International Journal of Foundations of Computer Science
© World Scientific Publishing Company

NEW EFFICIENT BIT-PARALLEL ALGORITHMS FOR THE (δ, α)-MATCHING PROBLEM WITH APPLICATIONS IN MUSIC INFORMATION RETRIEVAL

DOMENICO CANTONE [*]
SALVATORE CRISTOFARO [†]
SIMONE FARO [‡]

Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

We present new efficient variants of the (δ, α)-Sequential-Sampling algorithm, recently introduced by the authors, for the δ-approximate string matching problem with α-bounded gaps. These algorithms, which have practical applications in music information retrieval and analysis, make use of the well-known technique of bit-parallelism. An extensive comparison with the most efficient algorithms present in the literature for the same search problem shows that our newly proposed solutions achieve very good results in practice, in terms of both space and time complexity, and, in most cases, they outperform existing algorithms. Moreover, we show how to adapt our algorithms to other variants of the approximate matching problem with gaps, which are particularly relevant for their applications in other fields than music (e.g., molecular biology).

Keywords: approximate string matching with gaps, bit-parallel algorithms, music information retrieval.

1. Introduction

The δ-approximate string matching problem with α-bounded gaps (or (δ, α)-matching) $[5, 4, 2]$ is a generalization of the δ-approximate string matching problem [1] and arise in many questions in music information retrieval and music analysis. This is particularly true, for instance, in the context of monophonic music, when one wants to retrieve occurrences of a given melody from a complex musical score. We recall that two (monophonic) musical sequences have a δ-approximate matching if they have the same length (i.e., they contain the same number of notes) and notes at the same positions differ by at most δ semitones. Then, we say that a melody

[*] cantone@dmi.unict.it
[†] cristofaro@dmi.unict.it
[‡] faro@dmi.unict.it

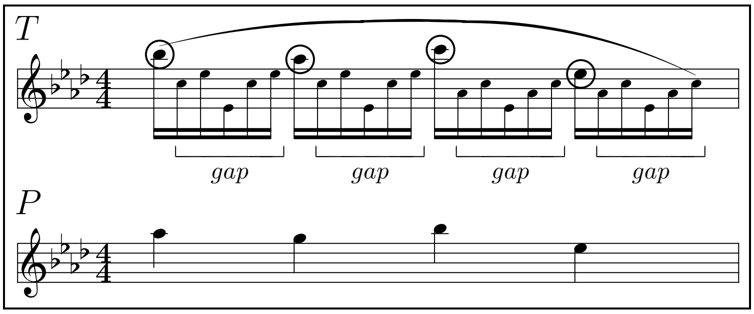

Fig. 1. An excerpt from study *Op. 25 Nr. 1 for Piano Solo* by F. Chopin (first score). Melody P has a δ-approximate occurrence with α-bounded gaps in T, for $\delta \geq 2$ and $\alpha \geq 5$, indicated by the circled notes. Tiny notes represent arpeggios and form the gaps. Notice that, in this case, the gaps are all of the same size 5. Observe also that the first and the third note of P differ from the corresponding matchings in T (circled notes) by 2 semitones; the second note differ by 1 semitone, whereas the last note equals its matching. In any case, the difference between a note and its matching does not exceed 2 semitones, so that we have a (δ, α)-occurrence of P in T, for any $\delta \geq 2$ and $\alpha \geq 5$.

(or pattern) P has a δ-approximate occurrence with α-bounded gaps within a musical score (or text) T—or, more shortly, a (δ, α)-occurrence—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 notes (the gap) between any two consecutive positions. Thus, δ-approximate matching with α-bounded gaps turns out to be very effective for finding closely related but not necessarily identical occurrences of melodies (δ-approximation), when small values of δ are allowed. In addition, gaps allow to skip over various kinds of musical ornamentations (e.g., arpeggios) which are of common use, especially in classical music. See Figure 1 for a pictorial illustration.

We mention also that many variants and generalizations of the δ-approximate string matching problem with α-bounded gaps have been considered for applications in fields other than music, such as, for instance, molecular biology [9, 10]. It turns out that the algorithms we present in this paper can be easily adapted so as to handle efficiently some of these important variants (see Appendix).

The paper is organized as follows. In the next section we introduce some basic notations and give a formal definition of the δ-approximate string matching problem with α-bounded gaps. Subsequently, in Section 3 we review some of the most efficient algorithms for this problem and then, in Section 4, we describe our newly proposed algorithms. Experimental results of an extensive comparison of our algorithms with some of the most efficient ones present in the literature are reported and discussed in Section 5, whereas in Section 6 we draw our concluding remarks. Finally, in the Appendix we show how the algorithms presented in Section 4 can be adapted so as to solve other variants of the approximate string matching problem with gaps, which are more relevant to the field of molecular biology.

2. Basic Definitions and Properties

Before entering into details, we review a bit of notations and terminology. A string P of length $|P| = m > 0$ is represented as a finite array $P[0\,..\,m-1]$. By $P[i]$ we denote the $(i+1)$-st symbol of P, for $0 \leq i < |P|$. Likewise, by $P[i\,..\,j]$ we denote the substring of P contained between the $(i+1)$-st and the $(j+1)$-st symbols of P (both inclusive), where $0 \leq i \leq j < |P|$. The substrings of the form $P[0\,..\,j]$, also denoted by P_j, for $0 \leq j < |P|$, are the nonempty PREFIXES of P.

Let Σ be a finite alphabet of *integer numbers* and let δ and α be nonnegative integers. Two symbols a and b of Σ are said to be δ-APPROXIMATE, in which case we write $a =_\delta b$, if $|a-b| \leq \delta$. Given a pattern P of length m and a text T of length n over the alphabet Σ, by a δ-APPROXIMATE OCCURRENCE WITH α BOUNDED GAPS OF P IN T, or simply a (δ, α)-OCCURRENCE OF P IN T, we mean a sequence $(i_0, i_1, \ldots, i_{m-1})$ of indices such that

(1) $0 \leq i_0 < i_1 < \cdots < i_{m-1} < n$,
(2) $T[i_j] =_\delta P[j]$, for $0 \leq j < m$, and
(3) $i_h - i_{h-1} \leq \alpha + 1$, for $0 < h < m$, provided that $m > 1$.

Given an index i, with $0 \leq i < n$, a (δ, α)-OCCURRENCE OF P AT POSITION i IN T is a (δ, α)-occurrence $(i_0, i_1, \ldots, i_{m-1})$ of P in T such that $i_{m-1} = i$. We write $P \trianglelefteq^i_{\delta,\alpha} T$ to mean that there is a (δ, α)-occurrence of P at position i in T (in fact, when the bounds δ and α are well understood from the context, one can simply write $P \trianglelefteq^i T$).

The δ-APPROXIMATE STRING MATCHING PROBLEM WITH α-BOUNDED GAPS, or (δ, α)-MATCHING PROBLEM, is the problem of finding the (δ, α)-occurrences of a given pattern P in a given text T. More precisely, the following variants may be considered [2]: (a) find all (δ, α)-occurrences of P in T; (b) find all positions i in T such that $P \trianglelefteq^i_{\delta,\alpha} T$; (c) find the number of all distinct (δ, α)-occurrences of P at position i in T, for each position i in T. In this paper we will concentrate only on the variant (b). [a]

[a] Notice that, in the context of exact (or even δ-approximate) string matching with no gaps, most string matching algorithms are designed so as to produce as output the positions of the text at which the occurrences of the pattern start, whereas, according to our definitions, in the case of (δ, α)-matching we are interested in determining the positions of the text at which the occurrences of the pattern terminate (which is in agreement with most of the literature on (δ, α)-matching). From our point of view, the main reason behind this choice is that it allows for a recursive formulation of the problem (see Lemma 1), leading naturally to algorithms which follow the traditional paradigm of left-to-right searching, namely the occurrences of the pattern are searched for in the text sequentially proceeding from left to right, from the leftmost occurrence to the rightmost one.

If one is interested in finding the starting positions of the approximate occurrences with gaps of a pattern P in a text T, a possible, simple solution could be that of searching the reverse of the pattern (i.e., the string $P^R = P[m-1] \cdots P[0]$) in the reverse of the text (i.e., the string $T^R = T[n-1] \cdots T[0]$). In fact, if i is a position of T^R at which an approximate occurrence with gaps of P^R ends, then it is immediate to verify that the pattern P has an approximate occurrence with gaps beginning at position $n-1-i$ in the text T, namely the reverse of the given occurrence

The following property is an immediate consequence of the above definitions:

Lemma 1. *Let P and T be respectively a pattern of length m and a text of length n over an alphabet Σ of integer numbers. Moreover, let δ and α be nonnegative integers. Then,*

(a) $P_0 \trianglelefteq^{i}_{\delta,\alpha} T \Leftrightarrow T[i] =_{\delta} P[0]$*;*
(b) $P_j \trianglelefteq^{i}_{\delta,\alpha} T \Leftrightarrow T[i] =_{\delta} P[j]$ AND $(\exists k \in \{1,\ldots,\alpha+1\} : i-k \geq 0$ AND $P_{j-1} \trianglelefteq^{i-k}_{\delta,\alpha} T)$, *for* $0 \leq i < n$ *and* $0 < j < m$*.*

The following notations and terminology will be used in connection with the bit-parallelism technique. A BIT MASK (or BINARY STRING) is a string whose symbols are the bits 0 and 1. In writing bit masks, we will use exponentiation to denote the concatenation of multiple copies of single bits or of bit masks as well. Thus, for instance, $101^{3}0$ denotes the bit mask 101110 and $1(01)^{3}0$ denotes 10101010.

We will employ the following standard operations on bit masks: the bit-wise and and bit-wise or operations, denoted respectively by & and |, and the right-shift and left-shift operations, denoted respectively by $\gg$ and $\ll$. We will also use the arithmetic operations of addition "+" and subtraction "−" between bit masks to calculate, respectively, the binary representations of the sum and of the difference between the nonnegative integers represented by the bit masks. It turns out that in all expressions of the form $X - Y$ which we will encounter in the rest of the paper, the nonnegative integer represented by the bit mask X is always no less than the integer represented by Y. Likewise, in all expressions of the form $X \,\&\, Y$, $X \,|\, Y$, $X + Y$, and $X - Y$, the two bit masks X and Y will have the same length, so that we will not need to deal with special cases. Notice that if X and Y are bit masks of the same length ℓ, then the length of the bit masks $X \,\&\, Y$, $X \,|\, Y$, and $X - Y$ is ℓ, whereas the length of $X + Y$ might be $\ell + 1$, due to the carry bit.

Concerning the unitary left-shift operation, we will assume that the string $X \ll 1$ has the same length as X, if the leading bit of X is 0 (which corresponds to just dropping from X its leading bit 0 and adding a final bit 0), otherwise its length is one more than that of X (which corresponds to adding a final bit 0 to X). The k-ary left-shift operation is then defined as k iterations of the unitary left-shift. Instead, the right-shift is defined in such a way that $X \gg k$ has always the same length of X. Thus, for instance, if $X = 00110$ we have: $X \ll 1 = 01100$, $X \ll 2 = 11000$, $X \ll 3 = 110000$, and $X \gg 1 = 00011$, $X \gg 2 = 00001$, $X \gg 3 = X \gg 4 = \ldots = 00000$.

As far as concerns complexity issues, we will assume the computational model in which each of the above operations can be executed in $\mathcal{O}(\lceil L/w \rceil)$-space and time, where L is the length of the result and w is the computer word length. In fact, a bit mask B whose length exceeds the computer word length w can be readily represented by $\lceil |B|/w \rceil$ computer words.

of P^R in T^R. All of the algorithms we present in this paper allow for a similar inverse search.

