A novel approach to approximate order preserving matching

Una nueva aproximación al emparejamiento con preservación de orden

Uma nova abordagem al emparelhamento com preservação de ordem

Juan C. Mendivelso-Moreno[1]; *Rafael A. Niquefa-Velásquez*[2]; *Yoán J. Pinzón-Ardila*[3] ; *Germán J. Hernández-Pérez*[4]

[1] Ingeniero de sistemas, Msc, PhD. Grupo de investigación DiscreMath: Matemáticas Discretas y Ciencias de la Computación, Facultad de Ciencias, Departamento de Matemáticas, Universidad Nacional de Colombia, Bogotá, Colombia.

[2] Ingeniero de sistemas, MSc. Grupo de Investigación FICB-PG, Institución Universitaria Politécnico Grancolombiano, Bogotá, Colombia.

[3] Ingeniero de sistemas e industrial, MSc, PhD. Universidad Pontificia Javeriana, Cali, Colombia.

[4] Ingeniero de Sistemas, MSc, PhD. Grupo de investigación en Algoritmos y Combinatoria (ALGOS), Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogota, Colombia.
Email: jcmendivelsom@unal.edu.co

Recibido: febrero 23 de 2017 **Aceptado**: abril 14 de 2017

Resumen

Un problema importante en el análisis de mercado de valores y la recuperación de información musical es el emparejamiento con preservación de orden. Este problema es una variante recientemente introducida del problema de emparejamiento de cadenas en el que busca subcadenas en el texto cuya representación natural coincide con la representación natural del patrón. La representación natural de una cadena X es una cadena que contiene los rankings de los caracteres que ocurren en cada posición de X. Entonces, el emparejamiento con preservación de orden considera la estructura interna de las cadenas en lugar de sus valores absolutos. Pero tanto en el análisis de mercado de valores como en la recuperación de información musical, se requiere más flexibilidad: no sólo las subcadenas con exactamente la misma estructura son de interés, sino también las que son similares. En este artículo se propone una versión aproximada del problema de emparejamiento con preservación de orden basada en las distancias δγ que permiten un error individual entre el ranking de los símbolos correspondientes (delimitada por δ) y un error global de todas los rankings (delimitadas por γ). Se presenta un algoritmo que resuelve este problema en $O(nm+m\ \ \log m)$. Los resultados experimentales verifican la eficiencia del algoritmo propuesto.

Palabras clave: Búsqueda de cadenas, Análisis experimental de algoritmos, Métrica de similitud de cadenas.

Abstract

A problem with important applications in stock market analysis and music information retrieval is order-preserving matching. This problem is a recently introduced variant of the string matching problem that searches for substrings in the text whose natural representation matches the natural representation of the pattern. The natural representation of a string X is a string that contains the rankings of the characters occurring at each position of X. Then, order-preserving matching regards the internal structure of the strings rather than their absolute values. But both stock market analysis and music information retrieval require more flexibility: not only the substrings with exactly the same structure are of interest, but also the ones that are similar. In this paper, we propose an approximate version of order-preserving matching based on the δγ- distances that permit an individual error between the ranking of corresponding symbols (bounded by δ) and a global error of all the positions (bounded by γ). We present an algorithm that solves this problem in $O(nm+m \log m)$. Experimental results verify the efficiency of the proposed algorithm.

Keywords: String searching, Experimental algorithm analysis, Strings similarity metric, String searching algorithms.

Resumo

Um grande problema na análise do mercado de ações e na recuperação de informações musicais é o emparelhamento com a preservação de ordem. Esse problema é uma variante recentemente introduzida do problema de correspondência de cordas que procura por substrings no texto cuja representação natural corresponde à representação natural do padrão. A representação natural de uma corda X é uma corda que contém as classificações dos caracteres que ocorrem em cada posição de X. Então, a correspondência de preservação de ordem considera a estrutura interna das cordas em vez de seus valores absolutos. Mas na análise do mercado de ações, bem como na recuperação da informação musical, é necessária mais flexibilidade: não são apenas as sub-cordas com exatamente a mesma estrutura que interessam, mas também as que são semelhantes. Neste artigo, propomos uma versão aproximada da emparelhamento com preservação de ordem com base nas distâncias δγ- que permitem um erro individual entre a classificação de símbolos correspondentes (delimitada por δ) e um erro global de Todas as posições (delimitadas por γ). Apresentamos um algoritmo que resolve este problema em $O(nm+m \log m)$. Os resultados experimentais verificaram a eficiência do algoritmo proposto.

Palavras-chave: Emparelhamento das cordas, Análise experimental dos algoritmos, Métrica de similaridade da cordas, algoritmos de busca da cordas.

