34th Annual International Conference of the IEEE EMBS
San Diego, California USA, 28 August - 1 September, 2012

Approximate string matching using phase correlation

Alfonso Alba[1], Margarita Rodriguez-Kessler[2], Edgar R. Arce-Santana[3] and Martin O. Mendez[4]

***Abstract*— A novel method for approximate string matching with applications to bioinformatics is presented in this paper. Unlike most methods in the literature, the proposed method does not depend on the computation of the edit distance between two sequences, but uses instead a similarity index obtained by applying the phase correlation method. The resulting algorithm provides a finer control over the false positive rate, allowing users to pick out relevant matchings in less time, and can be applied for both offline and online processing.**

I. Introduction

Approximate string matching (ASM) is the problem of finding one or more instances of a pattern string $a_1 a_2 \dots a_m$ within a longer sequence $b_1 b_2 \dots b_n$, where each instance may differ from the pattern string in a way that may be quantified by an adequate metric. The most typical metrics are the Hamming distance, which indicates the number of mismatched symbols between two sequences of the same length, i.e., the number of symbol substitutions required to transform one sequence into the other, and the Levenshtein distance (also called edit distance) which measures the number of edit operations (insertions, deletions, and substitutions) required to transform one string into another, not necessarily of the same length.

ASM has a wide number of applications, including database searching, spelling correction, plagiarism detection, and bioinformatics [1], [2]. In many of these applications, algorithms are required to search one or more patterns within very large databases; this has led to the development of very efficient ASM algorithms, some of which perform in linear or even sub-linear time; i.e., the number of operations performed is, in the worst case, proportional to the size of the search space n [3], [4]. For a good review of the most relevant ASM methods and their applications, see [5], [6], [7], [8], and [9].

One major drawback of most of these methods is that they produce a significant amount of false positives; that is, approximate instances of the pattern string that are observed by chance, but are of no relevance for the particular application under development. In large databases, it is common to observe from ten thousand to a hundred million of false positives, depending on the length of the pattern string [10]. For many of these methods, the only parameter is the maximum distance k (e.g., Hamming or Levenshtein) allowed between the pattern string and a reported instance within the search sequence. If k is too small, there is the risk that a relevant instance will be missed (false negative), and if k is too large, a large number of false positives will be reported. Depending on the application, it is often preferable to reduce the number of false negatives (type II error) even at the cost of many false positives, but then it is up to the user to cull the results and discard those instances that are of no relevance. This is often more time consuming than the search process itself, and may be one of the main reasons why developers do not always bother using state-of-the-art ASM algorithms.

The aim of this work is to present an algorithm for approximate string matching which is not based on either the Hamming or the Levenshtein distance, but provides finer control over the amount of false positives (type I error). The proposed method is based on estimating a similarity index at each position in the search string, based on the phase-only correlation (POC) function.

This paper is organized as follows: first, the classic phase correlation method is presented. Then, the proposed methodology is explained, followed by some preliminary results (with both synthetic sequences and real protein sequences) and the corresponding discussion. Finally, the conclusions are drawn.

II. Phase correlation method

The phase correlation method is a widely used signal and image aligning technique with applications in image registration [11], stereo vision [12], motion estimation [13], and video encoding [14]. This technique has its foundations in the shift properties of the Fourier transform. Consider a discrete-time signal $f(x)$ and a shifted version $g(x) = f(x - d)$, and suppose we are interested in finding the unknown displacement d. In the classic phase correlation method, one computes the normalized cross-spectrum $R(\omega) = F(\omega)G^*(\omega)/|F(\omega)G^*(\omega)|$ where F and G are the Fourier transforms of f and g, respectively, and G^* represents the complex conjugate of G. The inverse Fourier transform $r(x)$ of $R(\omega)$ is known as the phase-only correlation (POC) function. Note that, if g is truly a shifted version of f, then $G(\omega) = F(\omega)\exp\{-j2\pi\omega d\}$; therefore, $R(\omega) = \exp\{j2\pi\omega d\}$ and $r(x) = \delta(x+d)$, where δ represents the Dirac impulse function. This suggests that,

This work was supported in part by grant 154623 from CONACYT, Mexico.