The following additional notations will also be used. Given a matrix $\mathcal{M}$ of dimensions $h \times k$, we denote by $(\mathcal{M})_{i,j}$ the element of $\mathcal{M}$ located at the intersection of the $(i+1)$-st row and $(j+1)$-st column of $\mathcal{M}$, for $0 \leq i < h$ and $0 \leq j < k$. A bit-matrix is a matrix whose entries belong to $\{\mathsf{0}, \mathsf{1}\}$. Given two integers h and k, with $h \leq k$, we denote by $[h\,..\,k]$ the set (interval) of all integers x such that $h \leq x \leq k$.

In the sequel, we will assume that all patterns and texts in the paper are strings over an alphabet Σ of size $\sigma \geq 1$, having the form $\{0, 1, \ldots, \sigma - 1\}$.

3. Efficient algorithms for the (δ, α)-matching problem

The δ-approximate string matching problem with α-bounded gaps has been first formally defined in [5], where the δ-Bounded-Gaps algorithm has been proposed (see also [4, 2]). The δ-Bounded-Gaps algorithm, whose time and space complexity is $\mathcal{O}(nm)$, with n and m the lengths of the text T and of the pattern P respectively, is presented as an incremental procedure, based on the dynamic programming approach. Scanning the pattern P from left to right, the δ-Bounded-Gaps algorithm looks for the (δ, α)-occurrences of each prefix P_j of the pattern P in the whole text T, for $0 \leq j < m$. Specifically, the δ-Bounded-Gaps algorithm proceeds by filling in a table D of dimensions $m \times n$ such that

$$D[j, i] = \max(\{k \geq 0 : i - \alpha \leq k \leq i \text{ and } P_j \trianglelefteq^k T\} \cup \{-1\})$$

for $0 \leq j < m$ and $0 \leq i < n$. Notice that $P_j \trianglelefteq^i T$ if and only if $D[j, i] = i$.

An algorithm, slightly more efficient than the δ-Bounded-Gaps, has been presented by the authors in [2], under the name (δ, α)-Sequential-Sampling. As in the case of the δ-Bounded-Gaps algorithm, the (δ, α)-Sequential-Sampling is also based on dynamic programming, but it follows a different computation ordering than the δ-Bounded-Gaps algorithm does; more precisely, it scans the text T from left to right and for each position i of T it looks for the (δ, α)-occurrences at position i of all prefixes of the pattern P. The (δ, α)-Sequential-Sampling algorithm has an $\mathcal{O}(nm)$ running time and requires $\mathcal{O}(m\alpha)$-space. A much more efficient variant of it is the (δ, α)-Tuned-Sequential-Sampling algorithm, which has an average case running time of $\mathcal{O}(n)$, in the case in which α is assumed constant (cf. [3]).

Another algorithm, named (δ, α)-Shift-And, has also been described in [3]. The (δ, α)-Shift-And algorithm is a very simple variant of a forward search algorithm presented in [9] for a pattern matching problem with gaps and character classes, particularly suited for applications to protein searching. It uses bit-parallelism to simulate the behavior of a nondeterministic finite automaton with ε-transitions. The automaton has $\ell = (\alpha + 1)(m - 1) + 2$ states, and the simulation is carried out by representing it as a bit mask B of length $\ell - 1$ (the initial state of the automaton need not be represented in the bit mask since it is always active during the computation). When $\ell < w$ (the computer word length), the entire bit mask B fits in a single computer word. In this case the (δ, α)-Shift-And algorithm becomes extremely fast in practice.

Other efficient algorithms for the (δ, α)-matching problem have been presented more recently in [6] and [7]. In particular, [6] presents two algorithms, called DA-bpdb and DA-mloga-bits. The first one inherits the basic idea from the dynamic programming algorithm δ-Bounded-Gaps presented in [4]. It uses bit-parallelism to compute an $m \times n$ bit-matrix $\mathcal{D}$ such that $(\mathcal{D})_{j,\,i} = 1$ if and only if $P_j \trianglelefteq^i T$, for $0 \leq j < m$ and $0 \leq i < n$. Basically, the algorithm DA-bpdb partitions each row of the matrix $\mathcal{D}$ as a sequence of $\lceil n/w \rceil$ consecutive bit masks, each of which represents a group of w bits on that row. Then, the computation of the j-th bit mask in row i is performed bit-parallelely by using the $(j-1)$-st and the j-th bit masks of the $(i-1)$-st row. It turns out that DA-bpdb has an $\mathcal{O}(n\delta + \lceil n/w \rceil m)$ worst-case execution time, which becomes $\mathcal{O}(\lceil n/w \rceil \lceil \alpha\delta/\sigma \rceil + n)$ on the average. The second algorithm presented in [6], namely DA-mloga-bits, is based on a compact representation, in the form of a systolic array, of the nondeterministic automaton used in the algorithm (δ, α)-Shift-And. The systolic array is composed of m building blocks, called *counters* in [6], one for each symbol of the pattern, and is represented as a bit mask of length $(m-1)(\lceil \log_2(\alpha+1) \rceil + 1) + 1$. Notice that this improves the representations used in [9, 3] in which $(\alpha+1)(m-1)+1$ bits are needed to represent the automaton. It turns out that the DA-mloga-bits algorithm has an $\mathcal{O}(n\lceil (m \log_2 \alpha)/w \rceil)$ worst-case searching time.

The algorithms presented in [7], called SDP-rows, SDP-columns, SDP-simple, and SDP-simple-compute-L_0, use different computation orderings, in combination with sparse dynamic programming techniques, to implement the calculation of the table D above. Specifically, in the case of the SDP-rows algorithm, the computation is performed row-wise, whereas a column-wise computation is used by SDP-columns. The algorithm SDP-simple, which can be considered as a brute force variant of SDP-rows, performs very well in practice, especially for small values of δ and α; SDP-simple-compute-L_0 improves the average case running time of SDP-simple by using a Boyer-Moore-Horspool-like shifting strategy [8], suitably adapted to handle gaps. In particular, the latter two algorithms turn out to be among the most efficient ones, in terms of running time, in many practical cases, especially for small values of α, as shown in [7]. However, although these algorithms are very fast in practice, they require additional $\mathcal{O}(n)$-space, plus $\mathcal{O}(\sigma)$-space in the case of SDP-simple-compute-L_0.

4. New efficient variants of the (δ, α)-Sequential-Sampling algorithm

In this section we present four new efficient variants of the algorithm (δ, α)-Sequential-Sampling, all based on bit-parallelism. In particular, one of these variants, the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm, is extremely efficient in most practical cases and outperforms both algorithms SDP-simple and SDP-simple-compute-L_0. Also, the variant (δ, α)-Sequential-Sampling-BP$^+$ turns out to be faster than existing algorithms (e.g., (δ, α)-Shift-And) in the case of short patterns and very small values of α.

We begin by describing the general approach.

Given a text T of length n and a pattern P of length m, let $\mathcal{M}_i$ be the bit-matrix of dimensions $(\alpha+1) \times m$ such that

$$(\mathcal{M}_i)_{k,j} = \begin{cases} 1 & \text{if } i - \alpha + k \geq 0 \text{ AND } P_j \trianglelefteq^{i-\alpha+k} T \\ 0 & \text{otherwise}, \end{cases}$$

for $-1 \leq i < n$, $0 \leq j < m$ and $0 \leq k \leq \alpha$. Notice that, for $0 \leq i < n$ and $0 \leq j < m$, we have $P_j \trianglelefteq^i T$ if and only if $(\mathcal{M}_i)_{\alpha,j} = 1$. Thus, the problem of determining the positions i of T at which $P \trianglelefteq^i T$ holds translates into the problem of determining all values i such that $(\mathcal{M}_i)_{\alpha,m-1} = 1$, which in turn reduces to the problem of effectively computing the matrices $\mathcal{M}_{-1}, \mathcal{M}_0, \ldots, \mathcal{M}_{n-1}$. This can be done as follows. To begin with, notice that, by the very definition of the matrices $\mathcal{M}_{-1}, \mathcal{M}_0, \ldots, \mathcal{M}_{n-1}$, we have

$$(\mathcal{M}_i)_{k,j} = (\mathcal{M}_{i-1})_{k+1,j} , \tag{3}$$

for $0 \leq i < n$, $0 \leq j < m$ and $0 \leq k < \alpha$; i.e., the first α rows of $\mathcal{M}_i$ coincide with the last α rows of $\mathcal{M}_{i-1}$. In addition, by Lemma 1, we have also that

$$(\mathcal{M}_i)_{\alpha,j} = \begin{cases} 1 & \text{if } T[i] =_\delta P[j] \text{ AND} \\ & \quad (j = 0 \text{ OR } (\exists k \in \{0, \ldots, \alpha\} : (\mathcal{M}_{i-1})_{k,j-1} = 1)) \\ 0 & \text{otherwise}, \end{cases} \tag{4}$$

for $0 \leq i < n$ and $0 \leq j < m$, which expresses the $(j+1)$-st item in the last row of matrix $\mathcal{M}_i$ in terms of the j-th column of matrix $\mathcal{M}_{i-1}$. These recursive relations, coupled with the initial condition $\mathcal{M}_{-1} = 0^{(\alpha+1)\times m}$, allow one to compute the matrices $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{n-1}$ in an iterative fashion, starting from the initial matrix $\mathcal{M}_{-1}$.

For instance, in the case of the (δ, α)-Sequential-Sampling algorithm, the computation is carried out by calculating in sequence the matrices $\mathcal{M}_{-1}, \mathcal{M}_0, \ldots, \mathcal{M}_{n-1}$, which are maintained in a circular fashion in a bit table M of dimensions $(\alpha+1) \times m$. More specifically, initially the table M is filled in with all 0's (which corresponds to the initial matrix $\mathcal{M}_{-1}$). Then, n iterations are performed, for $i = 0, 1, \ldots, n-1$. At iteration i, the last row of $\mathcal{M}_i$ is computed, by calculating in turn the elements $(\mathcal{M}_i)_{\alpha,m-1}, (\mathcal{M}_i)_{\alpha,m-2}, \ldots, (\mathcal{M}_i)_{\alpha,0}$ according to recurrence (4), and stored at the row of index $i \bmod (\alpha+1)$ of table M; thus, just after step i, we have that $\mathsf{M}[(i+k+1) \bmod (\alpha+1), j] = (\mathcal{M}_i)_{k,j}$, for $0 \leq k \leq \alpha$ and $0 \leq j < m$. In performing such step, the (δ, α)-Sequential-Sampling algorithm makes use of an additional array C, of length m, whose $(j+1)$-st entry $C[j]$ counts the number of 1's in the $(j+1)$-st column of M, for $0 \leq j < m$. This allows to perform each step of the computation in $\mathcal{O}(m)$-time, yielding an overall running time of $\mathcal{O}(nm)$.[b]

[b] We mention here that the (δ, α)-Sequential-Sampling algorithm in its original form presented in [2] allows one to count the number of all distinct (δ, α)-approximate occurrences of each prefix P_j of the pattern P at any position i of the text T, and not only to check whether $P_j \trianglelefteq^i T$.

The computation of the matrices $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{n-1}$ can be carried out in various ways using the bit-parallelism technique, as we show next.