Introduction

String matching is one of the most useful computational primitives (Apostolico and Galil (1997)). The input to the string matching problem consists of two strings defined over a given alphabet ∑: the pattern $P = P_{1..m}$ and the text $T = T_{1..n}$. The output should list all occurrences of the pattern string in the text string, i.e. all the positions *i* such that $P_j = T_{i+j-1}, 1 \le j \le m$. However, exact string matching does not support all the applications. For instance, in some areas the alphabet is drawn from a set of integer values. These integer strings are normally found in cipher text, financial data, meteorology data, image data, and music data, to name some. In such numeric strings, it would be unrealistic and ineffective to search for exact occurrences of a pattern but rather ought to search for similar instances of it. Then, some variants of the problem have been defined, including δγ-matching and order-preserving matching.

The δγ-matching problem consists of finding all the text windows in T for which: (i) the distance to the corresponding symbols in P is at most δ; and (ii) the sum of such distances is at most γ. In other words, the output of this problem is the set of positions i such that $|P_j - T_{i+j-1}| \le \delta, 1 \le j \le m$, and $\sum_{j=j}^{m} |P_j - T_{i+j-1}| \le \gamma$. Notice that δ bounds the individual error of each position while γ bounds the total error. Then, δγ-matching has important applications in bioinformatics, computer vision, but mainly, music information retrieval. Many kinds of algorithms have been put forward to resolve δγ-matching (see for instance Cambouropoulos *et al.*, (2002), Crochemore et al., (2002), Crochemore *et al.*, (2003), Clifford and Iliopoulos (2004), Cantone *et al.*, (2004), Crochemore *et al.*, (2005) and Lee *et al.*, (2006)). Recently, it has been used to make more flexible other string matching paradigms such as parameterized matching (see for instance Lee *et al.*, (2008) and Mendivelso (2010)), function matching (Mendivelso *et al.*, (2012)) and jumbled matching (Mendivelso *et al.*, (2015) and Mendivelso *et al.*, (2014)).

On the other hand, order-preserving matching considers the order relations within the numeric strings rather than the approximation of their values. In particular, the natural representation of a string is a string composed by the rankings of each symbol in such string. Then, order-preserving matching consists of finding every text window in *T* such that its natural representation matches the natural representation of P. Note that this problem is interested in matching the internal structure of the strings rather than their absolute values. Then, it has important applications in music information retrieval and stock market analysis. Specifically, in music information retrieval, one may be interested in finding matches between relative pitches; similarly, in stock market analysis the variation pattern of the share prices may be more interesting than the actual values of the prices (Kim *et al.*, (2014)). Since Kim *et al.*, (2014) and Kubica *et al.*, (2013) defined the problem, it has gained great attention from several other researchers (Crochemore *et al.,* (2013), Crochemore *et al.*, (2013a), Chhabra *et al.*, (2014), Faro *et al.,* (2015), Crochemore *et al.*, (2015), Hasan *et al.*, (2015), Chhabra *et al.,* (2015)).

Notwithstanding, the only approximate variant of order-preserving matching in previous literature, to the best of our knowledge, was recently proposed by Gawrychowski *et al.*, (2015). In particular, they allow *k* mismatches between the pattern and each text window. Then, they regard the number of mismatches but not their magnitude. In this paper, we propose a different approach to approximate order-preserving matching that bounds the magnitude of the mismatches through the δγ-distance. Specifically, δ is a bound between the ranking of each character in the pattern and its corresponding character in the text window; likewise, γ is a bound on the sum of all such differences in ranking. Thus, δ and γ respectively restrict the magnitude of the error individually and globally across the strings. We define δγ-order-preserving matching as the problem of finding all the text windows in *T* that match the pattern P under this new paradigm.

