# A Hybrid Heuristic/Deterministic Dynamic Programing Technique for Fast Sequence Alignment

Talal Bonny
Department of Electrical and Computer Engineering
College of Engineering
University of Sharjah, UAE

***Abstract*—Dynamic programming seeks to solve complex problems by breaking them down into multiple smaller problems. The solutions of these smaller problems are then combined to reach the overall solution. Deterministic algorithms have the advantage of accuracy but they need large computational power requirements. Heuristic algorithms have the advantage of speed but they provide less accuracy. This paper presents a hybrid design of dynamic programing technique that is used for sequence alignment. Our technique combines the advantages of deterministic and heuristic algorithms by delivering the optimal solution in suitable time. we implement our design on a Xilinx Zynq-7000 Artix-7 FPGA and show that our implementation improves the performance of sequence alignment by 63% for in comparison to the traditional known methods.**

***Keywords*—Sequence alignment, dynamic programing, Performance, optimization, FPGA**

## I. Introduction

A large computing problem can be handled using dynamic programing. The programing that is a method for solving a complex problem by breaking it down into a collection of simpler sub-problems. This method appears to be both very precise and efficient. However, it does require a very large amount of computational power. An example of using dynamic programing is sequence comparison in computational linguistics [5], [9], [15]. In this field, researchers have established that almost all European languages are related and belong to a single family. Genetically related languages originate from a common proto-language. In the absence of historical records, proto-languages have to be reconstructed from surviving cognates. Sequence comparison is used to find the cognates in large database of divergent languages to reconstruct their proto-form and consequently, to reconstruct an entire proto-language which is an extremely time-consuming process that has yet to be accomplished for many language families.

Image processing is also using dynamic programing for Sequence comparison to retrieve information on handwritten document images [23]. Each word image is represented as a sequence of graphs and the similarity between word images is measured.

In Biology [10], [13], [14], dynamic programing is used to search a large database of sequences for close matches to particular sequence of interest, typically a recently discovered protein. If correlations are found, new drugs may be developed or better techniques invented to treat the disease.

Deterministic algorithms, such as Needleman-Wunsch [3] (for global alignment) and Smith-Waterman [4] (for local alignment), guarantee the return of the optimal alignment of two sequences. The first one is called query sequence (Q) and the second one is called database sequence (D). These kinds of algorithms take a long time to find the highest similarity score as the computing and memory requirements grow proportionally to the product of the lengths of the two sequences being compared, i.e, if n is the length of the query sequence and m is the length of the database sequence, then the previous algorithms provide the optimal alignment (highest similarity score) in n x m steps. Therefore when searching a whole database the computation time grows linearly with the size of the database. Heuristic algorithms, such as FASTA [1] and BLAST [2], provide an approximated solution by comparing query sequence to database sequence and calculating the statistical significance of matches. An approximation is obviously faster than an optimal solution provided by deterministic algorithms since less and easier calculations need to be performed, but it is less accurate because it might miss one or more unexpected but important homologies that would be found in the exact solution.

Various methods and techniques have been proposed to improve the speed of implementations of such algorithms [7]: In [11], the authors implemented the Recursive Variable Expansion (RVE) based technique, which is proved to give better speedup than any best dataflow approach at the cost of extra area. The authors computed a block of (k x k) elements in parallel instead of computing one element on the FPGA. Compared to dataflow approach, their implementation was 2.29 times faster at the expense of 2.82 times more area.

In [12], the authors developed new tool, called SWIPE, for sequence alignment based on the Smith-Waterman algorithm. In SWIPE, residues from 16 different database sequences are processed in parallel and compared simultaneously to the same query residue. The operations are carried out using vectors consisting of 16 independent bytes. The 16 residues are fed into sixteen independent channels. When the first of these sixteen database sequences ends, the first residue of the next database sequence is loaded into the channel. SWIPE was found to be performing at a speed of 106 GCUPS with a 375 residue query sequence on a dual Intel Xeon X5650 six-core processor system. The authors achieved over six times more rapid than software based on Farrar's 'striped' approach [6].