The basic idea is to represent each column of the matrices $\mathcal{M}_i$ as a bit mask of length $\alpha + 1$ (which is very natural, since the columns of $\mathcal{M}_i$ are nothing but vectors of bits). Consequently, the whole matrix $\mathcal{M}_i$ can be represented as an array of m bit masks, each of which corresponds to a column of $\mathcal{M}_i$, and each of which fits in a single computer word if $\alpha < w$, where w is the computer word length (see below for a brief discussion on the condition $\alpha < w$).[c]

To be more precise, let us denote with $\mathcal{C}_i^{(j)}$ the bit mask of length $\alpha + 1$ such that $\mathcal{C}_i^{(j)}[k] = (\mathcal{M}_i)_{k,j}$, for $-1 \leq i < n$, $0 \leq j < m$, and $0 \leq k \leq \alpha$.[d] Then, by (3), we have that $\mathcal{C}_i^{(j)}[0 \mathinner{..} \alpha - 1] = \mathcal{C}_{i-1}^{(j)}[1 \mathinner{..} \alpha]$, i.e., the first α bits of $\mathcal{C}_i^{(j)}$ coincide with the last α bits of $\mathcal{C}_{i-1}^{(j)}$. Moreover, by (4), we have that the last bit of $\mathcal{C}_i^{(j)}$ is $\mathsf{1}$, if $T[i] =_\delta P[j]$ and $\mathcal{C}_{i-1}^{(j-1)} \neq \mathsf{0}^{\alpha+1}$; otherwise it is $\mathsf{0}$, provided that $j > 0$. If $j = 0$, the last bit of $\mathcal{C}_i^{(0)}$ is $\mathsf{1}$ if and only if $T[i] =_\delta P[0]$ holds. Therefore, if we put $I = \mathsf{01}^\alpha$, we obtain

$$\mathcal{C}_i^{(j)} = \begin{cases} ((\mathcal{C}_{i-1}^{(j)} \,\&\, I) \ll 1) \,|\, \mathsf{0}^\alpha \mathsf{1}, & \text{if } T[i] =_\delta P[j] \text{ AND } (j = 0 \text{ OR } \mathcal{C}_{i-1}^{(j-1)} \neq \mathsf{0}^{\alpha+1}) \\ (\mathcal{C}_{i-1}^{(j)} \,\&\, I) \ll 1, & \text{otherwise}, \end{cases} \tag{5}$$

for $0 \leq i < n$ and $0 \leq j < m$. Such relations suggest the simple algorithm reported in Figure 2 (on the left), named (δ, α)-Sequential-Sampling-HBP ((δ, α)-S-S-HBP, for short), which uses an array C of length m to maintain the bit masks $\mathcal{C}_i^{(0)}, \mathcal{C}_i^{(1)}, \ldots, \mathcal{C}_i^{(m-1)}$. This algorithm is very close in spirit to the (δ, α)-Sequential-Sampling, improving the space complexity of the latter algorithm to $\mathcal{O}(m\lceil\alpha/w\rceil)$, though its running time, which is $\mathcal{O}(nm\lceil\alpha/w\rceil)$, is worse than that of the (δ, α)-Sequential-Sampling algorithm. The reason is that, in general, we need $\lceil(\alpha + 1)/w\rceil$ computer words to represent a bit mask of length $\alpha + 1$. Consequently, any update of the entry $C[j]$ costs $\mathcal{O}(\lceil\alpha/w\rceil)$-time, for $j = 0, 1, \ldots, m - 1$. However, we notice that in almost all practical applications in music information retrieval the value of the gap bound α is at most 10 (or less), therefore smaller than the size w of a computer word (which is 32 or 64). Thus, in practice, a bit mask of length $\alpha + 1$ can be maintained in a single computer word and in this case it turns out that the (δ, α)-Sequential-Sampling-HBP algorithm is faster than the (δ, α)-Sequential-Sampling.

Now, by a trick similar to the one employed in the (δ, α)-Tuned-Sequential-Sampling algorithm, we obtain a variant of the (δ, α)-Sequential-Sampling-HBP which performs extremely well in practice, as will be discussed in the next section.

As in the case of the (δ, α)-Tuned-Sequential-Sampling algorithm, we observe that, during each step of the computation of the (δ, α)-Sequential-Sampling-HBP

[c] Notice that a similar idea of packing the columns of a bit-matrix into computer words has been already introduced by the authors in [3], in connection with the algorithm (δ, α)-Tuned-Sequential-Sampling. Here, we have further refined it.

[d] Notice that $P \trianglelefteq^i T$ holds if and only if the the last bit of $\mathcal{C}_i^{(m-1)}$ (i.e., $\mathcal{C}_i^{(m-1)}[\alpha]$) is a $\mathsf{1}$, which corresponds to the condition that $\mathcal{C}_i^{(m-1)} \,\&\, \mathsf{0}^\alpha\mathsf{1} \neq \mathsf{0}^{\alpha+1}$.

```
$(\delta, \alpha)$-S-S-HBP($P$, $m$, $T$, $n$, $\delta$, $\alpha$)
 1.  for $j := 0$ to $m - 1$ do
 2.     $C[j] := 0^{\alpha+1}$
 3.  $I := (0^{\alpha}1 \ll \alpha) - 0^{\alpha}1$
 4.  for $i := 0$ to $n - 1$ do
 5.     for $j := m - 1$ downto $1$ do
 6.        $C[j] := (C[j] \,\&\, I) \ll 1$
 7.        if $T[i] =_{\delta} P[j]$ AND $C[j-1] \neq 0^{\alpha+1}$
 8.        then $C[j] := C[j] \mid 0^{\alpha}1$
 9.     $C[0] := (C[0] \,\&\, I) \ll 1$
10.     if $T[i] =_{\delta} P[0]$ then
11.        $C[0] := C[0] \mid 0^{\alpha}1$
12.     if $(C[m-1] \,\&\, 0^{\alpha}1) \neq 0^{\alpha+1}$ then
13.        print($i$)
```

```
$(\delta, \alpha)$-T-S-S-HBP($P$, $m$, $T$, $n$, $\delta$, $\alpha$)
 1.  for $j := 0$ to $m - 1$ do $C[j] := 0^{\alpha+1}$
 2.  $next[0] := next[m] := m$
 3.  $I := (0^{\alpha}1 \ll \alpha) - 0^{\alpha}1$
 4.  for $i := 0$ to $n - 1$ do
 5.     $p := m$
 6.     $j := next[p]$
 7.     while $j < m$ do
 8.        if $j < m - 1$ AND $T[i] =_{\delta} P[j+1]$ then
 9.           $C[j+1] := C[j+1] \mid 0^{\alpha}1$
10.           if $p > j + 1$ then
11.              $next[p] := j + 1$
12.              $next[j+1] := j$
13.              $p := j + 1$
14.        $C[j] := (C[j] \,\&\, I) \ll 1$
15.        if $C[j] = 0^{\alpha+1}$ then $next[p] := next[j]$
16.        else $p := j$
17.        $j := next[p]$
18.     if $T[i] =_{\delta} P[0]$ then
19.        $C[0] := C[0] \mid 0^{\alpha}1$
20.        if $p > 0$ then $next[p] := 0$
21.     if $(C[m-1] \,\&\, 0^{\alpha}1) \neq 0^{\alpha+1}$ then
22.        print($i$)
```

Fig. 2. The (δ, α)-Sequential-Sampling-HBP algorithm (on the left) and the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm (on the right) for the δ-approximate string matching problem with α-bounded gaps.

algorithm, an iteration of the **for-loop** at line 5 relative to a value of $j > 0$ has no effect if the items $C[j]$ and $C[j-1]$ are both null, i.e., if $C[j] = C[j-1] = 0^{\alpha+1}$. In fact, the only items of the array C which need to be updated are the $C[j]$'s such that $C[j] \neq 0^{\alpha+1}$ or (if $j > 0$) $C[j-1] \neq 0^{\alpha+1}$. Therefore, it is enough to scan only those positions j of the array C such that $C[j] \neq 0^{\alpha+1}$. Thus, for each such j, we first check whether $T[i] =_{\delta} P[j+1]$, provided that $j < m-1$, and, if this is the case, we update the entry $C[j+1]$ by assigning to it the bit mask $C[j+1] \mid 0^{\alpha}1$. After that, $C[j]$ is updated as in line 6 of the (δ, α)-Sequential-Sampling-HBP algorithm. To perform such process, the positions j of the nonnull items of C (i.e., the j's such that $C[j] \neq 0^{\alpha+1}$) are maintained into an ordered, linked list $\mathcal{L}$, which is scanned from the highest value of j up to the lowest one. The resulting algorithm, named (δ, α)-Tuned-Sequential-Sampling-HBP ((δ, α)-T-S-S-HBP, for short), is reported in Figure 2 (on the right). Notice that the list $\mathcal{L}$ is implemented as a circular array, *next*, of length $m+1$, whose last entry, $next[m]$, is used as a pointer to the location which contains the first (i.e., highest) element of $\mathcal{L}$ (or $next[m] = m$, in the case the list $\mathcal{L}$ is empty).

By a simple inspection, it is immediate to verify that the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm has an $\mathcal{O}(nm\lceil \alpha/w \rceil)$ worst-case running time and requires $\mathcal{O}(m\lceil \alpha/w \rceil)$-space. Moreover, by arguing as in [3], it can be shown that the running time of the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm is $\mathcal{O}(n)$ on the average (for a fixed α).

Notice that a slightly simpler variant of the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm could be obtained if we maintained into the array C the reverses of the bit masks $\mathcal{C}_i^{(0)}, \mathcal{C}_i^{(1)}, \ldots, \mathcal{C}_i^{(m-1)}$, rather than the bit masks themselves. In essence, this would involve replacing each left-shift by a right-shift. More precisely, we would have to replace the instruction at line 14 by the assignment $C[j] := C[j] \gg 1$ (thus avoiding to perform any operation prior to the shift) and the instructions at lines 9 and 19 by the assignments $C[j+1] := C[j+1] \,|\, \mathsf{10}^{\alpha}$ and $C[0] := C[0] \,|\, \mathsf{10}^{\alpha}$, respectively. Also, the condition in the **if**-statement of line 21 would need to be replaced by the condition "$(C[m-1] \,\&\, \mathsf{10}^{\alpha}) \neq \mathsf{0}^{\alpha+1}$". The above modifications would have the effect to slightly reduce the number of operations performed during each step of the computation.

Observe also that the last entry $C[m-1]$ of the array C is used by the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm only in the conditional test of line 21. Therefore we do not need to maintain it, since such a test could be implicitly performed during the execution of the **while-loop** of lines 7-17 as follows. If during the execution of the **while-loop** the variable j assumes the value $m-2$ (which means that position $m-2$ is in the list $\mathcal{L}$, i.e., $C[m-2]$ is nonnull), then we check whether $T[i] =_{\delta} P[m-1]$ and, if this is the case, the value i can be directly reported as the position of a (δ, α)-occurrence of the pattern P in the text T. Otherwise, if the variable j does not ever take the value $m-2$ during the execution of the **while-loop**, then the pattern P can have no (δ, α)-occurrence at position i in the text, and therefore, even in this case, the test at line 21 does not need to be checked. It turns out that the variation just outlined slightly improves the overall running time of the algorithm.

In the last variant of the (δ, α)-Sequential-Sampling algorithm, which we are going to describe (actually a variant of the (δ, α)-Sequential-Sampling-HBP algorithm), each matrix $\mathcal{M}_i$ is represented as a single bit mask of length $L = (\alpha+1)m$, obtained by concatenating the bit masks corresponding to the columns of $\mathcal{M}_i$ (i.e., the bit masks $\mathcal{C}_i^{(j)}$). More precisely, the following bit mask is used as a representation of the matrix $\mathcal{M}_i$, for $-1 \leq i < n$:[e]

$$\mathcal{B}_i = \mathcal{C}_i^{(m-1)} \mathcal{C}_i^{(m-2)} \ldots \mathcal{C}_i^{(0)} \,.$$

Assuming such a representation for the matrices $\mathcal{M}_i$ as single bit masks, the task is to find an efficient way to compute bit-parallely the bit mask $\mathcal{B}_i$ from the bit mask $\mathcal{B}_{i-1}$. (Notice that the initial bit mask $\mathcal{B}_{-1}$ is the null bit mask, i.e., $\mathcal{B}_{-1} = \mathsf{0}^L$.)

