Efficient (δ, γ)-Pattern-Matching with Don't Cares*

Yoan José Pinzón Ardila
Manolis Christodoulakis
Costas S. Iliopoulos
Manal Mohamed

King's College London,
Department of Computer Science,
London WC2R 2LS, UK

E-mail: Yoan.Pinzon@kcl.ac.uk,
{manolis,csi,manal}@dcs.kcl.ac.uk

Abstract

Here we consider string matching problems that arise naturally in applications to music retrieval. The δ-Matching problem calculates, for a given text $T_{1..n}$ and a pattern $P_{1..m}$ on an alphabet of integers, the list of all indices $\mathcal{I}_\delta = \{1 \leq i \leq n-m+1 : \max_{j=1}^{m} |P_j - T_{i+j-1}| \leq \delta\}$. The γ-Matching problem computes, for given T and P, the list of all indices $\mathcal{I}_\gamma = \{1 \leq i \leq n-m+1 : \sum_{j=1}^{m} |P_j - T_{i+j-1}| \leq \gamma\}$. In this paper, we extend the current result on the different matching problems to handle the presence of "*don't care*" symbols. We present efficient algorithms that calculate $\mathcal{I}_\delta, \mathcal{I}_\gamma$, and $\mathcal{I}_{(\delta,\gamma)} = \mathcal{I}_\delta \cap \mathcal{I}_\gamma$, for pattern P with occurrences of "don't cares".

Keywords: δ-matching; γ-matching; wildcard matching; music information retrieval.

1 Introduction

The string matching problem is to find all the occurrences of a given pattern $P_{1..m}$ in a large text $T_{1..n}$, both being sequences of characters drawn from a finite character set Σ. This problem is interesting as a fundamental computer science problem and is a basic need of many applications, such

*A preliminary version of this paper appeared in the proceedings of the 16th Australasian Workshop on Combinatorial Algorithms (AWOCA 2005).

as text retrieval, music retrieval, computational biology, data mining, network security, among many others. Several of these applications require, however, more sophisticated forms of searching, in the sense of extending the basic paradigm of the pattern being a simple sequence of characters.

In this paper we are interested in music retrieval. A musical score can be viewed as a string over an alphabet consisting of integers, representing the set of notes in the chromatic or diatonic notation, or the set of intervals that appear between notes (*e.g.* pitch may be represented as MIDI[1] numbers and pitch intervals as number of semitones). One of the most common uses of pattern matching in musical sequences, is for locating melodies within songs. However, two songs with the same melody are usually made of "similar" notes, but rarely exactly the same. One way to account for similarity between closely related but non-identical musical strings is to permit a difference of at most δ units between each pattern character and its corresponding text character in an occurrence. For an obvious example, a C-minor 7 $\{60, 63, 67, 70\}$, a C-major 7 $\{60, 64, 67, 71\}$ and a B-major 7 $\{59, 63, 66, 70\}$ sequence can be matched if a tolerance $\delta = 1$ is allowed in the matching process (see Fig. 1). This type of similarity is also becoming widely known as *δ-matching*.

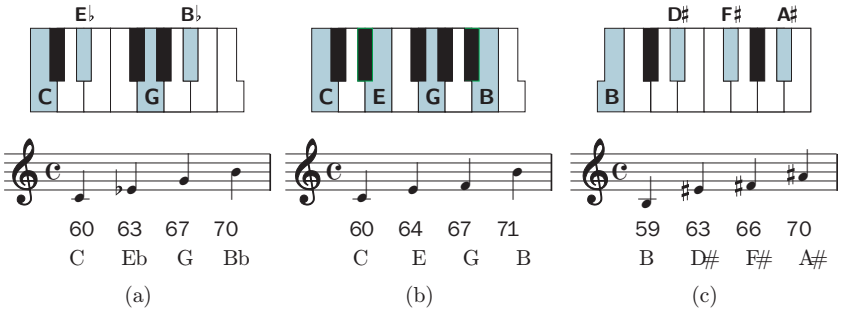

Figure 1: Representation of the (a) C-Minor 7, (b) C-Major 7 and (c) B-Major 7 chords as monophonic scores, using the chromatic scale.