In [16], the authors used GPU and CPU to improve the performance of aligning the sequences by running the long sequences

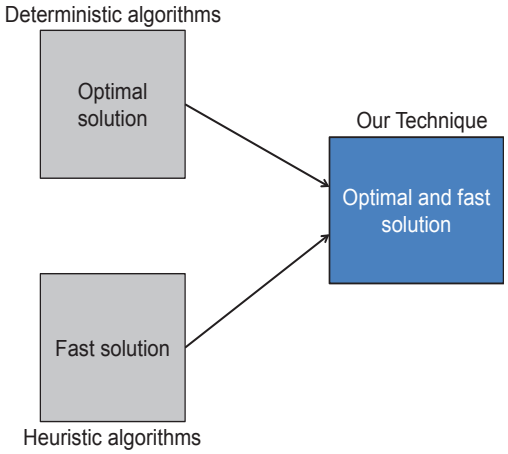


Fig. 1: Our technique combines the advantages of deterministic and heuristic algorithms

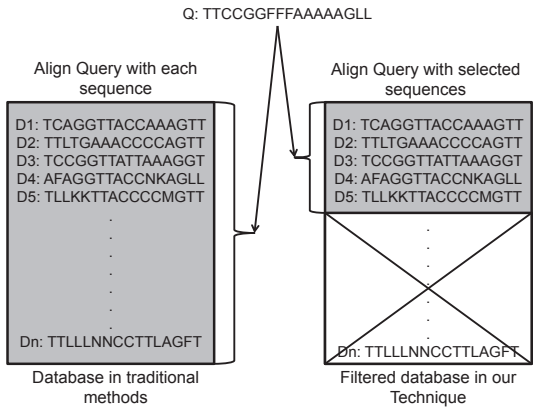


Fig. 2: Sequence alignment in traditional methods and in our technique

on the GPU and the short ones on the CPU. Another different methods were introduced for heuristic sequence alignment.
In [18], the authors proposed, a parallel algorithm that uses GPU to align huge sequences. Their algorithm (CUDAlign 2.1 ) combined the Smith-Waterman algorithm with linear space complexity. In order to achieve that, they proposed optimizations which are able to reduce significantly the amount of data processed, while enforcing full parallelism most of the time. Using the NVIDIA GTX 560 Ti board and comparing real DNA sequences that range from 162 KBP (Thousand Base Pairs) to 59 MBP (Million Base Pairs), they showed that CUDAlign 2.1 is able to produce the optimal alignment between the chimpanzee chromosome 22 (33 MBP) and the human chromosome 21 (47 MBP) in 8.4 hours.

In all previous work and applications, one object has to be searched/compared/aligned with all objects in the database to find the most closest one by using deterministic or heuristic algorithm. The object might be database sequence, string file, video stream, website page, etc. We have found that to get the optimal alignment, we do not need to apply the alignment algorithms on the whole database sequences. Instead, many database sequences may be excluded from the searching process. In this case, the alignment algorithms may only be applied on the remaining sequences of the database. This will reduce the searching scope and consequently the time required to find the optimal alignment.

In this work, an efficient hybrid design of dynamic programing technique is introduced. It has the advantage of deterministic algorithms, which is delivering optimal solution, and the advantage of heuristic algorithms, which is delivering fast solution, to provide the optimal solution in suitable time (see Fig. 1). Our technique uses new criteria to measure the difference score for each sequence of the database. It excludes the sequences which have high difference scores from the searching process and applies the dynamic programing algorithm (Needleman-Wunsch or Smith-Waterman) on part of the database and not on the whole of it.
Using our technique, we explicitly improve the time performance of the database sequence comparison applications by 63% in comparison to the traditional methods used. Applying our technique in conjunction with the previous state-of-the-art methods (as in [17], [20], [22]) will further improve their time performance.