To begin with, let $\mathcal{X}_i^{(j)}$ be the bit mask of length $\alpha+1$ defined by

$$\mathcal{X}_i^{(j)} = \begin{cases} \mathsf{0}^{\alpha}\mathsf{1} & \text{if } T[i] =_{\delta} P[j] \text{ AND } (j = 0 \text{ OR } \mathcal{C}_{i-1}^{(j-1)} \neq \mathsf{0}^{\alpha+1}) \\ \mathsf{0}^{\alpha+1} & \text{otherwise}\,, \end{cases}$$

[e] Notice plainly that, once the bit mask $\mathcal{B}_i$ has been computed, we can check in constant time whether $P \trianglelefteq^i T$ holds by simply checking whether the $(\alpha+1)$-st bit of $\mathcal{B}_i$ is $\mathsf{1}$, i.e., if $\mathcal{B}_i[\alpha] = \mathsf{1}$, which corresponds to the condition that $\mathcal{B}_i \,\&\, U \neq \mathsf{0}^L$ where $U = \mathsf{0}^{\alpha}\mathsf{1}\mathsf{0}^{L-\alpha-1}$.

for $0 \leq j < m$, and let $\mathcal{X}_i = \mathcal{X}_i^{(m-1)}\mathcal{X}_i^{(m-2)} \dots \mathcal{X}_i^{(0)}$. Then, by (5) we have that $\mathcal{C}_i^{(j)} = ((\mathcal{C}_{i-1}^{(j)} \,\&\, \mathsf{01}^\alpha) \ll 1) \,|\, \mathcal{X}_i^{(j)}$, for $0 \leq i < n$ and $0 \leq j < m$, and therefore

$$\mathcal{B}_i = ((\mathcal{B}_{i-1} \,\&\, I) \ll 1) \,|\, \mathcal{X}_i\,, \tag{8}$$

for $0 \leq i < n$, where $I = (\mathsf{01}^\alpha)^m$. Thus, we need only to be able to compute effectively the bit mask $\mathcal{X}_i$ from the bit mask $\mathcal{B}_{i-1}$, which we do as follows.

For each symbol s of the alphabet Σ and each $0 \leq j < m$, let $\mathsf{b}_s^{(j)}$ be the bit value $\mathsf{1}$, if $s =_\delta P[j]$ holds, otherwise let $\mathsf{b}_s^{(j)}$ be the bit value $\mathsf{0}$. Also, let

$$\mathcal{H}(s) = \mathsf{0}^\alpha(\mathsf{b}_s^{(m-1)}\mathsf{0}^\alpha)(\mathsf{b}_s^{(m-2)}\mathsf{0}^\alpha)\dots(\mathsf{b}_s^{(1)}\mathsf{0}^\alpha)\mathsf{b}_s^{(0)}\,.$$

Furthermore, let $\mathsf{x}_i^{(j)}$ be the last bit of the bit mask $\mathcal{X}_i^{(j)}$ (i.e., $\mathsf{x}_i^{(j)} = \mathcal{X}_i^{(j)}[\alpha]$), for $0 \leq j < m$, so that we have

$$\mathcal{X}_i = \mathsf{0}^\alpha(\mathsf{x}_i^{(m-1)}\mathsf{0}^\alpha)(\mathsf{x}_i^{(m-2)}\mathsf{0}^\alpha)\dots(\mathsf{x}_i^{(1)}\mathsf{0}^\alpha)\mathsf{x}_i^{(0)}\,. \tag{10}$$

Then, we claim that

$$\mathsf{x}_i^{(0)} = \mathsf{b}_i^{(0)}\,, \tag{11}$$

and

$$\mathsf{x}_i^{(j)}\mathsf{0}^\alpha = (\mathsf{b}_i^{(j)}\mathsf{0}^\alpha) \,\&(((\mathcal{C}_{i-1}^{(j-1)} \,\&\, \mathsf{01}^\alpha) + \mathsf{01}^\alpha) \,|\, \mathcal{C}_{i-1}^{(j-1)})\,, \tag{12}$$

for $0 < j < m$, where we have written $\mathsf{b}_i^{(j)}$ in place of $\mathsf{b}_{T[i]}^{(j)}$ (just to simplify the notation). We need only to verify (12), since (11) is an immediate consequence of the definitions of $\mathsf{b}_i^{(0)}$ and $\mathcal{X}_i^{(0)}$. To do this, we begin by noting that the operation $\mathcal{C}_{i-1}^{(j-1)} \,\&\, \mathsf{01}^\alpha$ sets the first bit of $\mathcal{C}_{i-1}^{(j-1)}$ to $\mathsf{0}$, leaving unchanged the remaining bits. Thus, by performing the arithmetic addition of $\mathcal{C}_{i-1}^{(j-1)} \,\&\, \mathsf{01}^\alpha$ with $\mathsf{01}^\alpha$, we obtain a bit mask whose first bit is $\mathsf{0}$ if and only if the last α bits of $\mathcal{C}_{i-1}^{(j-1)}$ are all $\mathsf{0}$'s. Therefore, the bit mask $(((\mathcal{C}_{i-1}^{(j-1)} \,\&\, \mathsf{01}^\alpha) + \mathsf{01}^\alpha) \,|\, \mathcal{C}_{i-1}^{(j-1)})$ has its first bit equal to $\mathsf{0}$ if and only if $\mathcal{C}_{i-1}^{(j-1)}$ is null (i.e., if and only if $\mathcal{C}_{i-1}^{(j-1)} = \mathsf{0}^{\alpha+1}$). At this point (12) is an immediate consequence of the definitions of $\mathsf{b}_i^{(j)}$ and $\mathcal{X}_i^{(j)}$, and thus our claim is correct.

By (10), (11), (12), and by the definition of the function $\mathcal{H}$, we get

$$\mathcal{X}_i = (((\mathcal{W}_{i-1} \,\&\, F) \ll 1) \,|\, \mathsf{0}^{L-1}\mathsf{1}) \,\&\, \mathcal{H}(T[i])\,, \tag{13}$$

where we have put $F = \mathsf{01}^{L-1}$ and $\mathcal{W}_{i-1} = ((\mathcal{B}_{i-1} \,\&\, I) + I) \,|\, \mathcal{B}_{i-1}$.

Relations (13) and (8) provide the required recursive formulae for computing the bit mask $\mathcal{B}_i$ from the bit mask $\mathcal{B}_{i-1}$. The resulting algorithm, named (δ, α)-Sequential-Sampling-BP ((δ, α)-S-S-BP, for short), is reported in Figure 3 (on the left). It uses an array H, indexed by the symbols of the alphabet Σ, which is computed in such a way that $H[s] = \mathcal{H}(s)$, for each $s \in \Sigma$. Notice also that at the end of the execution of the **for-loop** of line 6 we have that *tmp1* $= (\mathsf{10}^\alpha)^m$ and *tmp2* $= (\mathsf{0}^\alpha\mathsf{1})^m$, and so the assignment of line 13 computes correctly the value of the bit mask I (i.e., $I = (\mathsf{01}^\alpha)^m$), and moreover $U = \mathsf{0}^\alpha\mathsf{10}^{L-\alpha-1}$ (cf. footnote e).

```
(δ, α)-S-S-BP(P, m, T, n, Σ, δ, α)
 1.  $L := (\alpha + 1)m$
 2.  for $s \in \Sigma$ do $H[s] := 0^L$
 3.  $\mathit{tmp1} := 0^{L-1}1 \ll (L - 1)$
 4.  $\mathit{tmp2} := 0^{L-1}1 \ll (L - \alpha - 1)$
 5.  $U := 0^{L-1}1$
 6.  for $j := 0$ to $m - 1$ do
 7.     for $s \in \Sigma \cap [P[j] - \delta \,..\, P[j] + \delta]$ do
 8.        $H[s] := H[s] \mid U$
 9.     if $j < m - 1$ then
10.        $\mathit{tmp1} := \mathit{tmp1} \mid (U \ll \alpha)$
11.        $\mathit{tmp2} := \mathit{tmp2} \mid U$
12.        $U := U \ll (\alpha + 1)$
13.  $I := \mathit{tmp1} - \mathit{tmp2}$
14.  $F := (0^{L-1}1 \ll (L - 1)) - 0^{L-1}1$
15.  $B := 0^L$
16.  for $i := 0$ to $n - 1$ do
17.     $W := ((B \,\&\, I) + I) \mid B$
18.     $X := (((W \,\&\, F) \ll 1) \mid 0^{L-1}1) \,\&\, H[T[i]]$
19.     $B := ((B \,\&\, I) \ll 1) \mid X$
20.     if $(B \,\&\, U) \neq 0^L$ then
21.        print($i$)
```

```
(δ, α)-S-S-BP⁺(P, m, T, n, Σ, δ, α)
 1.  $\ell := (\alpha + 1)(m - 1) + 1$
 2.  for $s \in \Sigma$ do $H[s] := 0^\ell$
 3.  $\mathit{tmp} := 0^\ell$
 4.  $U := 0^{\ell-1}1$
 5.  for $j := 0$ to $m - 1$ do
 6.     for $s \in \Sigma \cap [P[j] - \delta \,..\, P[j] + \delta]$ do
 7.        $H[s] := H[s] \mid U$
 8.     if $j < m - 1$ then
 9.        $\mathit{tmp} := \mathit{tmp} \mid U$
10.        $U := U \ll (\alpha + 1)$
11.  $J := U - \mathit{tmp}$
12.  $F := (0^{\ell-1}1 \ll (\ell - 1)) - 0^{\ell-1}1$
13.  $B := 0^\ell$
14.  for $i := 0$ to $n - 1$ do
15.     $B := (B \,\&\, F) \ll 1$
16.     $C := B \,\&\, J$
17.     $B := (((C + J) \mid B) \,\&\, H[T[i]]) \mid C$
18.     if $(B \,\&\, U) \neq 0^\ell$ then
19.        print($i$)
```

Fig. 3. The (δ, α)-Sequential-Sampling-BP algorithm (on the left) and the (δ, α)-Sequential-Sampling-BP$^+$ algorithm (on the right) for the δ-approximate string matching problem with α-bounded gaps.