A different measure of similarity is what is known as the *γ-matching*. The γ-matching again allows distances between the characters of the pattern and their corresponding characters in the text, with the difference that instead of bounding the distance between each individual pair of symbols, it bounds the sum of the distances, throughout the length of the pattern, *e.g.*, a C-minor 7 $\{60, 63, 67, 70\}$, and a B-major 7 $\{59, 63, 66, 70\}$ sequence cannot be matched if a tolerance $\gamma = 2$ is allowed in the matching process. On

[1] Musical Instrument Digital Interface.

the contrary, a C-minor 7 $\{60, 63, 67, 70\}$, and a C-major 7 $\{60, 64, 67, 71\}$ can be matched if a tolerance $\gamma = 2$ is allowed (see Fig. 1).

These two different measures of similarity, δ and γ, are usually combined together, in what is commonly called (δ, γ)-*matching*, which is the most generally accepted form of pattern matching for musical sequences. The (δ, γ)-matching is defined as follows: the alphabet Σ is assumed to be an interval of integers, $\Sigma \subset \mathbb{Z}$. Apart from the pattern P and the text T, two extra parameters, $\delta, \gamma \in \mathbb{N}$, are given. The goal is to calculate the set of all indices $\mathcal{I}_{(\delta,\gamma)}$ such that: $\forall\ i \in \mathcal{I}_{(\delta,\gamma)}$ (1) $\max_{j=1}^{m} |P_j - T_{i+j-1}| \leq \delta$, and (2) $\sum_{j=1}^{m} |P_j - T_{i+j-1}| \leq \gamma$. As an example, Fig. 2 shows the results of searching for a pattern $A = \{63, 67, 70, 63\}$ in a bigger text T. If $(\delta, \gamma) = (1, 2)$ for example, then A is found ending at position 5 in T. If γ is increased by one then we would find another occurrence terminating at position 16 in T.

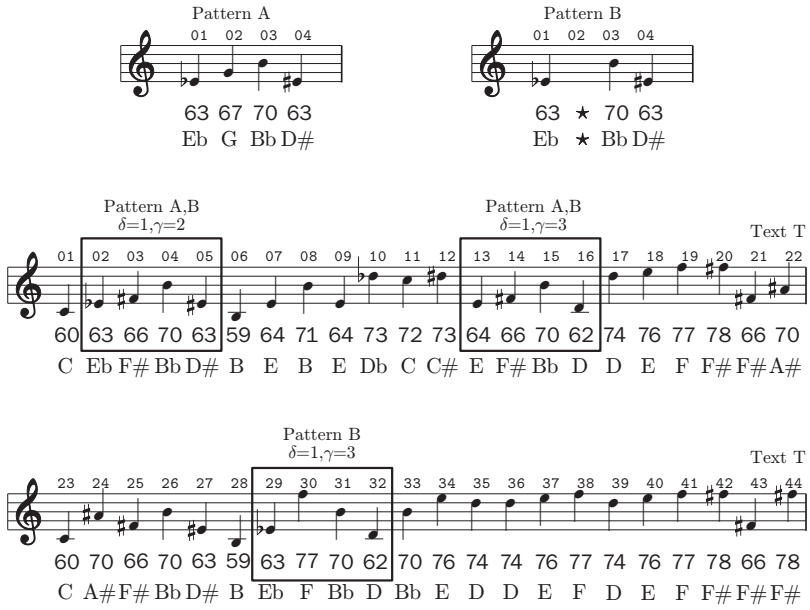

Figure 2: Illustration of some instances of the (δ, γ)-pattern matching problem to find a pattern with/without "don't care" symbols.

In [3, 7] $O(mn)$-time solutions for δ, γ, and (δ, γ)-matchings were given. Two different approaches are taken. The first employs bitwise methods known as Shift-And and Shift-Or [2]. By using the fact that calculations on the bits in a single word can be performed in parallel, considerable speedups

are possible. In general, the time complexity of the best of these algorithms is $O(nm/w)$, where w is the number of bits in a computer word. The second approach adapts classic exact matching algorithms to give the δ-*Tuned-Boyer-Moore*, δ-*Skip-Search*, and δ-*Maximal-Shift* algorithms. More recently, using Fast Fourier Transform, [8] provided an $O(|\Sigma|n(\log m + |\Sigma|))$-time algorithm for the δ-matching; also, [4] presented $O(\delta n \log m)$-time algorithms for the δ and (δ, γ)-matching. The same two groups of researchers independently presented two $O(n\sqrt{m \log m})$-time algorithms for the γ-matching.