The rest of the paper is organized as follows. In Section 2, we introduce the traditional methods for database sequence computing applications using Needleman-Wunsch algorithm and then introduce our technique. The complexity of our technique is introduced in Section 3. In Section 4, we present the hardware implementation of our technique on a FPGA. Experimental results are presented in Section 5. We conclude this paper in Section 6.

## II. TYPICAL DATABASE SEQUENCE COMPUTING APPLICATIONS

In any database sequence computing application, the data must be coded into a set of sequences using a finite alphabet of states. Then, a cost matrix must be defined on these states and a gap cost scheme must be chosen. This matrix contains the cost of each operation required to transform one of a pair of sequences into the other. The operations used for transformation are insertion, deletion and replacement. An alignment algorithm is then applied on the database sequence and the compared one, resulting in a matrix of distances between all pairs of sequences. The database sequence alignment method is described in details, in the next sections, using the traditional methods and using our technique.

### *A. SEQUENCE ALIGNMENT USING TRADITIONAL METHODS*

Traditional methods for aligning sequences are based on align the query sequence [1] 'Q' with each sequence of the database starting from the first sequence 'D1' of the database till the last one '$D_n$' (see left part of Fig. 2). In each alignment process, a score to each cell comparison between the two sequences is computed. The score is based on the result of the comparison, which is either match, or mismatch. If the sequences are mismatched, then one of three operations may be

[1]the query is the searched sequence in the database or the compared sequence with other sequences in the database

done: insertion, deletion, or substitution. Gaps may be added to one or both sequences to make them close to each others Each of these operations has a previously defined score.
For each alignment process between the query and the database sequence, an alignment score (AS) is computed as following:

$$\begin{aligned} AS \;=\; & (\#\ of\ matches \times match\_score) \\ & + (\#\ of\ gaps \times gap\_score) \\ & + (\#\ of\ mismatches \times mismatch\_score) \end{aligned} \quad (1)$$

Usually, the match score is positive but the mismatch and the gap scores are negative. Therefore, more number of matches increases the alignment score but more number of gaps or mismatches decreases the alignment score. The scores of match, mismatch and gap are given as input parameters of the alignment algorithm.
The optimal number of matches, mismatches and gaps are computed using the Needleman-Wunsch algorithm [3] or Smith-Waterman [4] algorithm. In any algorithm, a scoring matrix of size "m x n" (m is the length of the query sequence and n is the length of the database sequence) is first formed. The optimal score at each matrix element is calculated by adding the current match score to previously scored positions and subtracting gap penalties. Each matrix element may have a positive, negative or 0 value. For two sequences, query (Q) and database (D)
$Q = a_1 a_2 .....a_m$
$D = b_1 b_2 .....b_n$
where $H_{ij} = T(a_1 a_2 .....a_m, b_1 b_2 .....b_n)$ then the element at the (i,j)th position of the matrix $H_{ij}$ is given by

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + S \\ H_{i-1,j} - G_x \\ H_{i,j-1} - G_y \end{cases} \quad (2)$$

Where $H_{ij}$ is the score at position i in the sequence Q and position j in the sequence D. S is the score of match or mismatch. Gx is the penalty for a gap of length x in the sequence Q and Gy is the penalty for a gap of length y in the sequence D. After the matrix is filled up, to determine an optimal alignment of the sequences from scoring matrix, a method called trace back is used. The trace back keeps track of the position in the scoring matrix that contributed to the highest overall score found. The positions may align or may be next to a gap, depending on the information in the trace back matrix. There may exist multiple maximal alignments.
The time required to get the optimal alignment for two sequences (the query sequence and just one sequence of the database) is proportional to the product of the lengths of the two sequences being compared, i.e, n x m steps.