It can easily be verified that space and time complexities of the (δ, α)-Sequential-Sampling-BP algorithm are $\mathcal{O}(\sigma\lceil(m\alpha)/w\rceil)$ and $\mathcal{O}((\sigma + n + m\delta)\lceil(m\alpha)/w\rceil)$, respectively. In particular, the algorithm has an $\mathcal{O}(n\lceil(m\alpha)/w\rceil)$ searching time, while preprocessing takes $\mathcal{O}((\sigma + m\delta)\lceil(m\alpha)/w\rceil)$-time.[f]

Let us make some remarks on the latter algorithm. To begin with, notice that if we replace, respectively, the instructions at lines 3, 4 and 14 of the algorithm (δ, α)-Sequential-Sampling-BP by the assignments $\mathit{tmp1} := 0^L$, $\mathit{tmp2} := 0^L$ and $F := (0^{L-1}1 \ll (L - \alpha - 1)) - 0^{L-1}1$, then the resulting algorithm still does the same work of the original one, except that the first α bits of the bit mask B (and of all the other bit masks) are always left unset (i.e., they remain 0's) during the computation;[g] but, since the conditional test of line 20 (i.e., the test whether $P \trianglelefteq^i T$) involves only the $(\alpha + 1)$-st bit of B, the modified algorithm solves the

[f] Notice that this estimation of the time complexity of the (δ, α)-Sequential-Sampling-BP algorithm algorithm is in agreement with our assumptions on the complexity of operations on bit masks stated in Section 2. However, in practice, more efficient implementations of the algorithm could be obtained. For instance, observe that for $0 \leq j < m$, when iteration j of the **for-loop** of line 6 of the (δ, α)-Sequential-Sampling-BP algorithm starts, we have $U = 0^\ell 1 \ll (j(\alpha + 1))$. Therefore, the assignments of lines 8, 10 and 11 could be implemented so as to take constant time, assuming the model in which a bit mask X is represented as a sequence of $\lceil |X|/w \rceil$ computer words, thus reducing the preprocessing time to $\mathcal{O}(\sigma\lceil(m\alpha)/w\rceil + m\delta)$.

[g] In fact, with such modifications, when the algorithm enters the **for-loop** of line 16, we have that $I = 0^{\alpha+1}(01^\alpha)^{m-1}$ and $F = 0^{\alpha+1}1^{L-\alpha-1}$ hold, as can be easily verified.

(δ, α)-matching problem as well. Thus, the first α bits of the bit masks used by the (δ, α)-Sequential-Sampling-BP algorithm can be dropped, and therefore the number of bits of these bit masks which need to be actually stored during the computation is $\ell = L - \alpha = (\alpha + 1)(m - 1) + 1$ (and hence, in particular, if $\ell \leq w$ all of these bit masks fit each in a single computer word). Observe also that for $I = \mathsf{0}^{\alpha+1}(\mathsf{0}\mathsf{1}^{\alpha})^{m-1}$ and $F = \mathsf{0}^{\alpha+1}\mathsf{1}^{L-\alpha-1}$ (cf. footnote g), the part of code of the (δ, α)-Sequential-Sampling-BP algorithm from line 15 up to line 21 turns out to be equivalent to the following one:

```
$B := \mathsf{0}^L$
for $i := 0$ to $n - 1$ do
   $B := (B \,\&\, F) \ll 1$
   $C := B \,\&\, J$
   $B := (((C + J) \,|\, B) \,\&\, H[T[i]]) \,|\, C$
   if $(B \,\&\, U) \neq \mathsf{0}^L$ then
      print($i$)
```

where $J = \mathsf{0}^{\alpha}(\mathsf{0}\mathsf{1}^{\alpha})^{m-1}\mathsf{1}$, as can be easily verified by very simple algebraic manipulations, thus further reducing the overall number of operations which need to be performed.

The above considerations translate into the variant of the (δ, α)-Sequential-Sampling-BP algorithm reported in Figure 3 (on the right), named (δ, α)-Sequential-Sampling-BP$^+$ ((δ, α)-S-S-BP$^+$, for short), which, although characterized by the same asymptotic space and time complexity of the original algorithm, turns out to be slightly more efficient in practice (see also footnote f).

Notice that at the end of the execution of the **for-loop** of line 5 of the (δ, α)-Sequential-Sampling-BP$^+$ algorithm, we have that $tmp = \mathsf{0}(\mathsf{0}^{\alpha}\mathsf{1})^{m-1}$ and $U = \mathsf{1}\mathsf{0}^{\ell-1}$, so that $U - tmp = (\mathsf{0}\mathsf{1}^{\alpha})^{m-1}\mathsf{1}$ (which is equal to the bit mask resulting from J by dropping its first α bits $\mathsf{0}$).

5. Experimental Results

In this section we report experimental data relative to an extensive comparison of our newly presented algorithms (δ, α)-Tuned-Sequential-Sampling-HBP and (δ, α)-Sequential-Sampling-BP$^+$, described in Section 4, and the algorithms SDP-simple, DA-mloga-bits, and (δ, α)-Shift-And, reviewed in Section 3, which are among the most efficient algorithms for the (δ, α)-matching problem.[h]

In particular, we have performed two main sets of experimental tests: the first one, the experimental set Es1, concerns the comparison of the algorithms (δ, α)-Tuned-Sequential-Sampling-HBP and SDP-simple, whereas the second one, the experimental set Es2, involves the algorithms (δ, α)-Tuned-Sequential-Sampling-HBP, (δ, α)-Sequential-Sampling-BP$^+$, DA-mloga-bits, and (δ, α)-Shift-And.

[h] We have also considered in our experimental tests the algorithm SDP-simple-compute-L_0, but, because of layout problems due to the dimensions of the tables and to the fact that it turned out to be always slower than the SDP-simple algorithm, we omitted to report its timings.

All algorithms have been implemented in the **C** programming language using the Borland C++ compiler, version 5.5, 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, with 512 MB of RAM, running Windows XP. In particular, they have been tested on three Randσ problems, for $\sigma = 50, 90, 130$, and on a real music text buffer. Each Randσ problem consisted in searching for a set of 150 random patterns of length $m = 6, 8, 10, 20, 30, 40, 60, 80, 100$ in a random text sequence of length $n = 5,242,880$, over a common alphabet of size σ. For each Randσ problem, the values of the approximation bound δ and of the gap bound α have been set to 1, 3, 5 and to 2, 5, 8, respectively. The running times of the algorithms have been averaged over all patterns. Concerning the tests on the real music text buffer, these have been performed on a fixed text sequence T of length $n = 2,982,507$ obtained by combining a set of various classical pieces in MIDI format, with an overall alphabet of 76 distinct symbols, i.e., the MIDI values of the notes of the pieces. For each m as above, we have randomly selected a set of 150 substrings of T of length m which subsequently have been searched for in T.

In the case of the experimental set Es2, the tests have been performed just as described above except that, this time, the algorithms involved in the comparison, i.e., (δ, α)-Tuned-Sequential-Sampling-HBP, (δ, α)-Sequential-Sampling-BP$^+$, DA-mloga-bits, and (δ, α)-Shift-And, have been tested using only short patterns and very small values of α. More precisely, just the pairs $(\alpha, m) \in \{1\} \times \{6, 8, 12, 16\} \cup \{2\} \times \{6, 8, 10\}$ have been used. The main reason behind this choice is that, for such pairs, each of the bit masks used by the last three algorithms fits into a single computer word, a condition which allows these algorithms to reach their best performances in practice.[i] The algorithm (δ, α)-Tuned-Sequential-Sampling-HBP has been included in this set of experimental tests mainly for comparing it with the algorithm DA-mloga-bits.

All running times in the tables are expressed in hundredths of second and, for each length of the pattern, the best result has been boldfaced. Moreover, the following abbreviations have been used to denote the algorithms: TSS-HBP for (δ, α)-Tuned-Sequential-Sampling-HBP; SS-BP for (δ, α)-Sequential-Sampling-BP$^+$; DA-NFA for (δ, α)-Shift-And; DA-CNFA for DA-mloga-bits; SDP-S for SDP-simple.

[i] However, as already remarked, notice that by allowing only small values of the gap bound α (e.g., $\alpha \leq 2$) is not a real limitation in many practical applications in music. In fact, searching with small gaps is enough to take into account various kinds of musical ornamentations, such as mordent, acciaccatura and appoggiatura, as well as many other common musical technicalities such as pedal notes.

Experimental results on a Real Music Problem (Es1)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 60$	$m = 80$	$m = 100$
TSS-HBP	$(1, 2)$	**2.36**	**2.40**	**2.44**	**2.50**	**2.54**	**2.30**	**2.39**	**2.44**	**2.48**
SDP-S	$(1, 2)$	3.50	3.38	3.79	3.75	3.87	3.42	3.70	3.81	3.73
TSS-HBP	$(1, 5)$	**4.14**	**4.33**	**4.95**	**4.93**	**5.17**	**4.35**	**4.85**	**4.85**	**4.63**
SDP-S	$(1, 5)$	4.93	5.15	5.95	6.03	6.16	5.39	5.58	5.83	5.71
TSS-HBP	$(1, 8)$	**5.65**	**6.36**	**7.69**	**7.93**	**8.13**	**6.67**	**7.45**	**7.61**	**7.33**
SDP-S	$(1, 8)$	6.15	6.79	8.06	8.76	8.65	7.58	8.17	8.52	8.07
TSS-HBP	$(3, 2)$	**5.11**	**4.78**	**5.27**	**5.16**	**5.90**	**4.87**	**5.53**	**4.97**	**5.57**
SDP-S	$(3, 2)$	6.60	6.27	7.00	6.81	7.38	6.40	7.01	6.77	7.06
TSS-HBP	$(3, 5)$	**9.57**	**10.00**	**12.40**	**12.96**	**14.61**	**12.68**	**13.42**	**12.59**	**13.49**
SDP-S	$(3, 5)$	10.59	10.73	12.92	14.52	16.32	14.35	15.27	14.12	15.23
TSS-HBP	$(3, 8)$	**11.13**	**12.63**	**16.59**	**20.43**	**24.57**	**22.56**	**24.19**	**21.83**	**23.60**
SDP-S	$(3, 8)$	12.73	14.20	18.11	23.41	28.91	27.10	28.86	26.04	28.49
TSS-HBP	$(5, 2)$	**9.03**	**9.05**	**10.54**	**10.58**	**15.46**	**18.78**	**18.98**	**18.88**	**21.63**
SDP-S	$(5, 2)$	10.39	10.49	11.68	12.44	18.66	22.22	22.60	22.83	25.07
TSS-HBP	$(5, 5)$	**13.14**	**15.02**	**19.46**	**23.73**	**24.83**	**25.06**	**51.78**	**28.93**	**37.44**
SDP-S	$(5, 5)$	15.85	17.91	22.64	28.41	30.73	31.15	65.30	36.76	47.81
TSS-HBP	$(5, 8)$	**12.94**	**15.92**	**21.29**	**30.10**	**36.26**	**36.67**	**48.03**	**55.14**	**52.40**
SDP-S	$(5, 8)$	17.59	20.36	26.91	38.40	46.99	48.06	64.66	76.90	75.33

Experimental results on a Rand50 problem (Es1)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 60$	$m = 80$	$m = 100$
TSS-HBP	$(1, 2)$	**3.02**	**2.92**	**2.98**	**2.84**	**2.94**	**2.94**	**2.90**	**2.98**	**2.96**
SDP-S	$(1, 2)$	4.60	4.78	4.58	4.69	4.75	4.73	4.77	4.65	4.77
TSS-HBP	$(1, 5)$	**4.33**	**4.17**	**4.35**	**4.29**	**4.35**	**4.27**	**4.32**	**4.25**	**4.35**
SDP-S	$(1, 5)$	6.19	6.11	6.25	6.21	6.17	6.19	6.01	6.09	6.11
TSS-HBP	$(1, 8)$	**5.65**	**5.59**	**5.79**	**5.89**	**5.89**	**5.69**	**5.73**	**5.68**	**5.79**
SDP-S	$(1, 8)$	7.43	7.65	7.79	7.61	7.71	7.67	7.59	7.67	7.67
TSS-HBP	$(3, 2)$	**5.89**	**5.81**	**5.79**	**5.95**	**5.94**	**5.91**	**5.84**	**5.84**	**5.71**
SDP-S	$(3, 2)$	8.85	8.73	8.85	8.83	8.83	8.62	8.79	8.85	8.79
TSS-HBP	$(3, 5)$	**12.24**	**12.96**	**13.31**	**13.84**	**13.75**	**13.47**	**13.71**	**13.95**	**13.58**
SDP-S	$(3, 5)$	13.10	14.63	15.56	16.27	16.09	15.80	16.28	16.28	16.11
TSS-HBP	$(3, 8)$	17.38	20.48	22.83	**26.10**	**26.29**	**25.95**	**26.46**	**51.53**	**50.98**
SDP-S	$(3, 8)$	**17.22**	**19.63**	**22.61**	29.15	29.75	29.28	30.32	59.02	58.44
TSS-HBP	$(5, 2)$	**11.49**	**11.79**	**11.68**	**11.79**	**11.99**	**11.94**	**22.87**	**22.27**	**22.07**
SDP-S	$(5, 2)$	14.11	14.82	15.28	15.12	15.28	15.51	29.34	29.10	28.77
TSS-HBP	$(5, 5)$	**22.91**	**27.01**	**29.66**	**35.88**	**37.35**	**37.61**	**39.97**	**64.85**	**36.72**
SDP-S	$(5, 5)$	24.04	27.65	30.35	40.83	44.26	45.15	47.06	77.20	43.95
TSS-HBP	$(5, 8)$	**26.53**	**34.68**	**43.38**	**121.11**	**175.83**	**212.14**	**257.04**	**329.08**	**320.51**
SDP-S	$(5, 8)$	29.82	37.62	46.58	135.99	205.92	254.88	318.26	429.03	422.84