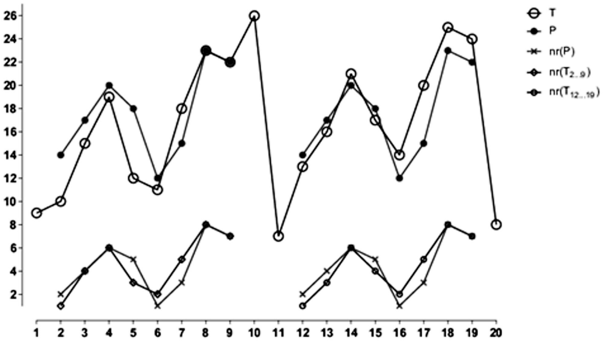

Figure 1

As an example of how δγ-order-preserving matching finds similarity in the order of the strings, we illustrate two substrings of a text *T* that are similar to a pattern *P* in Figure 1. The strings *T*=⟨9,10,15,19,12,11,18, 23, 22, 26, 7, 14, 16, 21, 17, 13, 20, 25, 24, 8⟩ and *P* = ⟨14, 17, 20, 18, 12, 15, 23, 22⟩ are defined over the alphabet ∑={1..26}. The *X*-axis and the *Y*-axis respectively correspond to the positions and values and rankings of both the pattern *P* and the substrings in *T* where there is two δγ-approximate order preserving matches with δ=2 and γ=6 in positions 2 and 12. Recall that δ is a bound on the distance between corresponding symbols and γ is a bound on the sum of such differences. The figure in the lower side shows the similarity between the natural representation of the pattern and the natural representation of the substrings $T_{2..9}$ and $T_{12..19}$. Then, with *nr(P)* in Figure 1, we refer to the natural representation of string *P*, i.e., the sequence of the rankings of the symbols in the integer string *P*. More formal definitions of these concepts are provided in the next section.

The motivation to define δγ-order-preserving matching stems from the observation that the application areas of order-preserving matching, mainly stock market analysis and music information retrieval, require to search for occurrences of the pattern that may not be exact but rather have slight modifications in the magnitude of the rankings. For example, let us assume that the text *T* presented in Figure 1 is a sequence of stock prices and that we want to determine whether it contains similar occurrences of the pattern P (also shown in this figure). Under the exact order-preserving matching paradigm, there are no matches, but there are similar occurrences at positions and 2 and 12. In particular, $T_{2..9}$ and $T_{12..19}$ are similar, regarding order structure, to the pattern. This similarity can be seen even more clearly if we consider natural representations of these strings (also shown in in Figure 1). These matches can be retrieved with δ=2 and γ=6.

The outline of the paper is as follows. In Section Preliminaries and problem definition, we present the preliminaries and define the δγ-order-preserving matching problem. Next, we present its solution in Section *Algorithm*. We evaluate the efficiency of our algorithm in Section *Experimental Results*. Finally, conclusions are drawn in the last section.

Preliminaries and problem definition

A string is a sequence of zero or more symbols from an alphabet ∑; the string with zero symbols is denoted by ε. The cardinality of alphabet ∑, denoted by |∑|, is the number of characters in ∑. The set of all strings over the alphabet ∑ is denoted by ∑*. Throughout the paper, we consider the numeric alphabet Σ_σ which is assumed to be an interval of integers from 1 to |∑|, i.e. Σ_σ={1,2,..,σ}= where |∑|=σ. $T=T_{1...n}$ is a string of length n defined over Σ_σ. T_i is used to denote the i-th element of *T*, $T_{i..j}$ is used as a notation for the substring $T_i\, T_{i+1}..T_j$ of *T*, where 1 ≤ *i* ≤ *j* ≤ *n*. Similarly, a pattern $P = P_{i..m}$ is a string of length m defined over Σ_σ. For easy notation, we use T^i to denote the length-m substring of *T* starting at position *i*; thus $T^i = T_{i..i+m-1}$. Next, we present the definition of δγ-match and order-preserving match for the string comparison problem.

Definition 1: δγ—*match*: Let $X = X_{i..m}$ and $Y=Y_{1..m}$ be two equal-length strings defined over Σ_σ. Also, let δ and γ be two given numbers (δ,γ ∈ℕ). Strings *X* and *Y* are said to δγ-match, denoted as $X =_\gamma^\delta Y, iff \max_{j=1}^{m}|X_j - Y_j| \leq \delta \text{ and} \sum_{j=1}^{m}|X_j - Y_j| \leq \gamma$. Note that the operator $=_\gamma^\delta$ is conmutative.

Example 1: There is a δγ-match, for δ=2 and γ=7, between the strings X=⟨1,3,1,3,6,3,3,4,1,2⟩ and Y=⟨2,2,1,3,4,3,4,5,2,2⟩ defined over Σ_6 as |X-Y|=⟨1,1,0,0,2,0,1,1,1,0⟩. Note that the maximum difference between corresponding characters is 2 and takes place at the fifth position. Similarly, the sum of all differences is 7.

Now, some important concepts for the definition of order-preserving match are introduced. Let X=$X_{1..m}$ be a string defined over Σ_σ. The rank of X_i in X is defined as $Rank_X(i)=1+|\ \{j : X_j < X_i, 1 \leq j < i\ \}|$. Furthermore, the natural representation of X is nr(X)=$Rank_x$ (1)$Rank_x$ (2)... $Rank_x$ (m). For simplicity of the description, we assume that all the characters appear at most once in X; however, the extension to the general case is straightforward. For example, if X = ⟨10,15,19,12,11,18,23,22⟩, then $nr(X)$ = ⟨1,4,6,3,2,5,8,7⟩.