In this paper, we extend the δ, γ and (δ, γ)-matching to handle "don't care"[2] symbols. A "don't care" symbol, is a symbol that matches every other symbol, including itself. Therefore, in the context of δ, γ and (δ, γ)-matching, the distance of the "don't care" symbol when aligned with any other symbol, ought to be 0. For example, in Fig. 2, when looking for pattern $A = \{63, 67, 70, 63\}$ and $(\delta, \gamma) = (1, 3)$, we fail to report an occurrence ending at position 32. This occurrence could be detected if we used pattern $B = \{63, \star, 70, 63\}$ instead.

The outline of the paper is as follows: Some preliminaries are described in Section 2. In Section 3 we provide two alternative algorithms for the δ-matching with "don't cares": the first runs in $O(|\Sigma|n(\log m + |\Sigma|))$ time, and the second in $O(\delta n \log m)$. An outline of the algorithm for the (δ, γ)-matching with "don't cares" is given in Section 4. Then, a γ-matching algorithm is presented in Section 5. A summary appears in Section 6.

2 Preliminaries

Throughout the paper, the *alphabet* Σ is assumed to be an interval of integers and considered to be $\Sigma = \{1, 2, ..., |\Sigma|\}$. A *text* $T = T_{1..n}$ is a string of length n defined on Σ. T_i is used to denote the *i-th element* of T, and $T_{i..j}$ is used as a notation for the *substring* $T_i T_{i+1} \cdots T_j$ of T, where $1 \leq i \leq j \leq n$. Similarly, a *pattern* $P = P_{1..m}$ is a string of length m defined on $\Sigma \cup \{\star\}$, where $\star$ is the "*don't care*" symbol.

The "don't care" *matches* every symbol including itself, that is, $\star = a$ for each $a \in \Sigma \cup \{\star\}$. A pattern P is said to *occur* in T at position i if $P_j = T_{i+j-1}$, for $1 \leq j \leq m$.

Let δ, γ be two given numbers ($\delta, \gamma \in \mathbb{N}$). Then two patterns P^1 and P^2 are said to be δ-*matched* (denoted as $P^1 =_\delta P^2$), if $\max_{j=1}^{m} |P_j^1 - P_j^2| \leq \delta$. Additionally, P^1 and P^2 are said to be γ-*matched* (denoted as $P^1 =_\gamma P^2$), if $\sum_{j=1}^{m} |P_j^1 - P_j^2| \leq \gamma$, where $| \star - a | = | a - \star | = 0, \ \forall \ a \in \Sigma \cup \{\star\}$.

The δ, γ and (δ, γ)-Matching problems are defined as follows:

[2] *a.k.a.* wildcard symbols

Definition 1 (δ-Matching Problem) *For a given text T, pattern P and integer δ, the δ-Matching (L_1-Matching) problem is to calculate the set, $\mathcal{I}_\delta$, of all indices $1 \leq i \leq n$ such that if $T_{i..i+m-1} =_\delta P$, then $i \in \mathcal{I}_\delta$. In other words, $\mathcal{I}_\delta = \{i : \max_{j=1}^{m} |P_j - T_{i\ +j-1}| \leq \delta\}$.*

Definition 2 (γ-Matching Problem) *For a given text T, pattern P and integer γ, the γ-Matching (L_∞-Matching) problem is to calculate the set, $\mathcal{I}_\gamma$, of all indices $1 \leq i \leq n$ such that if $T_{i..i+m-1} =_\gamma P$, then $i \in \mathcal{I}_\gamma$. In other words, $\mathcal{I}_\gamma = \{i : \sum_{j=1}^{m} |P_j - T_{i\ +j-1}| \leq \gamma\}$.*

Definition 3 ((δ, γ)-Matching Problem) *For a given text T, pattern P and integers δ, γ; the (δ, γ)-Matching problem is to calculate the set, $\mathcal{I}_{(\delta,\gamma)}$, of all indices $1 \leq i \leq n$ such that $\forall\ i \in \mathcal{I}_{(\delta,\gamma)}$:*
(1) $T_{i..i+m-1} =_\delta P$, and (2) $T_{i..i+m-1} =_\gamma P$.

