T$, where P_k is a pattern prefix of length $k+1$, with $k < m$, and i is any position in the text T. Observe that $\Pr\{P[k] =_\delta T[i]\} = p_\delta$. In addition, for each $0 \leq j < k$ and each position $\ell \geq \alpha$ in T, we have that the probability that $P[j]$ has a δ-approximate occurrence in $T[\ell - \alpha \,..\, \ell]$ is equal to $1-(1-p_\delta)^{\alpha+1}$. Such considerations suggest the following rough estimate

$$\Pr\{P_k \trianglelefteq^i T\} \approx p_\delta(1-(1-p_\delta)^{\alpha+1})^k\,,$$

which would hold as equality if the gapped δ-matchings of the characters of P_k with corresponding characters of

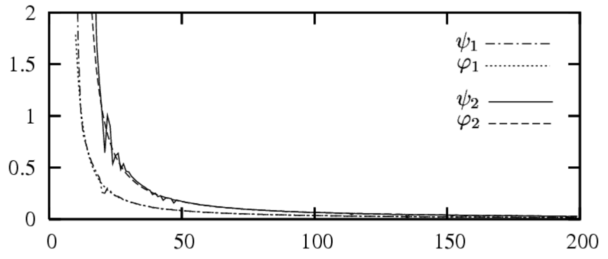

Figure 5: Comparison between the expected number φ_δ of approximate matching pattern prefixes at any given position of the text, as estimated by equation (2), and its experimental valuation ψ_δ, for $\delta = 1, 2$.

T were independent events, which is not the case. Nevertheless, there is experimental evidence that the above approximation is very accurate (see Figure 5).

Thus, an estimate of the expected number φ_δ of prefixes P_k such that $P_k \trianglelefteq^i T$, for each position i of the text, is given by

$$\begin{aligned}\varphi_\delta &\approx p_\delta \cdot \sum_{k=0}^{m-1} \left(1-(1-p_\delta)^{\alpha+1}\right)^k \\ &= p_\delta \cdot \frac{1-\left(1-(1-p_\delta)^{\alpha+1}\right)^m}{(1-p_\delta)^{\alpha+1}} .\end{aligned}$$

In fact, when m is large enough, we have

$$\varphi_\delta \approx \frac{p_\delta}{(1-p_\delta)^{\alpha+1}} . \tag{2}$$

In Figure 5 we have plotted the values φ_δ (given by (2) above) and ψ_δ as functions of the alphabet size σ, for $\delta = 1, 2$, with the fixed values $\alpha = 4$ and $m = 140$, where ψ_δ is an experimental estimate of the average number of matching prefixes of P at any given position in a random text of 10Mb. It is interesting to notice that the functions φ_δ and ψ_δ, for $\delta = 1, 2$, overlap almost perfectly, thus supporting the legitimacy of our approximations in the calculation of φ_δ.

5.2 Tuning the (δ, α)-SEQUENTIAL-SAMPLING algorithm

In view of the results of the preceding section, at any time during the execution of the (δ, α)-SEQUENTIAL-SAMPLING algorithm the average number of nonnull items in the array $\mathcal{C}$ is constant. Therefore, if the indices of such nonnull items are maintained in an ordered linked list $\mathcal{L}$, the **for**-loop at line 14 in Figure 4 can be executed in constant average time, rather than in $\mathcal{O}(m)$-time. This will have the overall effect to reduce the $\mathcal{O}(mn)$-time complexity of the (δ, α)-SEQUENTIAL-SAMPLING algorithm to $\mathcal{O}(n)$ average time complexity.