All authors are with Facultad de Ciencias, Universidad Autonoma de San Luis Potosi, Av. Salvador Nava Mtz S/N, Zona Universitaria, 78230 San Luis Potosi, SLP, Mexico

[1]fac@fc.uaslp.mx (corresponding author)
[2]mrodriguez@fc.uaslp.mx
[3]arce@fciencias.uaslp.mx
[4]mmendez@fc.uaslp.mx

in practice, one can determine d by locating the maximum of the POC function $r(x)$.

When $g(x)$ is not a shifted version of $f(x)$, but instead contains various segments of $f(x)$, each of them shifted by a different amount, it can be shown that the POC function can be modeled as the sum of distorted and displaced delta functions, each of which is centered at a position corresponding to one of the displacement values [15]. In other words, if $g(x)$ contains various displaced versions of $f(x)$, it is possible, to some extent, to estimate those displacements by computing the maxima of the POC function.

Another way to see this is to consider that the POC function $r(x)$ gives an estimate of the likelihood that an instance of f is located in g beginning at position x.

III. Proposed methodology

In order to apply the phase correlation method to discrete data, such as sequences of symbols taken from a finite alphabet $\mathcal{A}$, it is necessary to transform the data into a series of numbers (i.e., a discrete-time signal). One way to do this is by means of a mapping function $\phi : \mathcal{A} \to \mathbb{R}$, which we call *codebook*. One well-known example of a codebook is the ASCII code. The transformation of a sequence $a_1 a_2 \dots a_m$ into a signal $f(x)$ is straightforward, given simply by $f(x) = \phi(a_x)$, $x = 1, \dots, m$. It is important, however, to use the same codebook to encode both the pattern string and the search sequence, resulting in signals $f(x)$ and $g(x)$, respectively. Once these signals are obtained, it is possible to estimate the POC function using the classic method as shown in the previous section.

Note that it is possible to design a codebook so that specific symbols are considered equivalent (by assigning them similar codes) or by penalizing specific symbol substitutions (by assigning those symbols values which are far apart in the codebook). A generic algorithm, however, should not attempt to favor or penalize any specific substitutions; thus, it is a good idea to compute the average POC over different (arbitrarily chosen) codebooks.

In most database searching applications, the search sequence is typically much longer than the pattern string. Instead of computing the Fourier transform of such long sequences, we suggest to divide the search string in overlapping segments whose length is a power of two (so that efficient FFT implementations can be used). The length w of these segments must be greater than the length m of the pattern string, for example, $w = 2^q$ where $q = \lceil \log_2 m \rceil$. The overlapping between segments should be at least of m symbols, in order to avoid missing instances which may lie near the limits of a segment.

With these things in mind, we propose an algorithm which takes as inputs the pattern string $a_1 a_2 \dots a_m$, the search sequence $b_1 b_2 \dots b_n$, the segment length $w \geq m$, and a set of C codebooks $\Phi = \{\phi_1, \dots, \phi_C\}$. The output of the algorithm is a similarity index I_k for each $k = 1, \dots, n$ indicating the likelihood of observing an approximate copy of the pattern string at position k of the search sequence. The algorithm is as follows:

1) Initialize $I_k = 0$ for $k = 1, \dots, n$.
2) Let $N = \lceil \frac{n-w}{w-m} + 1 \rceil$ be the number of segments into which the search sequence will be split.
3) Let $\Delta = \frac{n-w}{N-1}$ be the distance between the initial position of consecutive segments.
4) For each codebook $\phi \in \Phi$ do the following iteration:
   a) Let $f(x) = \phi(a_x)$ for $x = 0, \dots, m-1$ and $f(x) = 0$ otherwise.
   b) For each segment $j = 0, \dots, N-1$ do
      i) Let $g(x) = \phi(b_{\lfloor j\Delta \rfloor + x})$, for $x = 0, \dots, w-1$.
      ii) Compute the POC function $r(x)$ between $f(x)$ and $g(x)$.
      iii) Let $I_{\lfloor j\Delta \rfloor + x} = I_{\lfloor j\Delta \rfloor + x} + r(x)/C$ for $x = 0, \dots, w-1$.

Figure 1 shows an example of the matching indices found in a text string when searching for the pattern `ing`. Note that the algorithm can be parallelized in various ways; for example, by distributing the segments across different threads, or by dedicating one core to each codebook. Such implementations, however, are beyond the scope of this article. Also, the algorithm can be easily modified (by changing the order of the two main loops) to process large search sequences online with minimal memory requirements. Finally, when searching more than one pattern, one can store the FFTs of the search sequence segments in order to save time.