Definition 2: **Order-Preserving Match**: Let X=$X_{1..m}$ and Y = $Y_{1..m}$ be two equal-length strings defined over Σ_σ. Strings X and Y are said to order-preserving match, denoted as $X \leftrightsquigarrow Y$ *iff* $nr(X) = nr(Y)$.

Example 2: Given integer strings X = ⟨10, 15, 19, 12, 11, 18, 23, 22⟩ and Y = ⟨12,18,22,15,13,20,30,23⟩, we conclude that $X \leftrightsquigarrow Y$ as $nr(X) = nr(Y)$ = ⟨1, 4, 6, 3, 2, 5, 8, 7⟩

Next, we define order-preserving match for the string comparison problem and then generalize it to the associated pattern matching problem.

Definition 3: δγ-order-preserving match: Let X = $X_{1..m}$ and Y=$Y_{1..m}$ be two equal-length strings defined over Σ_σ. Also, δ and γ be two given numbers (δ,γ ∈ℕ). Strings X and Y are said to δγ-order-preserving match, denoted as $X \leftrightsquigarrow^{\delta}_{\gamma}$, *iff* $nr(X) =^{\delta}_{\gamma} nr(Y)$. Note that the operator $\leftrightsquigarrow^{\delta}_{\gamma}$ is conmutative.

Example 3: Given δ= 2,γ=6, X= ⟨10,15,19,12,11,18,23, 22⟩ and Y = ⟨14,17,20,18,12,15,23,22⟩, we conclude that $X \leftrightsquigarrow^{\delta}_{\gamma} Y$, as $nr(X)$ = ⟨1,4,6,3,2,5,8,7⟩, $nr(Y)$ = ⟨2,4,6,5,1,3,8,7⟩ and $nr(X) =^{\delta}_{\gamma} nr(Y)$.

Problem 1: δγ-order-preserving matching: Let P = $P_{1..m}$ be a pattern string and T = $T_{1..n}$ be a text string, both defined over Σ_σ. Also, let δ and γ be two given numbers (δ,γ ∈ℕ). The δγ-order-preserving matching problem is to calculate the set of all indices i, $1 \leq i \leq n - m + 1$, satisfying the condition $P \leftrightsquigarrow^{\delta}_{\gamma} T^i$.

Algorithm

The input of the algorithm is the text T = $T_{1..n}$ and pattern P = $P_{1..m}$, both defined over integer alphabet Σ_σ, and the integer bounds δ,γ∈ℕ. We begin by creating a linear list with the length-m substring that starts at position 1 of the text, i.e. T^1; this list is denoted by $\mathcal{T}$. Furthermore, we create a sorted linear, called $\mathcal{S}$, with the same characters. Other two (unsorted) lists are created: $\mathcal{T}^{nr}$ and $\mathcal{P}^{nr}$; they contain the natural representations of $\mathcal{P}$ and $\mathcal{T}^1$, respectively. Then, the δγ- conditions are evaluated: the maximum difference between corresponding characters of $\mathcal{P}^{nr}$ and $\mathcal{T}^{nr}$ is compared with δ and the sum of such differences is compared with γ. If there is a δγ-order-preserving match (see Definition 3), position 1 is reported. After this, the natural representation of the other text windows is compared with the natural representation of the pattern. Specifically, our strategy is updating $\mathcal{T}^{nr}$ in $O(m)$ time so that it contains the natural representation of the next text window to consider.

Let us assume that $\mathcal{T}^{nr}$ contains the natural representation of the text window T^i, for $1 \leq i < n - m + 1$. Then, in order to transform $\mathcal{T}^{nr}$ so that it contains the natural representation of T^{i+1}, we update the ranks in $\mathcal{T}$ that are greater than the rank of $\mathcal{T}_1$ (the element that must be removed from the text window) by reducing them in one unit; note that the ranks that are less than ${\mathcal{T}_1}^{nr}$ remain the same. Then, we remove the element at the first position of $\mathcal{T}$ and $\mathcal{T}^{nr}$. Also, the corresponding element must be removed from $\mathcal{S}$, but it may be in any position of the list as it is sorted. After this step, the new character in the text window, namely $\mathcal{T}_{i+m}$, is inserted at the end of $\mathcal{T}$. It is also inserted in $\mathcal{S}$ but at the correct position according to the order. Such position corresponds to its rank in the text window, so it is added at the end of $\mathcal{T}^{nr}$. Finally, the ranks of the elements in the text window that are greater than or equal to $\mathcal{T}^{nr}_{i+m}$ are incremented in 1 due to the arrival of T_{i+m}.