In more detail, our proposed variant (δ, α)-TSS works as follows. For each position i of the text T, the (δ, α)-TSS algorithm scans the ordered list $\mathcal{L}$ in decreasing order, and for each value k in $\mathcal{L}$ it performs the following operations. After updating the entries $\mathcal{C}[k]$ and $\mathcal{M}[j, k]$, as in lines 10 and 11 of the (δ, α)-SEQUENTIAL-SAMPLING algorithm, the (δ, α)-TSS algorithm checks preliminarily whether $\mathcal{C}[k] = 0$. If this is the case, the current value of k is removed from $\mathcal{L}$. Otherwise, namely if $\mathcal{C}[k] > 0$ holds, the (δ, α)-TSS algorithm tests whether the prefix P_{k+1} has an approximate gapped matching at position i in the text T just by checking if $P[k+1] =_\delta T[i]$. If this is so and if $k + 1 = m - 1$, then a matching is reported at position i. Otherwise, namely if $k + 1 < m - 1$, the entries $\mathcal{M}[i, k+1]$ and $\mathcal{C}[k+1]$ are updated in such a way that $\mathcal{M}[i, k+1]$ contains the correct number of matches of P_{k+1} at position i in T and the invariant (1) is maintained; then the value $k+1$ is inserted in $\mathcal{L}$ just before k, provided that it is not already there. When the list $\mathcal{L}$ has been completely scanned, the (δ, α)-TSS algorithm checks whether $P[0] =_\delta T[i]$, and if this is the case the value 0 is inserted in the list $\mathcal{L}$.

Figure 6 shows the complete code of the (δ, α)-TSS algorithm. The list $\mathcal{L}$ is implemented in a circular fashion by means of the array $next$ of size m. The entry $next[m-1]$ is used as an extra sentinel which will always point to the first (i.e., highest) value of k contained in the list $\mathcal{L}$. In addition, to manage insertions and deletions in $\mathcal{L}$ efficiently, it is convenient to maintain a pointer p to the predecessor of the current value of k in $\mathcal{L}$.

As the (δ, α)-SEQUENTIAL-SAMPLING algorithm, the (δ, α)-TSS algorithm has $\mathcal{O}(mn)$-time and $\mathcal{O}(m\alpha)$-space worst-case complexity. However, since the average number of matched prefixes of the pattern at any given position of the text is constant for sufficiently long patterns, in practice the (δ, α)-TSS algorithm works in $\mathcal{O}(n)$-time. This is confirmed experimentally, as will be shown in the next section.

Note that, if we are interested only in variant (b) of our approximate matching problem, we can further improve the efficiency of the (δ, α)-TSS algorithm, at least in the case in which $\alpha + 2$ bits can fit in a computer word. Such assumption is realistic, since in practical cases the length of the gap is not greater than 16. In this case, each column of the table $\mathcal{M}$ can be represented by a computer word stored in the array $\mathcal{C}$. The resulting algorithm turns out to be slightly more efficient than the (δ, α)-TSS algorithm in Figure 6.

6 EXPERIMENTAL RESULTS

In Figures 7, 8, and 9, we report experimental data relative to an extensive comparison of the running times of our algorithm (δ, α)-TUNED-SEQUENTIAL-SAMPLING (TTS), 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 have been 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 $\sigma = 60, 90$ and on a real music text buffer.

Each Randσ problem consisted in searching for 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 has been set to 2, whereas the gap bound α has been set to 4.

The tests on real music have been performed on a

```
(δ,α)-TUNED-SEQUENTIAL-SAMPLING (T, P, δ, α)
 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.  $next[0] = next[m - 1] = m - 1$
 8.  for $i = 0$ to $n - 1$ do
 9.      $j = i \mod (\alpha + 2)$
10.      $p = m - 1$
11.      $k = next[m - 1]$
12.      while $k < m - 1$ do
13.          $\mathcal{C}[k] = \mathcal{C}[k] - \mathcal{M}[j, k]$
14.          $\mathcal{M}[j, k] = 0$
15.          if $(\mathcal{C}[k] = 0)$ then
16.              $next[p] = next[k]$
17.          else
18.              if $(P[k + 1] =_\delta T[i])$ then
19.                  if $(k = m - 2)$ then
20.                      output($i$)
21.                  else
22.                      $\mathcal{M}[j, k + 1] = \mathcal{C}[k]$
23.                      $\mathcal{C}[k + 1] = \mathcal{C}[k + 1] + \mathcal{C}[k]$
24.                      if $p > k + 1$ then
25.                          $next[p] = k + 1$
26.                          $next[k + 1] = k$
27.              $p = k$
28.          $k = next[k]$
29.      if $(P[0] =_\delta T[i])$ then
30.          $\mathcal{M}[j, 0] = 1$
31.          $\mathcal{C}[0] = \mathcal{C}[0] + 1$
32.          if $p > 0$ then $next[p] = 0$
```