### *B. SEQUENCE ALIGNMENT USING OUR TECHNIQUE*

The database of the sequence computing applications contains large number of sequences (as explained in Section I). To align the query sequence 'Q' with each sequence of the database 'D', we need to apply Needlman-Wunsch algorithm on each pair of sequences (as explained in Section II-A). Our technique is based on filtering the database such that the database sequences which are not similar (not close) to the query are excluded from the searching and the Neddleman-Wunsch algorithm is applied only on the database sequences which are similar (close) to the query (see right part of Fig. 2).
Our technique passes through the following four steps (see Fig. 3):

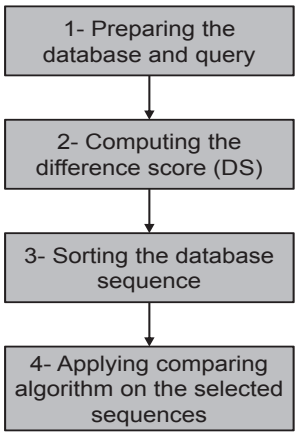


Fig. 3: Steps of our technique

**1- Preparing the database for the comparison:**
To prepare the database, we propose our new similarity measure, we call it similarity function. This function is based on the mathematical parameters: frequency and standard deviation of the alphabet codes for each database sequence. These codes have been created previously (as explained in Section II).
The frequency similarity function (Freq) is the number of repeated code in the sequence. It is an indicator for the similarity between two sequences. For instance, if the frequency of the code in a sequence is close to its frequency in another sequence, this is a good indicator that the two sequences might be similar.
To find the frequency for each code in the sequence, we scan the sequence starting from the first code till the last one using number of counters equal to the different codes. One counter for each code. Each counter is incremented by one when it meets new code of the same type. By the end of the scanning, all counters save their values beside the database sequence. The counter values beside any sequence refer to the frequency of codes types for that sequence.
The frequency similarity function does not give always correct results. For example, if the two sequences have the same (or close) number of codes frequencies but the codes are distributed in different way between the two sequences. In this case, the frequency score is not correct score to measure the similarity.
Therefore, we propose another similarity function which is the standard deviation.
The standard deviation function (STD) shows the distribution of the code in a sequence. It gives an idea of how close the entire code of a sequence to the average value. Code with large standard deviation has data spread out over a wide range of values. The standard deviation (STD) is defined

mathematically as:

$$STD = \sqrt{\frac{\sum_{i=1}^{i=n}(X_i - \bar{X})^2}{n-1}}$$

Where $\bar{X}$ is the the average value.

For each database sequence, the frequency and standard deviation functions for each code are computed and stored beside it. This step may take long time as the database includes large number of sequences, but it is done off-line, i.e., independent from the comparison process. Therefore, it does not matter how long time it takes because we do it only one time and prepare the database for future comparison process.

**2- Computing the difference score (DS):**
Once a query sequence needs to be searched in (aligned with) all database sequences, the technique computes the similarity functions "Freq" and "STD" of all different codes for it.
Usually, the sequence has more than one different codes. To find if there is similarity between two sequences each have different codes, the technique computes the frequency difference score (FDS) and the standard deviation difference score (DDS) and add them together to compute the difference score (DS) which reflects the similarity between two sequences.
Difference Score (DS) = frequency difference score (FDS) + standard deviation difference score (DDS)
The frequency difference score (FDS) is the sum of the absolute values of the differences between the two sequences for each code type. Mathematically, the frequency difference score (FDS) between the query sequence 'Q' and the database sequence 'D' is defined as following considering that both sequences have 'n' alphabet codes:

$$\begin{aligned} FDS \quad = \quad & |Freq\_code\_1(Q) - Freq\_code\_1(D)| \\ & + |Freq\_code\_2(Q) - Freq\_code\_2(D)| \\ & + |Freq\_code\_3(Q) - Freq\_code\_3(D)| \\ & + \ldots \\ & + \ldots \\ & + |Freq\_code\_n(Q) - Freq\_code\_n(D)| \end{aligned} \tag{3}$$