The following definition will be central to the techniques used in this paper.

Definition 4 (Convolution) *Given two numerical strings $X_{1..n}$ and $Y_{1..m}$, the* convolution *$Z_{1..n} = X \otimes Y$ is to calculate the inner products,*

$$Z_i = \sum_{j=1}^{m} (X_{i+j-1}\ ^\bullet\ Y_j) \quad \forall 1 \leq i \leq n$$

In this paper, we will assume the RAM model of computation, which allows arithmetic on $\log N$ bit numbers in $O(1)$ time, where N is of the order of the maximum problem size. For such model, the following theorem is standard and crucial to our algorithms [6].

Theorem 1 *Consider two numerical strings $X_{1..n}$ and $Y_{1..m}$. Then, $Z_{1..n} = X \otimes Y$ can be computed accurately and efficiently in $O(n \log m)$ time using the Fast Fourier Transform (FFT).*

3 δ-Matching with "Don't Cares" Algorithm

Informally, the δ-Matching problem computes all indices i such that the maximum $|P_j - T_{i\ +j-1}|$ over all j's is no larger than δ. In the following subsections, we present two different algorithms for the δ-Matching problem in the presence of "don't cares"; we will assume that the "don't cares" appear only in the pattern P.

3.1 An $O(|\Sigma|n(\log m + |\Sigma|))$-Time Algorithm

The main idea in our first algorithm is to encode the text and the pattern in such a way that one convolution, and one linear time pass on the convolution's output are sufficient to compute the desired output. A similar idea is used in [8] to compute the L_∞-matching for strings without "don't cares". The main steps of the algorithm (Fig. 3) are as follows:

Step 1: Encode both the text T and the pattern P in such a way that every symbol $\sigma \in \Sigma \cup \{\star\}$ is represented by a binary string, over $\{0,1\}$, of length $2|\Sigma|$. In particular, $\sigma = T_i$ $(1 \leq i \leq n)$ will be replaced by the sequence $c^t(\sigma) = c^t(\sigma)_1, \ldots, c^t(\sigma)_{2|\Sigma|}$, where

$$c^t(\sigma)_j = \begin{cases} 1, \text{ if } j = |\Sigma| + \sigma \\ 0, \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq j \leq 2|\Sigma|.$$

Additionally, every symbol $\sigma = P_i$ $(1 \leq i \leq m)$ will be replaced by

$$c^p(\sigma)_j = \begin{cases} 1, \text{ if } j = \sigma \text{ and } \sigma \neq \star \\ 0, \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq j \leq 2|\Sigma|.$$

Note, that all "don't-care" symbols ($\star$), in the pattern P, are not represented by any bit in this sequence.

Step 2: We compute the convolution between the encoded text, $c^t(T) = c^t(t_1), \ldots, c^t(t_n)$, and the encoded pattern, $c^p(P) = c^p(p_1), \ldots, c^p(p_m)$

$$R = c^t(T) \otimes c^p(P).$$

Now, the i-th block of length $2|\Sigma|$, in R, represents all the matches that occur when the first symbol of P is aligned with the i-th symbol of T. In particular, the j-th $(1 \leq j \leq 2|\Sigma|)$ number of any such block of $2|\Sigma|$ numbers in R, will contain r, if and only if, the number of symbols, P_ℓ, of P whose distance from the corresponding symbol, $T_{i+\ell-1}$, of T is $P_\ell - T_{i+\ell-1} = j - |\Sigma| - 1$, is r. In other words, if we would renumber the indices of the i-th block of $2|\Sigma|$ numbers, to run from $-|\Sigma|$ to $|\Sigma| - 1$, then the k-th $(-|\Sigma| \leq k \leq |\Sigma| - 1)$ number within the i-th block, is the number of symbols in P whose δ-distance from the corresponding symbols in T is k.

Since, we are only interested in those alignments, i, for which all the non-"don't-care" symbols of P are in distance at most δ from their corresponding symbols in T, it suffices to parse the positions $|\Sigma| - \delta + 1 \leq j \leq |\Sigma| + \delta + 1$ of the i-th block of length $2|\Sigma|$, in R: if the sum of the numbers in this range equals the number of non-"don't care" symbols in P, a match has been found.