The pseudocode of this algorithm, which we call δγ-*OPM*, is presented in section Algorithm 1: δγ–*OPM* algorithm. The elements of the lists are indexed from 1 to m. The methods of the lists are specified as follows:

- *add(x)*: For unsorted lists, X is inserted at the end of the list. For sorted lists, X is inserted in the correct position according to the order.
- *remove(i)*: The *i-th* element is removed from the list.
- *indexOf(x)*: It returns the index at which X occurs in the list.

The time complexity analysis of the algorithm is derived as follows. Line 1 takes constant time. Lines 2 and 4 take $O(m \log m)$ resulting from the sorting operation. The total cost of filling $\mathcal{T}$ and $\mathcal{S}$ is θ(m) as all of the insertions are done at the end of the lists (see line

3). Next, the operations within the loop of lines 5-12 are analysed for a single iteration. Evaluating the δγ-conditions takes θ(*m*) time (see line 6). Updating the rankings of $\mathcal{T}^{nr}$ takes θ(*m*) (see lines 7-8 and 11-12). Deletions from the first positions take constant time (see line 9). Even the removal from S can be done in constant time if we use doubly linked lists and keep pointers from the elements in $\mathcal{T}$ to the corresponding elements in $\mathcal{S}$. The insertions at the end of the list take *O*(1) (see line 10); however, the insertion in the sorted list $\mathcal{T}^{nr}$ takes linear time to find the correct position according to the order if doubly linked lists are used. Notice that if arrays were used to represent the linear lists, it would cost *O*(log *m*) to find the correct position of an element; however, the movement entailed by an insertion or deletion increases this cost to *O*(*m*).

Then, the total cost of the loop is *O*(*nm*) as the cost of each iteration is *O*(*m*) and there are *O*(*n*) iterations. Then, the total time complexity of δγ-OPM algorithm is *O*(*nm* + *m* log *m*). Since this is the first algorithm for the δγ-order-preserving matching problem, we compare it with a naive algorithm. In particular, a naive algorithm would calculate the natural representation of each text window and compare it with the natural representation of the pattern. Calculating the natural representation of each text window takes *O*(*m* log *m*) so the total time complexity is *O*(*n m* log *m*). The proposed algorithm improves this complexity to *O*(*nm*+*m* log *m*). It is important to remark that further improvements to this complexity are not easy to achieve since the updates on the considered text window may change all the rankings in it. Specifically, the removal of an element and the insertion of a new element in the window may greatly change the rankings. Furthermore the alignment of the pattern changes. Thus, it is necessary to evaluate the δγ- condition, which takes *O*(*m*).

Algorithm 1:** δγ**—OPM algorithm

```
Input: $P = P_{1..m}, T = T_{1..n}, \delta, \gamma, \Sigma_\sigma$
Output: $\{i \in \{1..n-m+1\}: P \overset{\delta}{\underset{\gamma}{\leftrightsquigarrow}} T^i\}$
1.  Create: $\mathcal{T}$, $\mathcal{T}^{nr}$, $\boldsymbol{\mathcal{P}}^{nr}$ as Lists, and $\mathcal{S}$ as a Sorted List.
2.  $W \leftarrow T_{1..m}.sort()$
3.  for $j = 1 \rightarrow m$ do $\mathcal{T}.add(T_j)$, $\mathcal{S}.add(W_j)$
4.  $\mathcal{T}^{nr} \leftarrow calculateNR(T_{1..m})$ ), $\boldsymbol{\mathcal{P}}^{nr} \leftarrow calculateNR(P)$
5.  for $j = 1 \rightarrow m$ do
6.   if isAMatch($\boldsymbol{\mathcal{P}}^{nr}, \mathcal{T}^{nr}, \delta, \gamma$) then report $i$
7.  for $j = 2 \rightarrow m$ do
8.    if $\mathcal{T}_j^{nr} < \mathcal{T}_1^{nr}$ then $\mathcal{T}_j^{nr} \leftarrow \mathcal{T}_j^{nr} - 1$
9.  $\mathcal{S}.remove(\mathcal{S}.indexOf(\mathcal{T}^{nr}))$, $\mathcal{T}^{nr}.remove(1)$, $\mathcal{T}.remove(1)$
10. $\mathcal{S}.add(\mathcal{T}_{i+m})$, $\mathcal{T}.add(\mathcal{T}_{i+m})$, $\mathcal{T}^{nr}.add(\mathcal{S}.indexOf(T_{i+m}))$
11. for $j = 1 \rightarrow m - 1$ do
12.   if $\mathcal{T}_j^{nr} \geq \mathcal{T}_m^{nr}$ then $\mathcal{T}_j^{nr} \leftarrow \mathcal{T}_j^{nr} + 1$
13. if isAMatch($\mathcal{P}^{nr}, \mathcal{T}^{nr}, \delta, \gamma$) then report $n$-$m$+11
```