Where "$Freq\_code\_1(Q)$" is the frequency of code 1 in the query sequence. '"$Freq\_code\_1(D)$" is the frequency of code 1 in the database sequence, etc.
The standard deviation difference score (DDS) is the sum of the absolute values of the differences between the two sequences for each code type. Mathematically, the standard deviation difference score (DDS) between the query sequence 'Q' and the database sequence 'D' is defined as following considering that both sequences have 'n' alphabet codes:

$$\begin{aligned} DDS \quad = \quad & |Dev\_code\_1(Q) - Dev\_code\_1(D)| \\ & + |Dev\_code\_2(Q) - Dev\_code\_2(D)| \\ & + |Dev\_code\_3(Q) - Dev\_code\_3(D)| \\ & + \ldots \\ & + \ldots \\ & + |Dev\_code\_n(Q) - Dev\_code\_n(D)| \end{aligned} \tag{4}$$

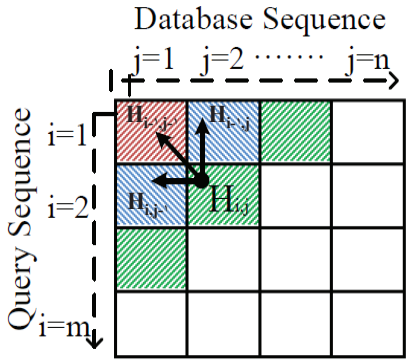


Fig. 4: Computation sequence of the similarity matrix. the score of the cells which have the same color are computed together

Where "$Dev\_code\_1(Q)$" is the standard deviation of code 1 in the query sequence. "$Dev\_code\_1(D)$" is the standard deviation of code 1 in the database sequence, etc.
Our technique is based on filtering the database such that the database sequences which are not similar (not close) to the query are excluded from the searching and the optimal or the heuristic alignment algorithm is applied only on the database sequences which are similar (close) to the query.

**3- Sorting the database sequences:**
In this step, the database sequences are sorted according to their difference score (DS), i.e. (FDS + DDS), such that the sequences which have low difference scores (more close to the query sequence) are shifted to the top of the database.

**4- Applying the Alignment Algorithm:**
In the last step, our technique applies the alignment algorithm (in our case the Needleman-Wunsch Algorithm) only on the sequences, which have the low difference scores (selected in the previous step). The sequences which have high difference scores will be excluded. This will provide the alignment in reasonable time because the alignment algorithm is applied only on a part of the database and not on whole of it.

## III. COMPLEXITY OF OUR TECHNIQUE

In this section, we compare between the complexity of the traditional methods and our technique. In case of the traditional methods, when Needleman-Wunsch Algorithm is used, the complexity is based on the length of the two sequences being compared and the number of sequences in the database. Let $m$, $n$ are the lengths of the query sequence Q and the database sequence D, respectively. As the Needleman-Wunsch Algorithm is based on dynamic programing, then the complexity to perform the alignment for one sequence is $O(m \times n)$. If $s$ is the number of sequences in the database, then the total complexity will be $O(m \times n \times s)$.
In case of our technique, assuming we have $c$ different codes. to compute the distribution of the $c$ codes in the query sequence, we need to scan the query along its length. If the length of the query sequence is $m$, then we need $m$ step to perform the scan. To compute the difference score (DS) between the query and one database sequence, we need $c$ steps to perform the subtraction for the $c$ codes and $c-1$ steps to sum up the results. If $s$ is the number of sequences in the database,