Once the matching indices have been computed, we have to select those that may represent relevant instances of the pattern string. To do this, we define a matching threshold $\tau = \mu + T\sigma$, where μ and σ represent the mean and standard deviation of the matching indices $I_0, \dots, I_{n-1}$, and T is a parameter provided by the user; reasonable values for T range between 2 and 4. The algorithm reports a matching instance at position x if $I_x > \tau$. If desired, one could then estimate an edit distance for each match as a complementary selection criterion.

Higher values of T will reduce the number of false positives at the possible expense of an increased number of false negatives; however, unlike the (integer) edit distance threshold used in most algorithms, T is a real number which can be finely tuned. Moreover, one can try different thresholds without having to recompute the matching indices, allowing the process of culling false positives to be semi-automated.

A quick analysis of the algorithm will show that it requires to compute approximately $3CN$ FFTs of size w, each of which has a time complexity of $O(w \log w)$. Since w is related more or less linearly to the pattern length m, and wN is proportional to the length n of the search sequence, then the execution time of the proposed algorithm is of order $O(Cn \log m)$, or simply $O(n \log m)$ if C is fixed.

IV. Results and discussion

A simple (non-parallel) implementation of the proposed method was written in GNU C for the MinGW compiler. For increased efficiency, FFTs were implemented using the

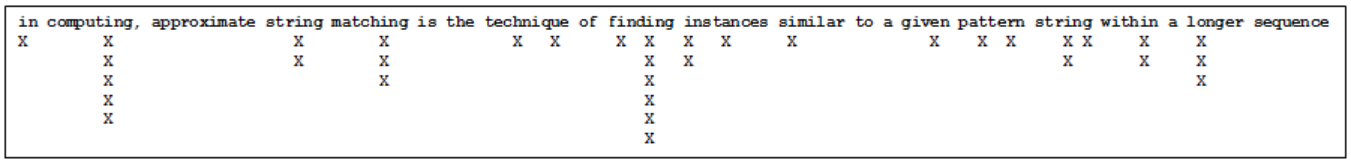

Fig. 1. Example of matching indices assigned to each position in a search sequence. The number of 'X' symbols below each position indicate the degree of similarity between the substring starting at that position and the pattern string. The pattern string being searched in this case is 'ing'.

FFTW library [16]. All tests were performed in an Intel 2.4 GHz Core2Duo CPU (although the algorithm runs in a single core) with 4 Gb of RAM and Windows Vista as the operating system.

In order to evaluate the efficiency of the proposed method, a series of test cases was randomly generated with the following parameters: pattern string length of $m = 32$, search sequence length of $n = 2^{20}$ (1 Gb), alphabet size of 32 symbols, and segment/FFT size $w = 64$. The search sequence contains 256 non-overlapping approximate instances of the pattern string with known positions. The approximate instances are obtained by applying k edit operations (insertions, deletions, and substitutions) to the pattern string. For each of the 256 known positions where the true instances were placed, we count a true positive if the position coincides with one of the candidates reported by the proposed algorithm with a difference of at most k. This is because insertion and deletion operations may affect the relative positions of the true instances and the ones found by our method. If a reported candidate does not lie within k symbols of one of the true known positions, it is counted as a false positive. Note, however, that it is possible to observe many contiguous false positives which may correspond to the same spurious instance, so the false positive rate is, in fact, overestimated in our evaluation.

Tests were performed for three different cases, corresponding to $k = 3, 5, 10$; an increased number of false positives and false negatives is expected for larger k-values, due to the alterations applied to the different instances of the pattern string. For each value of k, 100 test cases (consisting of a pattern string and a search sequence) were generated and the average true positive and false positive rates were estimated, as well as the average computation time. The number of codebooks or iterations C used to estimate the average POC function was also varied from 1 to 16 in order to analyze its effect on the performance; each codebook is obtained as a random permutation of the ASCII encoding, i.e., each symbol is assigned a random but unique code between 0 and 127. For each test case, the optimal value of the matching threshold T_{opt} is found by performing a brute force search to minimize the following criteria:

$$T_{\text{opt}} = \arg\min_{T \in \mathcal{T}} \{FP + \lambda FN\}, \qquad (1)$$

where $\mathcal{T} = \{0, 0.5, \ldots, 10\}$ defines the range of test values for the threshold T, FP and FN represent the number of false positives and false negatives, respectively, obtained with a given value of T, and λ is a penalization factor for false negatives. Since we are interested in minimizing the Type II error, a large value of $\lambda = 2^{12}$ (the ratio between n and the number of true instances) was chosen.