Example 4: Given δ=2, γ=6, and strings P=⟨14,17,20, 18,12,15,23,22⟩ and *T* = ⟨9,10,15,19,12,11,18,23,22,26, 7,14,16,21,17,13,20,25,24,8⟩, defined over Σ_{30}, the output of the δγ-order-preserving matching problem is {2,12} because $P \overset{\delta}{\underset{\gamma}{\leftrightsquigarrow}} T^2$ and $P \overset{\delta}{\underset{\gamma}{\leftrightsquigarrow}} T^{12}$ (see Figure 1). Notice that $P \overset{\delta}{\underset{\gamma}{\leftrightsquigarrow}} T^2$ was shown in Example 3 by taking *P* as *Y* and T^2 as *X*. On the other hand, $P \overset{\delta}{\underset{\gamma}{\leftrightsquigarrow}} T^{12}$ since T^{12} = ⟨14,16,21,17,13,20,25,24⟩, as *nr*(*P*) = ⟨1,4,6,3,2,5,8,7⟩, *nr*(T^{12}) = ⟨2,3,6,4,1,5,8,7⟩ and *nr*(*X*) ${}^{\delta}_{\gamma}$ *nr*(*Y*).

Experimental results

In this section, we describe the experimental setup we designed to evaluate the performance of the algorithm proposed. We compare it against the naive algorithm which processes each length-m text window separately. The time complexity of our solution is *O(nm+m log m)*, while the time complexity of the naive algorithm is *O(nm log m)*. In this section, we experimentally verify these theoretical bounds. Particularly, in section Experimental setup we present the experimental framework, while we describe the data generation in Section Random data generation Then, in Section Experimental results and analysis, we discuss the results obtained.

Experimental setup: In section Hardware and software, we describe the hardware and software used for the experiments. Then, we show how we vary the input parameters in Section Parameters.

- ***Hardware and software***: Both algorithms, the naive algorithm and our solution, were implemented using C++. The computer used for the experiments was a Lenovo ThinkPad with a processor Intel(R) Core(TM) i7 4600u CPU @ 2.10GHz 2.69 GHz and installed RAM memory of 8GB. The computer was running 64-bit Linux Ubuntu 14.04.5 LTS. The C++ compiler version was g++ (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4.
- ***Parameters***: It is clear that the defined problem of Order Preserving Matching under δ and γ distances

has several parameters. They may change depending on the area of study in which the problem and pattern searching algorithms are applied. To show how our solution behaves with different configuration of the given parameters, we perform five types of experiments. In each experiment, we vary one of the given parameters *n,m*,δ,γ and ∑, and let the other four parameters fixed at a given value. For each experiment type, we performed five different experiments and took the median as the value to plot. The variation of the parameter values for each experiment type is presented in Table 1.

Table 1. Parameters of the Experiments.

Variable	*Varying n*	*Varying m*	*Varying* δ	*Varying* γ	*Varying* σ
n	[500,10000] Δ*n*=500	10000	10000	10000	10000
m	40	[5,100] Δ*m*=5	40	40	40
δ	10	10	[0,38] Δδ=2	10	10
γ	60	60	60	[0,95] Δγ=5	60
σ	100	100	100	100	[2,40] Δσ=2

Random data generation: An experiment consists of two stages. The first stage is the pseudo-random generation of a text *T* of length n and the pattern *P* of length *m*. The second stage is the execution of both algorithms (naive and the proposed algorithm) on the generated strings *P* and *T*. The random generation of each character of both the pattern *P* and the text *T* is done by calling a function that pseudo-randomly selects a number between 1 and Σ_σ with the same probability for each number to be selected.

Experimental results and analysis: The first result to highlight is the fact that, in every experiment, our solution has lower execution time than the naive solution. The results shown in Figure 2 and Figure 3 show that the size of the alphabet and the parameter γ have no

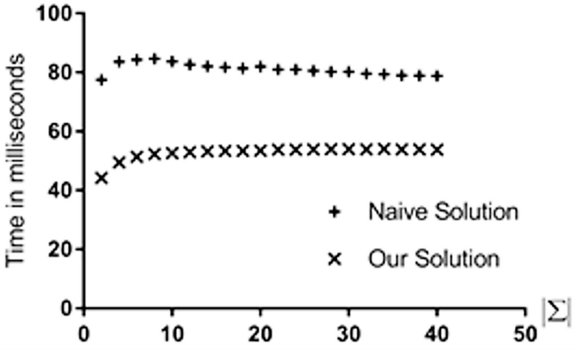

Figure 2. Experimental results varying the size of the alphabet.