Experimental results on a Rand90 problem (Es1)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 60$	$m = 80$	$m = 100$
TSS-HBP	$(1, 2)$	**2.27**	**2.27**	**2.40**	**2.42**	**2.40**	**2.38**	**2.26**	**2.30**	**2.36**
SDP-S	$(1, 2)$	3.70	3.78	3.64	3.71	3.71	3.71	3.77	3.69	3.67
TSS-HBP	$(1, 5)$	**2.94**	**3.03**	**3.11**	**3.00**	**3.05**	**2.93**	**2.96**	**2.94**	**2.97**
SDP-S	$(1, 5)$	4.42	4.31	4.27	4.43	4.25	4.37	4.39	4.43	4.36
TSS-HBP	$(1, 8)$	**3.57**	**3.21**	**3.61**	**3.36**	**3.35**	**3.43**	**3.41**	**3.57**	**3.45**
SDP-S	$(1, 8)$	4.97	4.81	4.87	5.00	4.97	4.87	4.93	4.97	4.87
TSS-HBP	$(3, 2)$	**3.41**	**3.51**	**3.55**	**3.39**	**3.47**	**3.49**	**4.93**	**6.53**	**6.66**
SDP-S	$(3, 2)$	5.40	5.37	5.40	5.39	5.42	5.40	7.47	10.15	10.17
TSS-HBP	$(3, 5)$	**5.59**	**5.55**	**5.71**	**5.57**	**5.81**	**5.71**	**5.63**	**5.61**	**5.61**
SDP-S	$(3, 5)$	7.66	7.63	7.92	7.62	7.74	7.64	7.68	7.56	7.66
TSS-HBP	$(3, 8)$	**8.06**	**8.25**	**8.33**	**8.32**	**8.51**	**8.45**	**8.28**	**8.29**	**8.15**
SDP-S	$(3, 8)$	9.59	10.17	10.56	10.28	10.30	10.19	10.24	10.24	10.31
TSS-HBP	$(5, 2)$	**7.11**	**7.01**	**9.86**	**9.66**	**9.28**	**9.68**	**9.63**	**9.72**	**9.75**
SDP-S	$(5, 2)$	9.55	9.57	14.81	14.63	14.36	14.65	14.60	14.77	14.68
TSS-HBP	$(5, 5)$	**10.04**	**10.54**	**10.81**	**10.79**	**10.45**	**10.85**	**10.77**	**10.93**	**10.82**
SDP-S	$(5, 5)$	11.43	12.43	13.28	13.17	12.77	13.15	12.99	13.29	13.39
TSS-HBP	$(5, 8)$	**14.48**	16.77	**17.86**	**19.45**	**19.39**	**19.80**	**19.57**	**19.85**	**19.86**
SDP-S	$(5, 8)$	14.53	**16.72**	18.82	21.70	21.55	21.81	21.71	21.99	22.06

Experimental results on a Rand130 problem (Es1)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 60$	$m = 80$	$m = 100$
TSS-HBP	$(1, 2)$	**2.16**	**2.12**	**2.14**	**2.12**	**2.14**	**2.10**	**2.14**	**2.14**	**2.15**
SDP-S	$(1, 2)$	3.37	3.30	3.44	3.53	3.41	3.38	3.47	3.33	3.34
TSS-HBP	$(1, 5)$	**2.61**	**2.56**	**2.52**	**2.60**	**2.62**	**2.44**	**2.52**	**2.50**	**2.54**
SDP-S	$(1, 5)$	3.72	3.83	3.74	3.66	3.70	3.78	3.81	3.83	3.75
TSS-HBP	$(1, 8)$	**2.92**	**2.78**	**2.96**	**2.78**	**2.88**	**2.80**	**2.89**	**2.82**	**2.80**
SDP-S	$(1, 8)$	4.15	4.33	4.09	4.12	4.06	4.08	4.07	4.11	4.09
TSS-HBP	$(3, 2)$	**2.83**	**2.95**	**2.83**	**2.84**	**2.83**	**2.84**	**2.81**	**2.88**	**2.80**
SDP-S	$(3, 2)$	4.42	4.57	4.59	4.52	4.62	4.59	4.52	4.39	4.58
TSS-HBP	$(3, 5)$	**4.07**	**4.04**	**4.15**	**4.04**	**3.94**	**4.05**	**4.01**	**4.07**	**7.65**
SDP-S	$(3, 5)$	5.66	5.66	5.66	5.68	5.79	5.77	5.72	5.78	10.76
TSS-HBP	$(3, 8)$	**5.08**	**4.96**	**5.23**	**5.17**	**5.13**	**5.11**	**5.22**	**5.04**	**5.19**
SDP-S	$(3, 8)$	6.87	6.95	7.04	7.01	6.99	6.94	6.96	6.89	6.99
TSS-HBP	$(5, 2)$	**3.78**	**3.78**	**3.84**	**3.68**	**3.72**	**6.14**	**7.06**	**7.05**	**6.91**
SDP-S	$(5, 2)$	5.79	5.74	5.93	5.71	5.66	9.69	10.91	10.96	10.77
TSS-HBP	$(5, 5)$	**9.14**	**9.39**	**9.78**	**9.53**	**9.77**	**9.56**	**9.82**	**9.71**	**9.46**
SDP-S	$(5, 5)$	10.39	11.27	11.52	11.48	11.52	11.39	11.40	11.50	11.31
TSS-HBP	$(5, 8)$	**10.23**	**10.15**	**12.34**	**11.96**	**11.82**	**12.00**	**11.97**	**11.94**	**11.69**
SDP-S	$(5, 8)$	12.20	12.29	16.42	15.68	15.88	15.78	15.96	15.94	15.72

Experimental results on a Real Music Problem (Es2)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 12$	$m = 16$		ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$
TSS-HBP	$(1, 1)$	2.00	1.88	2.06	1.98		TSS-HBP	$(1, 2)$	2.36	2.27	2.42
SS-BP	$(1, 1)$	**1.00**	**0.84**	**0.94**	**0.88**		SS-BP	$(1, 2)$	**0.92**	**0.90**	**0.82**
DA-NFA	$(1, 1)$	1.02	1.00	1.00	1.00		DA-NFA	$(1, 2)$	1.02	1.00	1.05
DA-CNFA	$(1, 1)$	9.15	9.14	9.12	9.17		DA-CNFA	$(1, 2)$	9.22	9.19	9.09
TSS-HBP	$(3, 1)$	3.26	3.28	3.41	3.48		TSS-HBP	$(3, 2)$	5.14	4.58	4.99
SS-BP	$(3, 1)$	**0.94**	**0.94**	**0.88**	**0.98**		SS-BP	$(3, 2)$	**0.92**	**0.94**	**0.94**
DA-NFA	$(3, 1)$	1.06	1.04	1.02	1.02		DA-NFA	$(3, 2)$	1.16	1.14	1.06
DA-CNFA	$(3, 1)$	9.34	9.35	9.49	9.20		DA-CNFA	$(3, 2)$	9.40	9.25	9.30
TSS-HBP	$(5, 1)$	5.13	5.35	5.69	5.28		TSS-HBP	$(5, 2)$	8.70	8.87	9.91
SS-BP	$(5, 1)$	1.08	**0.92**	**0.88**	**0.84**		SS-BP	$(5, 2)$	**1.18**	**1.06**	**1.02**
DA-NFA	$(5, 1)$	**1.07**	1.06	1.06	1.06		DA-NFA	$(5, 2)$	1.20	1.08	1.06
DA-CNFA	$(5, 1)$	9.38	9.25	9.25	9.29		DA-CNFA	$(5, 2)$	9.54	9.44	9.28

Experimental results on a Rand50 problem (Es2)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 12$	$m = 16$		ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$
TSS-HBP	$(1, 1)$	2.68	2.76	2.82	2.78		TSS-HBP	$(1, 2)$	3.05	2.98	2.92
SS-BP	$(1, 1)$	**1.68**	**1.68**	**1.52**	**1.58**		SS-BP	$(1, 2)$	**1.72**	**1.68**	1.78
DA-NFA	$(1, 1)$	1.94	1.70	1.92	1.76		DA-NFA	$(1, 2)$	1.90	1.81	**1.70**
DA-CNFA	$(1, 1)$	16.07	15.88	15.95	15.92		DA-CNFA	$(1, 2)$	16.07	16.06	15.95
TSS-HBP	$(3, 1)$	4.52	4.46	4.60	4.56		TSS-HBP	$(3, 2)$	5.95	5.81	5.73
SS-BP	$(3, 1)$	**1.74**	**1.64**	**1.67**	1.73		SS-BP	$(3, 2)$	**1.79**	**1.78**	**1.78**
DA-NFA	$(3, 1)$	1.92	1.89	1.74	**1.72**		DA-NFA	$(3, 2)$	1.84	1.80	1.92
DA-CNFA	$(3, 1)$	16.68	16.28	16.36	16.34		DA-CNFA	$(3, 2)$	16.53	16.33	16.24
TSS-HBP	$(5, 1)$	10.37	13.13	13.55	13.91		TSS-HBP	$(5, 2)$	11.35	11.57	11.57
SS-BP	$(5, 1)$	**2.46**	**3.44**	**3.36**	**3.22**		SS-BP	$(5, 2)$	**1.82**	**1.74**	**1.68**
DA-NFA	$(5, 1)$	2.70	3.56	3.46	3.50		DA-NFA	$(5, 2)$	1.94	1.86	1.84
DA-CNFA	$(5, 1)$	23.32	31.23	31.28	31.15		DA-CNFA	$(5, 2)$	16.58	16.35	16.31