Step 3: Finally, we construct a string $D_{1..n-m+1}$, over $\{0,1\}^*$, such that D_i $(1 \leq i \leq n-m+1)$ will be 1, if and only if, P occurs at position i of T; otherwise it will be 0. The values of D are obtained as follows:

$$D_i = \begin{cases} 1, \text{ if } \sum_{j=q-\delta}^{q+\delta} R[j] = m' \\ 0, \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq n-m+1,$$

where $q = (2(i-1)+1)|\Sigma|+1$, and m' is the number of non-"don't cares" in P.

Algorithm 1: δ**-Matching algorithm** — (*first version*)

Input: T, P, δ
Output: D

1. $m' \leftarrow m$
2. $c^t(T)_{1..2|\Sigma|n} \leftarrow 00\ldots0$
3. $c^p(P)_{1..2|\Sigma|m} \leftarrow 00\ldots0$
4. **for** $i = 1$ **to** n **do** $c^t(T)_{(2(i-1)+1)|\Sigma|+T_i} \leftarrow 1$
5. **for** $i = 1$ **to** m **do**
6. **if** $p_i \neq \star$ **then** $c^p(P)_{2(i-1)|\Sigma|+P_i} \leftarrow 1$
7. **else** $m' \leftarrow m' - 1$
8. $R \leftarrow c^t(T) \otimes c^p(P)$
9. $D_{1..n} \leftarrow 00\ldots0$
10. **for** $i = 1$ **to** n **do**
11. $c \leftarrow 0$
12. **for** $j = |\Sigma| - \delta + 1$ **to** $|\Sigma| + \delta + 1$ **do** $c \leftarrow c + R_{2(i-1)|\Sigma|+j}$
13. **if** $c = m'$ **then** $D_i \leftarrow 1$
14. **return** D

Figure 3: δ-Matching Algorithm (Algorithm 1).

Example 1 (δ-matching with "don't cares") *Fig. 4 illustrates the computation of $c^t(T)$, $c^p(P)$ and R for $T = 3,5,4,2,3,5,3,2$, $P = 3,\star,4$ and $\delta = 1$. The pattern P occurs in T at positions 1,3,4 and 5.*

It is easily seen that the algorithm computes the desired results, $\mathcal{I}_\delta = \{i : D_i = 1\}$. The time complexity of this algorithm is derived from the time to compute a convolution, and the size of the alphabet. In the first step, a single pass through T and P is required, spending $O(|\Sigma|)$ time per input symbol. Then, the convolution of the encoded text and pattern of size $2|\Sigma|n$ and $2|\Sigma|m$, respectively, is computed, using the Fast Fourier Transform, in $O(|\Sigma|n \log |\Sigma|m)$ time. Finally, the parsing of the output of the convolution is done in $O(|\Sigma|n)$ time. Therefore, the overall time needed for this algorithm is $O(|\Sigma|n \log |\Sigma|m) = O(|\Sigma|n(\log m + |\Sigma|))$.

Figure 4: Illustrations of the computation of $c^t(T)$, $c^p(P)$ and R for $T = 3, 5, 4, 2, 3, 5, 3, 2$, $P = 3, \star, 4$ and $\delta = 1$.

3.2 An $O(\delta n \log m)$-Time Algorithm

Based on ideas in [5], [4] presented an $O(\delta n \log m)$-time algorithm for the δ-Matching problem. A function that is zero when there is a match between two symbols and bounded away from zero otherwise, is constructed. Standard properties of the even periodic functions and their discrete cosine transform [6] are used to achieve such goals. An even periodic function $f_\delta(x)$ that is equal to x^2 for $|x| \leq \delta$ is used to construct the desired function as follows:

$$d(x-y) = (x-y)^2 - f_\delta(x-y).$$

Note that $d(x-y) = 0$ if $|x-y| \leq \delta$, and $d(x-y) \geq 1$ otherwise. Thus, to perform δ-matching we need to compute $\sum_{j=1}^{m} d(P_j - T_{i+j-1})$, for $1 \leq i \leq n-m+1$. Calculating each $d(P_j - T_{i+j-1})$ involves $O(\delta)$ inner products; see [4] for details.