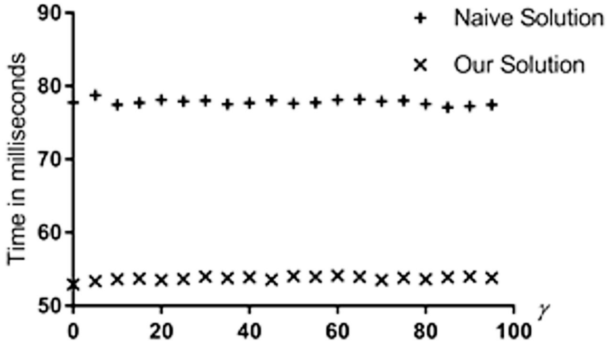

Figure 3. Experimental results varying ϒ.

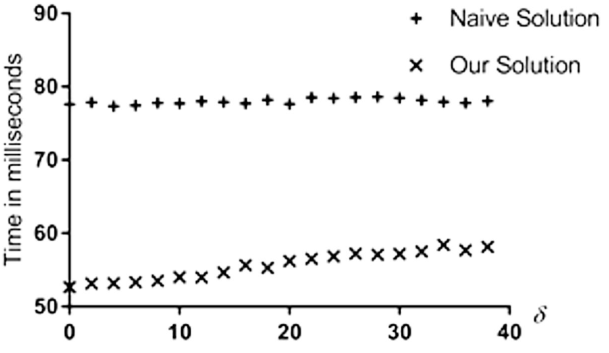

Figure 4. Experimental results varying δ.

impact on the execution time of any of the algorithms. The result shown in Figure 4 shows no effect of the parameter δ on the naive algorithm. However, increasing δ yielded a small increase in the execution time of the experiments with our algorithm. That behavior motivated the setup of more experiments with varying values of the parameter δ; notwithstading, despite the small increase of the execution time in our algorithm, our solution outperformed the naive solution in all the experiments.

Figures 5 and Figure 6 verify the theoretical complexity analysis that states that n and m are the parameters that really determine the execution time of both algorithms. In particular, the time complexity of the naive algorithm is *O(nm log m*) while ours is *O(nm + m log m)*.

In Figure 6, m is a constant and n is a variable while in Figure 5, n is a constant and m is a variable. Notice that, under these conditions, the graphs are expected to be linear and the experiments verify that. Morever, the log m extra factor of the naive algorithm's time complexity can be seen in the graphs as our solution always outperforms the naive solution in all the experiments. This also applies to the cases where both n and m are constant (see Figure 2, Figure 3 and Figure 4).

Conclusions and future work

In this paper, we propose an approximate variant of order-preserving matching that permits an individual error between the ranking of corresponding characters, and a global error across all the positions. The former is bounded by δ and the latter by γ. We present a *O(nm+m log m*) algorithm to solve this new problem. We verified its efficiency through experimental results. In particular, we clearly showed that the proposed algorithm outperforms the naive solution. To the best of our knowledge, it is the first approximate version of the problem that takes into account the magnitude of the rankings.

The question about the lower bound of an algorithm for the Order Preserving Matching under δ and γ distances remains open; it is left to see if there is an algorithm of better asymptotic complexity than *O(nm+m log m)*.

References

Apostolico A, Galil Z. 1997. Pattern matching algorithms. Oxford University Press, USA.

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

Crochemore M, Iliopoulos C, Lecroq T, Plandowski W, Rytter W. 2002. Three Heuristics for δ-Matching, δ-BM Algorithms. Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching. Springer-Verlag London, UK,178–189.

Crochemore M, et al. Occurrence and Substring Heuristics for d-Matching. Fundamenta Informaticae. 2003;56(1):1–21.

Clifford R, Iliopoulos C. Approximate string matching for music analysis. Soft Computing, 2004;8(9):597–603.

Cantone D, Cristofaro S, Faro S. 2004. Efficient Algorithms for the δ-Approximate String Matching Problem in Musical Sequences. Proc. of the Prague Stringology Conference.

Crochemore M, Iliopoulos C, Navarro G, Pinzon Y, Salinger A. Bit-parallel (δ,γ)-Matching and Suffix Automata. Journal of Discrete Algorithms. 2005;3(2-4):198–214.