Experimental results on a Rand90 problem (Es2)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 12$	$m = 16$		ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$
TSS-HBP	$(1, 1)$	2.24	2.24	2.28	2.30		TSS-HBP	$(1, 2)$	2.38	2.30	2.36
SS-BP	$(1, 1)$	**1.68**	**1.71**	**1.68**	**1.68**		SS-BP	$(1, 2)$	**1.68**	**1.70**	**1.68**
DA-NFA	$(1, 1)$	1.84	1.78	1.76	1.80		DA-NFA	$(1, 2)$	2.02	1.78	1.84
DA-CNFA	$(1, 1)$	16.12	15.92	16.02	15.96		DA-CNFA	$(1, 2)$	16.14	15.94	15.88
TSS-HBP	$(3, 1)$	3.16	3.03	3.03	2.97		TSS-HBP	$(3, 2)$	3.59	3.29	3.48
SS-BP	$(3, 1)$	1.79	**1.78**	**1.74**	1.84		SS-BP	$(3, 2)$	**1.76**	**1.83**	**1.72**
DA-NFA	$(3, 1)$	**1.76**	1.84	1.82	**1.74**		DA-NFA	$(3, 2)$	1.89	1.88	1.84
DA-CNFA	$(3, 1)$	16.57	16.22	16.39	16.30		DA-CNFA	$(3, 2)$	16.56	16.33	16.38
TSS-HBP	$(5, 1)$	3.99	4.09	4.00	3.97		TSS-HBP	$(5, 2)$	5.26	4.97	4.96
SS-BP	$(5, 1)$	**1.86**	**1.68**	1.77	**1.68**		SS-BP	$(5, 2)$	**1.72**	**1.76**	**1.76**
DA-NFA	$(5, 1)$	**1.86**	1.88	**1.76**	1.88		DA-NFA	$(5, 2)$	1.84	1.78	1.92
DA-CNFA	$(5, 1)$	16.51	16.31	16.30	16.34		DA-CNFA	$(5, 2)$	16.47	16.27	16.31

Experimental results on a Rand130 problem (Es2)

ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 12$	$m = 16$		ALGS	(δ, α)	$m = 6$	$m = 8$	$m = 10$
TSS-HBP	$(1, 1)$	2.30	2.07	2.20	2.02		TSS-HBP	$(1, 2)$	2.26	2.16	2.14
SS-BP	$(1, 1)$	**1.58**	**1.72**	**1.62**	**1.62**		SS-BP	$(1, 2)$	**1.52**	**1.70**	**1.66**
DA-NFA	$(1, 1)$	1.82	**1.72**	1.88	1.88		DA-NFA	$(1, 2)$	1.98	**1.70**	1.82
DA-CNFA	$(1, 1)$	16.12	15.95	15.92	15.96		DA-CNFA	$(1, 2)$	16.12	16.02	16.00
TSS-HBP	$(3, 1)$	2.73	2.85	2.61	2.62		TSS-HBP	$(3, 2)$	2.97	2.89	2.85
SS-BP	$(3, 1)$	**1.71**	**1.42**	**1.57**	**1.74**		SS-BP	$(3, 2)$	**1.65**	**1.73**	**1.70**
DA-NFA	$(3, 1)$	1.88	1.92	1.98	1.94		DA-NFA	$(3, 2)$	1.98	1.84	1.84
DA-CNFA	$(3, 1)$	16.44	16.32	16.32	16.49		DA-CNFA	$(3, 2)$	16.45	16.30	16.34
TSS-HBP	$(5, 1)$	3.15	3.20	5.51	6.14		TSS-HBP	$(5, 2)$	3.76	3.72	3.80
SS-BP	$(5, 1)$	**1.75**	**1.75**	**2.97**	**3.30**		SS-BP	$(5, 2)$	**1.79**	**1.69**	**1.49**
DA-NFA	$(5, 1)$	1.86	1.82	3.09	3.62		DA-NFA	$(5, 2)$	1.84	1.78	1.82
DA-CNFA	$(5, 1)$	16.48	16.29	27.50	31.15		DA-CNFA	$(5, 2)$	16.52	16.30	16.34

¿From the experimental results it turns out that our algorithms (δ, α)-Tuned-Sequential-Sampling-HBP and (δ, α)-Sequential-Sampling-BP$^+$ are very efficient in practice. In the case of very short patterns and very small values of α (cf. the experimental set Es2), the algorithm (δ, α)-Sequential-Sampling-BP$^+$ is in general the fastest one, and beats also the automaton based algorithm (δ, α)-Shift-And. Moreover, it is about 8-9 times faster than DA-mloga-bits. Notice also that the algorithm (δ, α)-Tuned-Sequential-Sampling-HBP is always faster than DA-mloga-bits.

In the more general case of patterns of very varied lengths (cf. the experimental set Es1), the algorithm (δ, α)-Tuned-Sequential-Sampling-HBP outperforms almost always the very efficient SDP-simple; very rarely SDP-simple wins against (δ, α)-Tuned-Sequential-Sampling-HBP (just in the 0.47 per cent of the cases, with very short patterns). However, we recall that, in the worst case, the (δ, α)-Tuned-Sequential-Sampling-HBP algorithm requires only $\mathcal{O}(m\lceil \alpha/w \rceil)$ extra space, whereas the SDP-simple algorithm uses $\mathcal{O}(n)$ extra space.

6. Conclusions

We have presented some efficient practical algorithms for the δ-approximate string matching problem with α-bounded gaps, which have important applications in music information retrieval. Despite their non-optimal asymptotic behavior, our algorithms perform very well in practice and, in particular, one of them wins against the fastest existing algorithms in most practical cases.

Acknowledgments

We thank K. Fredriksson and S. Grabowski for having provided us with the **C** source code of their algorithms SDP-simple, DA-mloga-bits, and SDP-simple-compute-L_0, which we have used in our tests.

Appendix A. Handling classes of characters and bounded-size gaps

The algorithms presented in Section 4 can easily be adapted to solve also other variants of the approximate string matching problem with gaps, other than (δ, α)-matching. This is the case, for instance, of the approximate string matching problem with classes of characters and bounded-size gaps [9], which has important application in computational biology (see also [10]).

In this section we explain how to modify the algorithms (δ, α)-Tuned-Sequential-Sampling-HBP and (δ, α)-Sequential-Sampling-BP in order to solve the latter problem. (The modifications relative to the other two algorithms, namely the algorithms (δ, α)-Sequential-Sampling-HBP and (δ, α)-Sequential-Sampling-BP$^+$, follow along similar lines and are omitted for brevity.)

The approximate string matching problem with classes of characters and bounded-size gaps generalizes the (δ, α)-matching problem with respect to two aspects. Firstly, the symbols of the pattern P are classes of characters, rather than single characters, so that the element $P[j]$ of P will have an approximate matching at a given position i of a text T if $T[i]$ is a member of the class $P[j]$. Secondly, gaps of different (minimum and maximum) sizes are allowed to occur between any two consecutive positions of the approximate matchings of the pattern in the text.

In more details, the approximate string matching problem with classes of characters and bounded-size gaps is defined as follows. A PATTERN WITH CLASSES OF CHARACTERS AND BOUNDED-SIZE GAPS (or CBG, using the terminology of [9]) is a triple (P, α^*, β^*), where P is a string of a given length m, whose symbols are classes of characters drawn from some fixed alphabet Σ (i.e., $P[j] \subseteq \Sigma$, for $0 \leq j < m$), and $\alpha^* = (\alpha_0, \alpha_1, \ldots, \alpha_{m-2})$ and $\beta^* = (\beta_0, \beta_1, \ldots, \beta_{m-2})$ are sequences of nonnegative integer numbers such that $\beta_i \leq \alpha_i$, for $0 \leq i < m-1$. The meaning is that a gap of at most α_i symbols, but not less than β_i symbols, must occur between any two consecutive approximate matchings of the symbols $P[i]$ and $P[i+1]$ of the pattern P in the text T, when the pattern P is searched within T. More precisely, given a CBG (P, α^*, β^*) of length m (i.e., $|P| = m$) and a text T of length n, in the APPROXIMATE STRING MATCHING PROBLEM WITH CLASSES OF CHARACTERS AND BOUNDED-SIZE GAPS one is interested in finding those positions i of the text T, with $0 \leq i < n$, such that there exists a sequence of indices $(i_0, i_1, \ldots, i_{m-1})$ satisfying the following conditions:

(1) $0 \leq i_0 < i_1 < \cdots < i_{m-1} = i$,
(2) $T[i_j] \in P[j]$, for $0 \leq j < m$, and
(3) $\beta_{h-1} + 1 \leq i_h - i_{h-1} \leq \alpha_{h-1} + 1$, for $0 < h < m$, provided that $m > 1$.

In such a case, the sequence $(i_0, i_1, \ldots, i_{m-1})$ is called an APPROXIMATE OCCURRENCE OF (P, α^*, β^*) AT POSITION i IN T (or also, an APPROXIMATE OCCURRENCE WITH BOUNDED-SIZE GAPS OF P AT POSITION i IN T.)

To begin with, notice that handling classes of characters is straightforward: we have just to replace any test of the form "$T[i] =_\delta P[j]$", with the test "$T[i] \in P[j]$", in the case of the algorithm (δ, α)-Tuned-Sequential-Sampling-HBP; similarly, we have to replace the condition "$s \in \Sigma \cap [P[j] - \delta \mathinner{..} P[j] + \delta]$" with the condition "$s \in P[j]$", in the case of the algorithm (δ, α)-Sequential-Sampling-BP.

To handle gaps of different sizes between consecutive symbols of the pattern, our algorithms require a little bit more involved adjustments.

In the case of the algorithm (δ, α)-Tuned-Sequential-Sampling-HBP (cf. Figure 2 (on the right)), basically we have to modify the way in which the array C is computed. Specifically, the array C has to be computed in such a way that, for $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, m-1$, at the end of the $(i+1)$-st step of the computation, the entry $C[j]$ contains the bit mask of length $\alpha_j + 1$ whose $(k+1)$-st bit, from right to left, is 1 or 0 according to whether the prefix P_j of the pattern P has an approximate occurrence with bounded-size gaps at position $i - k$ in the text T, for $k = 0, 1, \ldots, \alpha_j$.[j] Thus, when step i of the computation has been completed, we can check whether the pattern has an approximate occurrence with bounded-size gaps at position i in T by simply checking whether the first bit (from right to left) of the bit mask contained in $C[m-1]$ is a 1. (Notice that this generalizes the way in which the array C is used in the original (δ, α)-Tuned-Sequential-Sampling-HBP algorithm.) This can be done as follows.

First of all, we have to initialize each entry of the array C to a suitable null bit mask; more precisely, C must be initialized in such a way that $C[j] = 0^{\alpha_j+1}$, for $0 \leq j < m$. Then, we have to replace the condition of the **if**-statement of line 8, by the following one:

$$j < m-1 \quad \text{AND} \quad T[i] \in P[j+1] \quad \text{AND} \quad (C[j] \,\&\, G[j]) \neq 0^{\alpha_j+1}\,,$$

where G is a precomputed array of length m, of bit masks, such that $G[j] = 1^{\alpha_j+1-\beta_j}0^{\beta_j}$, for $0 \leq j < m$. In fact, the condition that $(C[j] \,\&\, G[j]) \neq 0^{\alpha_j+1}$ means that the prefix P_j of the pattern P has an approximate occurrence with bounded-size gaps in at least one of the positions $i - \alpha_j - 1, \ldots, i - \beta_j - 1$ of the text T, so that we can extend it to an occurrence of the prefix P_{j+1} at position i of T (provided that $T[i] \in P[j+1]$). Notice that the bit mask $G[j]$ can be computed, e.g., as

$$G[j] = ((0^{\alpha_j}1 \ll \alpha_j) - (0^{\alpha_j}1 \ll \beta_j)) \,|\, (0^{\alpha_j}1 \ll \alpha_j)\,,$$

for $0 \leq j < m$.

Next, we have to replace the instruction at line 9 by the assignment $C[j+1] := C[j+1] \,|\, 0^{\alpha_{j+1}}1$ (whose meaning is that we have extended an occurrence of P_j to an occurrence of P_{j+1} at position i of the text, as above), and moreover the instruction at line 14 by the assignment $C[j] := (C[j] \,\&\, I[j]) \ll 1$, where (this time) I is a precomputed array of length m of bit masks, such that $I[j] = 01^{\alpha_j}$, for $0 \leq j < m$. This allows to update correctly the bit mask $C[j]$ after having used it.