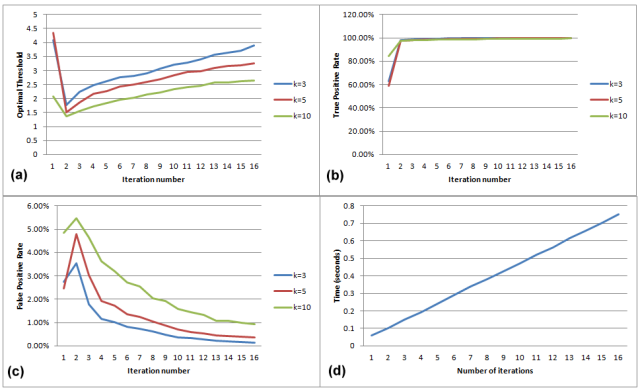

Fig. 2. Evaluation of the proposed method with a set of synthetic cases (see text for details) with different levels of difficulty given by the number of edit operations ($k = 3, 5, 10$): (a) Optimal threshold (which minimizes the FPR while maintaining the TPR close to 100%), (b) True Positive Rate, (c) False Positive Rate, (d) Computaton time.

Figures 2(a)-(c) show the average optimal threshold T_{opt}, true positive rate (TPR), and false positive rate (FPR), respectively, for $k = 3, 5, 10$ and $C = 1, \ldots, 16$. Note that the optimal threshold can be increased more or less linearly with respect of the number of iterations performed by the algorithm: this suggests that, with each iteration, the similarity index corresponding to true instances is reinforced. Also, the TPR quickly becomes close to 100% after the first 3 or 4 iterations, while the FPR decreases monotonically after the second iteration. Figure 2(d) shows the computation time, which does not depend on k, and clearly increases linearly with respect to the number of iterations/codebooks.

One more test was performed with three real plant protein sequences: *Opuntia strepthacanta* OpsDHN1 SK_3-type DHN (GenBank accession no. HO058650), *Arabidopsis thaliana* K_6-type DHN (GenBank accession no. AEE78733), and *A. lyrata* Y_3SK_2-type DHN (Phytozome accession no. 496967) [17], [18]. Dehydrins (DHN) are a family of intrinsically unstructured plant proteins that belong to the group 2 LEA proteins. These proteins are characterized by the presence of one or more K-segments (conserved lysine-rich consensus regions) that consist of 15 amino acid residues [EKKGIM(E/D)KIKEKLPG], which form a putative amphipathic α-helix [19], [20]. DHN proteins may also contain S-segments [LHRSGS$_{4-10}$(E/D)$_3$] formed by a stretch of 4-10 serine residues and Y-segments [(V/T) D(E/Q)YGNP]

Fig. 3. Results with real protein sequences (see text for details). For each sequence, we searched for three motifs which are characteristic of S-, K-, and Y-segments, respectively. The proposed method was able to detect all the known segments with only one false positive in the OpsDhn1 sequence, shown inside a rectangle.

that are located near the N-terminus, [19], [20]. In general, DHN proteins share low sequence identities, and the proper identification of conserved motifs is an important task for the classification of these proteins into DHN sub-types: Y_nSK_2, K_n, SK_n, K_nS, and Y_2K_n.

In each of the selected protein sequences, we searched for the following amino acids in the motifs: EKKGIMD-KIKEKLPG (K-segment), LHRSGS (S-segment), and DEYGNP (Y-segment). We started with a high threshold value $T = 4.0$ and decreased it until the known segments were detected. In the case of the S- and Y-segments, the default value of $T = 4.0$ produced the desired results. For the K-segments, the threshold had to be decreased until $T = 2.3$ to allow larger deviations from the pattern string. In all cases 16 iterations were performed and the total computation time (for a total of nine searches) was 15 milliseconds. The results are shown in Fig. 3. Only one false positive was found, corresponding to the OpsDHN1 sequence. However, this instance can be easily discarded in a postprocessing stage given that its length (5 symbols) is much shorter than the expected length of the motif (15 symbols).