Figure 6: The (δ, α)-TUNED-SEQUENTIAL-SAMPLING algorithm.

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 absolute pitch 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 seconds.

Experimental results show that the (δ, α)-TSS algorithm is faster than all other algorithms and its superiority is more noticeable as the size of the pattern increases. Moreover, it turns out from experimental results that its running time does not depend on the length of the pattern.

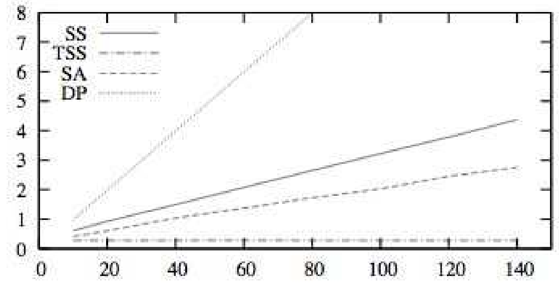

Figure 7: Experimental results on a Rand60 problem with $\delta = 2$ and $\alpha = 4$.

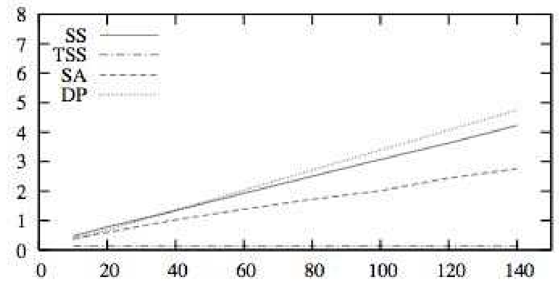

Figure 8: Experimental results on a Rand90 problem with $\delta = 2$ and $\alpha = 4$.

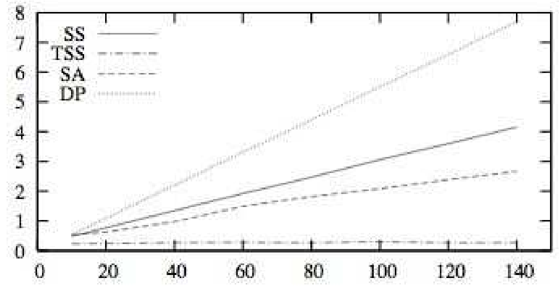

Figure 9: Experimental results on Real Data with $\delta = 2$ and $\alpha = 4$.

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, which exhibits a linear behavior in practical cases. The algorithm has been compared against various existing solutions for the same problem. Experimental results have shown that our algorithm is very fast. The performance 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.

REFERENCES

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

D. Cantone, S. Cristofaro, and S. Faro. An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences. In S.E. Nikoletseas, editor, *Proc. of the 4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, number 3503 in Lecture Notes in Computer Science, pages 428–439. Springer-Verlag, Berlin, 2005.

M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, and W. Rytter. Finding Motifs with Gaps. In *Proc. of International Symposium on Music Information Retrieval (ISMIR '00)*, Plymouth, USA, poster paper, pages 306–317, 2000.

M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis, and K. Tsichlas. Approximate string matching with gaps. *Nordic J. Comput.*, 9(1):54–65, 2002a.

G. Navarro and M. Raffinot. Fast and simple character classes and bounded gaps pattern matching, with application to protein searching. In *Proc. 5th Annual International ACM Conference on Computational Molecular Biology (RECOMB'01)*, pages 231–240, 2001.