Lee I, Clifford R, Kim S-R. 2006. Algorithms on extended (δ,γ)-matching. Computational Science and Its Applications-ICCSA. Springer. 1137–1142.

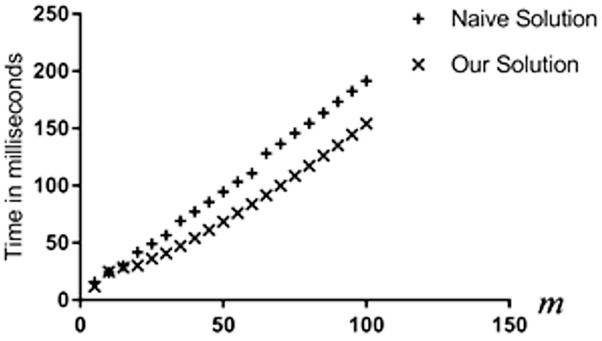

Figure 5. Experimental results varying the size of the pattern.

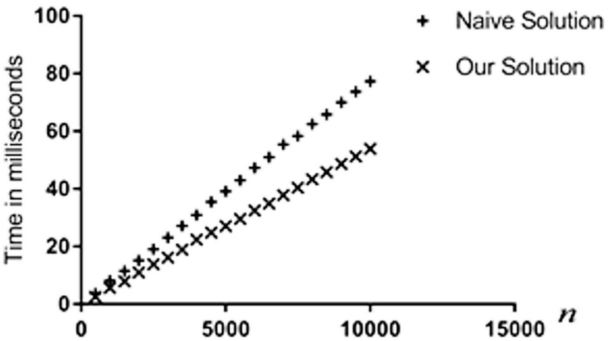

Figure 6. Experimental results varying the size of the text.

Lee I, Mendivelso J, Pinzon Y. 2008. δγ–parameterized matching. String Processing and Information Retrieval. Springer. 236–248.

Mendivelso J. 2010. Definition and solution of a new string searching variant termed δγ–parameterized matching. Master's thesis. Universidad Nacional de Colombia.

Mendivelso J, Lee I, Pinzon Y. 2012. Approximate function matching under δ-and γ-distances. String Processing and Information Retrieval. Springer. 348–359.

Mendivelso J, Pino C, Niño L, Pinzon Y. Approximate abelian periods to find motifs in biological sequences. Lecture Notes in Bioinformatics, Computational Intelligence Methods for Bioinformatics and Biostatistics. 2015;8623:121-130.

Mendivelso J, Pinzon Y. 2014. A novel approach to approximate parikh matching for comparing composition in biological sequences. Proceedings of the 6th International Conference on Bioinformatics and Computational Biology.

Kim J, Eades P, Fleischer R, Hong S H, Iliopoulos C S, Park K, Puglisi S J, Tokuyama T. Order-preserving matching. Theoretical Computer Science. 2014;525:68–79.

Kubica M, Kulczynski T, Radoszewski J, Rytter W, Walen T. A linear time algorithm for consecutive permutation pattern matching. Information Processing Letters. 2013;113(12):430–433.

Crochemore M, Iliopoulos CS, Kociumaka T, Kubica M, et al. 2013. Order preserving suffix trees and their algorithmic applications. arXiv preprint arXiv:1303.6872.

Crochemore M, Iliopoulos CS, Kociumaka T, Kubica M, Langiu A, Pissis SP, Radoszewski J, Rytter W, Walen T. 2013a. Order-preserving incomplete suffix trees and order-preserving indexes. String Processing and Information Retrieval. Springer. 84–95.

Chhabra T, Tarhio J. 2014. Order-preserving matching with filtration. Experimental Algorithms. Springer. 307–314.

Faro S, Kulekci O. 2015. Efficient algorithms for the order preserving pattern matching problem. arXiv preprint arXiv:1501.04001.

Crochemore M, Iliopoulos CS, Kociumaka T, Kubica M, Langiu A, Pissis SP, Radoszewski J, Rytter W, Walen T. 2015. Order-preserving indexing. Theoretical Computer Science.

Hasan M M, Islam AS, Rahman MS, Rahman MS. Order preserving pattern matching revisited. Pattern Recognition Letters. 2015;55:15–21.

Chhabra T, Kulekci MO, Tarhio J. 2015. Alternative algorithms for order-preserving matching. Proceedings of the Prague Stringology Conference. 36–46.

Gawrychowski P, Uznanski P. 2015. Order-preserving pattern matching with k mismatches. Theoretical Computer Science.