V. Conclusions

An novel technique for approximate string matching was presented. The main difference with respect to existing methods is that our algorithm does not attempt to compute an edit distance, such as the Hamming or Levenshtein metrics. Instead, we estimate a similarity index at each position in the search sequence which represents the likelihood of an instance of the pattern string beginning at that position. This index is obtained as the average phase-only correlation computed across various encodings of both sequences. Preliminary tests demonstrate the viability of the proposed method, and its applicability to bioinformatics. The main advantages of this method are (1) a finer control over the false positive rate that does not require recomputation of the similarity index, (2) the possibility of parallelization in various levels, and (3) the ability to include other metrics as complementary matching criteria. Future work will focus on the design of adequate codebooks for specific applications, and the parallelization of the algorithm for multiple searches.

References

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[3] W. I. Chang and E. L. Lawler, "Sublinear approximate string matching and biological applications," *Algorithmica*, vol. 12, pp. 327–344, 1994.

[4] R. A. Baeza-Yates and C. H. Perleberg, "Fast and practical approximate string matching," *Inf. Process. Lett.*, vol. 59, no. 1, pp. 21–27, 1996.

[5] E. Ukkonen, "Algorithms for approximate string matching," *Inf. Control*, vol. 64, no. 1-3, pp. 100–118, 1985.

[6] P. Jokinen, J. Tarhio, and E. Ukkonen, "A comparison of approximate string matching algorithms," *Softw. Pract. Exper.*, vol. 26, no. 12, pp. 1439–1458, 1996.

[7] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.

[8] G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio, "Indexing methods for approximate string matching," *IEEE Data Engineering Bulletin*, vol. 24, no. 4, pp. 19–27, 2001.

[9] L. Boytsov, "Indexing methods for approximate dictionary searching: Comparative analysis," *J. Exp. Algorithmics*, vol. 16, pp. 1.1:1.1–1.1:1.91, 2011.

[10] J. Buhler, "Efficient large-scale sequence comparison by locality-sensitive hashing," *Bioinformatics*, vol. 17, no. 5, pp. 419–428, 2001.

[11] B. S. Reddy and B. N. Chatterji, "An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration," *IEEE Transactions on Image Processing*, vol. 5, no. 8, pp. 1266–1271, 1996.

[12] M. A. Muquit, T. Shibahara, and T. Aoki, "A High-Accuracy Passive 3D Measurement System Using Phase-Based Image Matching," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E89-A, no. 3, pp. 686–697, 2006.

[13] K. Takita, M. A. Muquit, T. Aoki, and T. Higuchi, "A Sub-Pixel Correspondence Search Technique for Computer Vision Applications," *IECIE Trans. Fundamentals*, vol. E87-A, no. 8, pp. 1913–1923, 2004.

[14] L. H. Chien and T. Aoki, "Robust Motion Estimation for Video Sequences Based on Phase-Only Correlation," *6th IASTED International Conference on Signal and Image Processing*, pp. 441–446, 2004.

[15] A. Alba, E. R. Arce-Santana, R. M. Aguilar-Ponce, and D. U. Campos-Delgado, "Phase-correlation guided area matching for realtime vision and video encoding," *Journal of Real-Time Image Processing*, vol. In press, 2012.

[16] M. Frigo and S. G. Johnson, "FFTW Web Page," http://www.fftw.org, 2012.

[17] A. Ochoa-Alfaro, M. Rodríguez-Kessler, M. Pérez-Morales, P. Delgado-Sánchez, C. Cuevas-Velazquez, G. Gómez-Anduro, and J. Jiménez-Bremont, "Functional characterization of an acidic SK_3 dehydrin isolated from an *Opuntia streptacantha* cDNA library," *Planta*, vol. 235, pp. 565–578, 2012.

[18] M. Hundertmark and D. K. Hincha, "LEA (late embryogenesis abundant) proteins and their encoding genes in *Arabidopsis thaliana*," *BMC Genomics*, vol. 9, p. 118, 2008.

[19] C. R. Allagulova, F. R. Gimalov, F. M. Shakirova, and V. A. Vakhitov, "The plant dehydrins: Structure and putative functions," *Biochemistry (Moscow)*, vol. 68, pp. 945–951, 2003.

[20] M. Pessarakli, Ed., *Role of dehydrins in plant stress response*, 3rd ed. CRC Press, Florida, 2010, pp. 239–285.