In this subsection, we show how the same algorithm can be extended to handle the occurrences of the "don't cares" in the pattern. The main steps of our algorithm (Fig. 5) are as follows:

Step 1: If $\mathcal{J}^\star = \{1 \leq j \leq m \mid P_j = \star\}$. Then, a new pattern $P^{(0)}$ is created as follows:

$$P_j^{(0)} = \begin{cases} 0, & \text{if } j \in \mathcal{J}^\star; \\ P_j, & \text{otherwise.} \end{cases} \tag{1}$$

Additionally, we compute $\mathcal{D}_i^1 = \sum_{j=1}^{m} d(P_j^{(0)} - T_{i+j-1})$, for $1 \leq i \leq n-m+1$. Note that

$$\mathcal{D}_i^1 = \sum_{j \in \mathcal{J}^\star} d(T_{i+j-1}) + \sum_{\widetilde{\mathcal{J}}^\star} d(P_j - T_{i+j-1}),$$

where $\widetilde{\mathcal{J}}^\star = \{1, ..., m\} - \mathcal{J}^\star$.

Step 2: The aim of this step is to compute for each position i in T the value $\sum_{j \in \mathcal{J}^\star} d(T_{i+j-1})$. To achieve this, a new pattern $P^{(1)}$ is computed as follows:

$$P_j^{(1)} = \begin{cases} 1, & \text{if } j \in \mathcal{J}^\star; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Additionally, a new text T' is computed as follows:

$$T'_i = d(T_i) = T_i^2 - f_\delta(T_i).$$

Note that

$$f_\delta(x) = \sum_{k=0}^{\delta} \alpha_k h_k(x),$$

where

$$h_k(x) = r(k)\cos(xk\pi/\delta), k = 0, ..., \delta,$$

the coefficients are given as

$$\alpha_k = \sum_{x=1-\delta}^{\delta} x^2 h_k(x),$$

and $r(k) = 1/\sqrt{2\delta}$ if $k \mod \delta = 0$ and $r(k) = 1/\sqrt{\delta}$ otherwise [4].

Once $P^{(1)}$ and T' are obtained, the values $\sum_{j\in\mathcal{J}^\star} d(T_{i+j-1})$, for $1 \leq i \leq n-m+1$, can be easily obtained as $\mathcal{D}^2 = T' \otimes P^{(1)}$.

$$\mathcal{D}_i^2 = \sum_{j=1}^{m} P_j^{(1)} \bullet\ T'_{i+j-1}.$$

Step 3: In this step the actual δ-matching is computed as follows:

$$\mathcal{D}_i = \mathcal{D}_i^1 - \mathcal{D}_i^2.$$

Observe that $\mathcal{I}_\delta$ can be calculated as follows:

$$\mathcal{I}_\delta = \{1 \leq i \leq n-m+1 : \mathcal{D}_i = \sum_{\widetilde{\mathcal{J}}^\star} d(P_j - T_{i+j-1}) = 0\}.$$

Let us analyse the time complexity of our algorithm. In the first step, the δ-matching for text and pattern without "don't cares" is computed in $O(\delta n \log m)$ time using the algorithm in [4]. All α_k, for a given δ, can computed in $O(\delta^2)$ time. Additional $O(\delta n)$ time is needed to compute T' in the second step. The convolution in the second step is computed in $O(n \log m)$ time, using the Fast Fourier Transform. The remaining steps require $O(n+m)$ time. Therefore, the overall time needed for this algorithm is $O(\delta n \log m)$ time; which matches the best current running time of the δ-Matching algorithm for strings without "don't cares".

4 (δ, γ)-Matching with "Don't Cares" Algorithm

Recall that [4] presented an $O(\delta n \log m)$-time algorithm for (δ, γ)-matching without "don't cares". The algorithm follows the same method for the δ-Matching algorithm. The only difference is the choice of the even periodic function. A function $f_\gamma(x)$ that is equal to $|x|$, for $|x| \leq \delta$ is used to calculate $\mathcal{I}_\gamma$.