Finally, other modifications of the algorithm (δ, α)-Tuned-Sequential-Sampling-HBP involve lines 15, 19, and 21. Specifically, we have to replace the instruction at line 19 by the assignment $C[0] := C[0] \,|\, 0^{\alpha_0}1$ and the conditions of the **if**-statements

[j] Here, α_{m-1} denotes just a fictitious value; it may be any nonnegative integer number (e.g., $\alpha_{m-1} = 0$). We assume that $\alpha_{m-1} \geq 0$ is fixed throughout the section. Similarly, we assume that β_{m-1} is a fixed nonnegative integer number such that $\beta_{m-1} \leq \alpha_{m-1}$.

```
CBG-T-S-S-HBP($P$, $m$, $T$, $n$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$)
1.  for $j := 0$ to $m - 1$ do
2.     $C[j] := 0^{\boldsymbol{\alpha}_j+1}$
3.     $I[j] := 01^{\boldsymbol{\alpha}_j}$
4.     $tmp1 := 0^{\boldsymbol{\alpha}_j}1 \ll \boldsymbol{\alpha}_j$
5.     $tmp2 := 0^{\boldsymbol{\alpha}_j}1 \ll \boldsymbol{\beta}_j$
6.     $G[j] := (tmp1 - tmp2) \mid tmp1$
7.  $next[0] := next[m] := m$
8.  for $i := 0$ to $n - 1$ do
9.     $p := m$
10.    $j := next[p]$
11.    while $j < m$ do
12.       if $j < m - 1$ AND
13.          $T[i] \in P[j + 1]$ AND
14.          $(C[j] \,\&\, G[j]) \neq 0^{\boldsymbol{\alpha}_j+1}$
15.       then
16.          $C[j + 1] := C[j + 1] \mid 0^{\boldsymbol{\alpha}_{j+1}}1$
17.          if $p > j + 1$ then
18.             $next[p] := j + 1$
19.             $next[j + 1] := j$
20.             $p := j + 1$
21.       $C[j] := (C[j] \,\&\, I[j]) \ll 1$
22.       if $C[j] = 0^{\boldsymbol{\alpha}_j+1}$ then
23.          $next[p] := next[j]$
24.       else $p := j$
25.       $j := next[p]$
26.    if $T[i] \in P[0]$ then
27.       $C[0] := C[0] \mid 0^{\boldsymbol{\alpha}_0}1$
28.       if $p > 0$ then $next[p] := 0$
29.    if $(C[m - 1] \,\&\, 0^{\boldsymbol{\alpha}_{m-1}}1) \neq 0^{\boldsymbol{\alpha}_{m-1}+1}$
30.    then print($i$)
```

```
CBG-S-S-BP($P$, $m$, $T$, $n$, $\Sigma$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$)
1.  $L := \sum_{i=0}^{m-1}(\boldsymbol{\alpha}_i + 1)$
2.  for $s \in \Sigma$ do $H[s] := 0^L$
3.  $tmp1 := 0^{L-1}1 \ll (L - 1)$
4.  $tmp2 := 0^{L-1}1 \ll (L - \boldsymbol{\alpha}_{m-1} - 1)$
5.  $tmp3 := 0^{L-1}1 \ll (L - \boldsymbol{\alpha}_{m-1} + \boldsymbol{\beta}_{m-1} - 1)$
6.  $U := 0^{L-1}1$
7.  for $j := 0$ to $m - 1$ do
8.     for $s \in P[j]$ do
9.        $H[s] := H[s] \mid U$
10.    if $j < m - 1$ then
11.       $tmp1 := tmp1 \mid (U \ll \boldsymbol{\alpha}_j)$
12.       $tmp2 := tmp2 \mid U$
13.       $tmp3 := tmp3 \mid (U \ll \boldsymbol{\beta}_j)$
14.       $U := U \ll (\boldsymbol{\alpha}_j + 1)$
15. $I := tmp1 - tmp2$
16. $K := tmp1 - tmp3$
17. $F := (0^{L-1}1 \ll (L - 1)) - 0^{L-1}1$
18. $B := 0^L$
19. for $i := 0$ to $n - 1$ do
20.    $W := ((B \,\&\, K) + K) \mid B$
21.    $X := (((W \,\&\, F) \ll 1) \mid 0^{L-1}1) \,\&\, H[T[i]]$
22.    $B := ((B \,\&\, I) \ll 1) \mid X$
23.    if $(B \,\&\, U) \neq 0^L$ then
24.       print($i$)
```

Fig. 4. The CBG-Tuned-Sequential-Sampling-HBP algorithm (on the left) and the CBG-Sequential-Sampling-BP algorithm (on the right) for the approximate string matching problem with classes of characters and bounded-size gaps. It is assumed that $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are arrays of length m such that $\boldsymbol{\alpha}[j] = \alpha_j$ and $\boldsymbol{\beta}[j] = \beta_j$, for $0 \leq j < m$. Moreover, in order to save space, the generic component of the array $\boldsymbol{\alpha}$ is denoted as $\boldsymbol{\alpha}_j$ rather than as $\boldsymbol{\alpha}[j]$, and similarly for the array $\boldsymbol{\beta}$.

of lines 15 and 21 by "$C[j] = 0^{\alpha_j+1}$" and "$C[m-1] \,\&\, 0^{\alpha_{m-1}}1 \neq 0^{\alpha_{m-1}+1}$", respectively.

The modified algorithm is called CBG-Tuned-Sequential-Sampling-HBP (CBG-T-S-S-HBP, for short) and is reported in Figure 4 (on the left).

Plainly, the worst case running time of the algorithm CBG-Tuned-Sequential-Sampling-HBP is $\mathcal{O}(nm(\lceil \alpha/w \rceil + \tau))$, where $\alpha = \max(\alpha_0, \ldots, \alpha_{m-1})$ and τ is an upper bound to the time required by the membership tests of lines 13 and 26. If σ is the size of the alphabet Σ of the pattern and text, then $\tau = \mathcal{O}(\sigma)$, since each class $P[h]$ contains no more than σ characters, for $0 \leq h < m$. Notice, however, that if we use an additional Boolean table E of dimensions $\sigma \times m$ such that $E[s, h] =$ "$s \in P[h]$", for $s \in \Sigma$ and $0 \leq h < m$, then the conditional tests of lines 13 and 26 can be done in constant time (i.e., $\tau = \mathcal{O}(1)$), which yields a worst case searching time of the algorithm of $\mathcal{O}(nm\lceil\alpha/w\rceil)$ (but more time would be needed in the preprocessing phase to compute E).

Concerning the algorithm (δ,α)-Sequential-Sampling-BP (cf. Figure 3 (on the left)), searching for patterns with bounded-size gaps involves the following modifications. The basic idea is to represent the array C used by the CBG-Tuned-Sequential-Sampling-HBP algorithm as a single bit mask B obtained by concatenating (from left to right) the bit masks contained in C, which is similar to the way in which the original (δ,α)-Sequential-Sampling-BP algorithm uses its bit mask B. Notice that, in this case, the bit mask B will be a bit mask of length equal to

$$L = (\alpha_0 + 1) + \cdots + (\alpha_{m-2} + 1) + (\alpha_{m-1} + 1)\,,$$

i.e., the sum of the lengths of the bit masks contained in the array C.

Three main modifications are in order; these concern, respectively, the computation of the array H and the assignment instructions at lines 17 and 19. More precisely, the array H has to be computed in such a way that, for each symbol $s \in \Sigma$, $H[s]$ contains the bit mask $(0^{\alpha_{m-1}}\mathsf{b}_{m-1})\cdots(0^{\alpha_1}\mathsf{b}_1)(0^{\alpha_0}\mathsf{b}_0)$ where the bit b_j is $\mathsf{1}$ or $\mathsf{0}$ according to whether $s \in P[j]$ or not, for $0 \leq j < m$. This can be accomplished by simply replacing the instruction at line 12 by the assignment $U := U \ll (\alpha_j + 1)$.

As far as concerns the instruction at line 17, this must be replaced by

$$W := ((B \,\&\, K) + K) \,|\, B\,,$$

where K is a precomputed bit mask of length L such that

$$K = (\mathsf{0}\mathsf{1}^{\alpha_{m-1}-\beta_{m-1}}\mathsf{0}^{\beta_{m-1}})(\mathsf{0}\mathsf{1}^{\alpha_{m-2}-\beta_{m-2}}\mathsf{0}^{\beta_{m-2}})\ldots(\mathsf{0}\mathsf{1}^{\alpha_0-\beta_0}\mathsf{0}^{\beta_0})\,.$$

Finally, concerning the instruction at line 19, this actually does not need to be explicitly changed; rather, we have to modify the way in which the bit mask I referred in it is computed; more precisely, we must have

$$I = (\mathsf{0}\mathsf{1}^{\alpha_{m-1}})(\mathsf{0}\mathsf{1}^{\alpha_{m-2}})\ldots(\mathsf{0}\mathsf{1}^{\alpha_0})\,.$$

The resulting algorithm, which is called CBG-Sequential-Sampling-BP (CBG-S-S-BP, for short), is reported in Figure 4 (on the right). Its time complexity is $\mathcal{O}((n+m\sigma)\lceil(m\alpha)/w\rceil)$, where σ is the size of the alphabet Σ of the pattern P and the text T and $\alpha = \max(\alpha_0,\ldots,\alpha_{m-1})$.

Acknowledgments

The authors thank three anonymous referees for their helpful comments.

References

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

[2] D. Cantone, S. Cristofaro and S. Faro, "An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences," in *Proceedings of 4-th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, S. E. Nikoletseas, ed., vol. 3503 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 428–439.

[3] D. Cantone, S. Cristofaro and S. Faro, "On tuning the (δ, α)-sequential-sampling algorithm for δ-approximate matching with α-bounded gaps in musical sequences," in *Proceedings of 6-th International Conference on Music Information Retrieval (ISMIR 2005)*, S. D. Reiss and G. A. Wiggins, eds., 2005, pp. 454–459.

[4] M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis and K. Tsichlas, "Approximate string matching with gaps," *Nordic J. of Computing*, 9(1) 2002, pp. 54–65.

[5] M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon and W. Rytter, 'Finding motifs with gaps," in *Proceedings of International Symposium on Music Information Retrieval (ISMIR 2000)*, Plymouth, Massachusetts, 2000, pp. 306–317.

[6] K. Fredriksson and Sz. Grabowski, "Efficient Bit-parallel Algorithms for (δ, α)-matching," in *Proc. 5th Workshop on Efficient and Experimental Algorithms (WEA 2006)*, LNCS 4007, Springer–Verlag, 2006, pp. 170–181.

[7] K. Fredriksson and Sz. Grabowski, "Efficient algorithms for pattern matching with general gaps, character classes and transposition invariance," *Information Retrieval*, 11(4) 2008, pp. 335–357.

[8] R. N. Horspool, "Practical Fast Searching in Strings," *Software, Practice & Experience*, 10(6) 1980, pp. 501–506.

[9] G. Navarro and M. Raffinot, "Fast and simple character classes and bounded gaps pattern matching, with application to protein searching," in *RECOMB '01: Proceedings of the fifth annual international conference on Computational biology*, New York, NY, USA, 2001, ACM, pp. 231–240.

[10] Y. J. Pinzon and S. Wang, "Simple Algorithm for Pattern-Matching with Bounded Gaps in Genomic Sequences," in *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics (ICNAAM'05)*, 2005, pp. 827–831.