Algorithm 2: δ-**Matching algorithm** — (*second version*)

Input: T, P, δ
Output: $\mathcal{D}$

1. **for** $k = 1$ **to** δ **do**
2. $\quad \alpha_k \leftarrow 0$
3. $\quad$ **for** $x = 1 - \delta$ **to** δ **do**
4. $\quad\quad \alpha_k \leftarrow \alpha_k + x^2 h_k(x)$
5. **for** $i = 1$ **to** n **do**
6. $\quad sum \leftarrow 0$
7. $\quad$ **for** $k = 0$ **to** δ **do** $sum \leftarrow sum + \alpha_k h_k(x)$
8. $\quad T'_i \leftarrow T_i^2 - sum$
9. **for** $j = 1$ **to** m **do**
10. $\quad$ **if** $p_j \neq \star$ **then** $P_j^{(0)} \leftarrow P_j$; $P_j^{(1)} \leftarrow 0$
11. $\quad$ **else** $p_j \neq \star$ **then** $P_j^{(0)} \leftarrow 0$; $P_j^{(1)} \leftarrow 1$
12. **for** $i = 1$ **to** $n - m + 1$ **do** $\mathcal{D}_i^1 \leftarrow \sum_{j=1}^{m} d(P_j^{(0)} - T_{i+j-1})$
13. $\mathcal{D}^2 \leftarrow T' \otimes P^{(1)}$
14. **for** $i = 1$ **to** $n - m + 1$ **do** $\mathcal{D}_i = \mathcal{D}_i^1 - \mathcal{D}_i^2$
15. **return** $\mathcal{D}$

Figure 5: δ-Matching Algorithm (Algorithm 2).

Therefore, their algorithm can be modified in the same way as described in Subsection 3.2, to handle the occurrences of the "don't care" symbols in the P, and obtain an $O(\delta n \log m)$-time algorithm.

Once $\mathcal{I}_\delta$ and $\mathcal{I}_\gamma$ are calculated, the set of locations $\mathcal{I}_{(\delta,\gamma)}$ is easily identified as $\mathcal{I}_{(\delta,\gamma)} = \mathcal{I}_\delta \cap \mathcal{I}_\gamma$.

In the following section, we will present a γ-Matching algorithm whose running time is independent of δ.

5 γ-Matching with "Don't Cares" Algorithm

For strings without "don't cares", two $O(n\sqrt{m \log m})$-time algorithms were presented for the γ-Matching (L_1-Matching) problem [1, 4]. Both algorithms use the Fast Fourier Transform, and follow the divide and conquer approach. In this section, we will explain how both algorithms can be used to compute the γ-matching when the given pattern P has occurrences of "don't cares". The main steps of our algorithm (Fig. 7) are as follows:

Step 1: Create a new pattern $P^{(0)}$ in the same way as in (1).

Step 2: Calculate for each position $1 \leq i \leq n-m+1$ the value

$$\mathcal{G}_i^1 = \sum_{j=1}^{m} |P_j^{(0)} - T_{i+j-1}|.$$

Step 3: Create a new pattern $P^{(1)}$ as in (2). Additionally, a new text T' is computed as follows

$$T'_i = |T_i|, \quad 1 \leq i \leq n.$$

Step 4: Compute $\mathcal{G}^2 = T' \otimes P^{(1)}$. That is calculate for each position $1 \leq i \leq n-m+1$ the value

$$\mathcal{G}_i^2 = \sum_{j=1}^{m} P_j^{(1)} \bullet\ T'_{i+j-1} = \sum_{j \in \mathcal{J}^*} |T_{i+j-1}|.$$

Step 5: Compute the γ-matching as follows: $\mathcal{G}_i = \mathcal{G}_i^1 - \mathcal{G}_i^2$. Clearly, $\mathcal{I}_\gamma$ can be defined as $\mathcal{I}_\gamma = \{1 \leq i \leq n-m+1 : \mathcal{G}_i \leq \gamma\}$.

In the second step, the γ-matching for text and pattern without "don't cares" is calculated in $O(n\sqrt{m \log m})$ time using either of the algorithms presented in [1, 4]. The convolution in the fourth step requires $O(n \log m)$ time, using Fast Fourier Transform. The remaining steps require $O(n+m)$ time. It follows that the γ-Matching problem for patterns with "don't cares" can solved using $O(n\sqrt{m \log m})$ time.

Example 2 (γ-matching with "don't cares") *Fig. 6 illustrates the computation of the above 1–5 steps for*

$$T = 60, 71, 62, 60, 62, 60, 71, 69, 68, 64, 60, 64, 69, 69, 62, 63, 63, 69, 60, 64$$

$P = 68, \star, 60, \star, 68$ *and* $\gamma = 1$. *Note that pattern* P *occurs starting at positions 9 in* T.

6 Conclusion

We have given several algorithms for approximate matching problems on strings of integers allowing the presence of "don't cares". In particular, for a given text $T_{1..n}$ and a pattern $P_{1..m}$ with "don't cares", we have presented two algorithms for computing the δ-matching in $O(|\Sigma| n(\log m + |\Sigma|))$ and $O(\delta n \log m)$ time, respectively. Additionally, an $O(n\sqrt{m \log m})$-time algorithm has been presented for computing the γ-matching. An outline of an algorithm for the (δ, γ)-matching that runs in $O(\delta n \log m)$ time has been given.

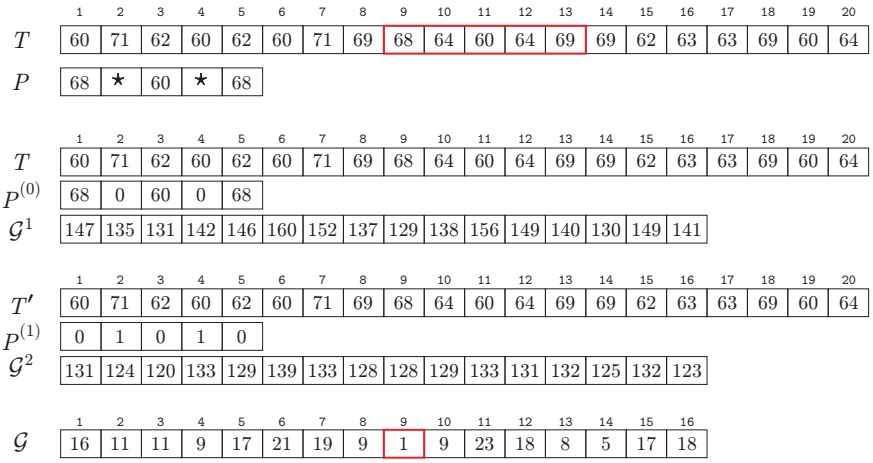

Figure 6: Illustrations of the computation of vector $\mathcal{G}$.

Algorithm 3: γ-**Matching algorithm**

Input: T, P, γ
Output: $\mathcal{G}$

1. **for** $i = 1$ **to** n **do** $T'_i \leftarrow |T_i|$
2. **for** $j = 1$ **to** m **do**
3. **if** $p_j \neq \star$ **then** $P_j^{(0)} \leftarrow P_j$; $P_j^{(1)} \leftarrow 0$
4. **else** $p_j \neq \star$ **then** $P_j^{(0)} \leftarrow 0$; $P_j^{(1)} \leftarrow 1$
5. **for** $i = 1$ **to** $n - m + 1$ **do**
6. $\mathcal{G}_i^1 \leftarrow \sum_{j=1}^{m} |P_j^{(0)} - T_{i+j-1}|$
7. $\mathcal{G}^2 \leftarrow T' \otimes P^{(1)}$
8. **for** $i = 1$ **to** $n - m + 1$ **do** $\mathcal{G}_i \leftarrow \mathcal{G}_i^1 - \mathcal{G}_i^2$
9. **return** $\mathcal{G}$

Figure 7: γ-Matching Algorithm.

Currently we are working on various two-dimensional pattern matching algorithms for music information retrieval, based on the distinct algorithms presented in this paper. For this purpose, we might need to handle the case of "don't cares" in the text as well. *i.e.* a $\star$ in the text matches any symbol in the pattern.

References

[1] A. Amir, O. Lipsky, and E. Porat. Approximate matching in the l_1 metric. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pages 91–103, 2005.

[2] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[3] E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, L. Mouchard, and Y. J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. *International Journal of Computer Mathematics*, 79(11):1135–1148, 2002.

[4] P. Clifford, R. Clifford, and C. S. Iliopoulos. Faster algorithms for δ, γ-matching and related problems. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching*, pages 68–78, 2005.

[5] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the 34th Symposium on Theory of Computing*, pages 592–601, 2002.

[6] H. Cormen, C. E. Lieserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[7] M. Crochemore, C. S. Iliopoulos, T. Lecroq, and Y. Pinzon. Approximate string matching in musical sequences. In *Proceedings of the Prague Stringology Conference*, pages 26–36, 2001.

[8] O. Lipsky. Efficient distance computations. Master's thesis, Bar-Ilan University, 2003.