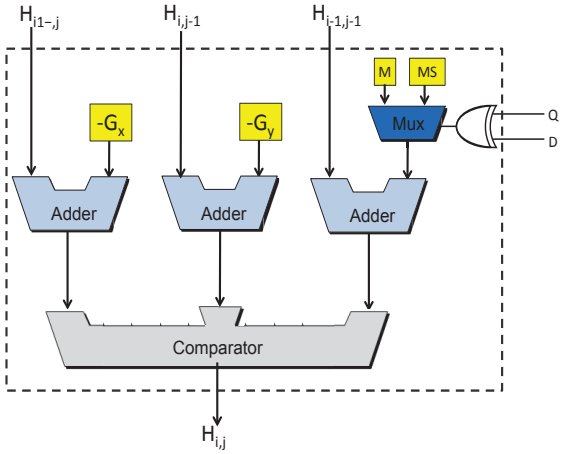


Fig. 5: The design implementation of our processing element for sequence alignment

then we need $((2c-1)\times s)$ steps to compute the difference score. To sort the $s$ difference scores from the smallest to the largest one using Heap sort or merge sort algorithm, we need $(s \times logs)$ steps.
Assuming that 50% of the database sequences are selected to apply Needleman- Wunsch Algorithm on them. To perform this step we need $m \times n \times s/2$ steps.
Consequently, the total steps of our technique is $((m+(2c-1)\times s+s\times logs)+(m\times n\times s/2))$, i.e., the complexity is $O(m\times n\times s/2)$.
For instance, if the length of the query m = 500, and the number of the database sequences s = 10000. Each sequence of the database has length n = 500. To align the query sequence with the database sequences using the traditional method, we need: 500 x 500 x 10000 = 2500000000 steps (2500 Million steps). Using our technique, and assuming that the data are codded with 4 different codes, we need:
500 + (7 x 10000) + (10000 x log 10000)+ (500 x 500 x 5000) = $\approx$ 1250 Million steps.
Using our technique we save 50% of the time required to align the sequences using the traditional methods.

## IV. Hardware Implementation

As explained in Section II-A, to implement the alignment algorithm, we need first to form a similarity matrix of size ”m x n” using the Equation 2 and then we need to trace the scores in the matrix back. The time complexity of the alignment algorithm is O(MN). To reduce this complexity, multiple entries of the matrix are calculated in parallel. From the equation 2, we can determine that the score of any cell, H(i,j), in the matrix depends on the scores of the three other elements (see Fig. 4): The left neighbor, H(i,j-1), the up neighbor, H(i-1,j), and the up-left neighbor, H(i-1,j-1). This means that the score of any cell can not be computed before computing the scores of cells located on the anti-diagonal positions. Fig. 4 shows the sequence of the cell computation in the similarity matrix. All the cells located on the same anti-diagonal positions (have the same color) are computed together (simultaneously) because they are independent of each other.
To measure the performance of the alignment implementation, the Cell Updates per Second (CUPS) metric is commonly used [8], which represents the time required to complete the computation for one cell of the similarity matrix. The total number of cell updates gives the implementation performance of the sequence alignment algorithm:

$$Performance\ (CUPS) = \frac{size(Query)\ x\ size(Database)}{Time\ to\ complete\ the\ computation} \quad (5)$$

We implement our technique using a FPGA-based linear systolic array to reduce the complexity order of the computation. A linear systolic array [21] is an array of processing cores where each cell shares its data with the other cells in the array. Each processing core solves a subproblem and shares the solution to all other cells in the array to prevent calculation of the same problem twice. Each anti-diagonal has M cells, and each cell can be updated in parallel, so the systolic array consists of M Processing Elements (PEs) that each computes a new value for the cell in the row that they are responsible for [19].
We design the processing element (PE) of systolic array using the FPGA logic components. The PE is used to build a systolic array architecture of any size. Fig 5 shows the design implementation of our processing element. Assuming the sequences we are going to align are DNA sequences. Each sequence consists of four codes (letters): A, C, G, T. In this case, the two sequences Q and D are encoded using two bits for each code (letter) as following:
A: 00, C: 01, G: 10, T: 11.
Based on the equation 2, the PE receives the values of the three neighbor elements, the left neighbor, H(i,j-1), the up neighbor, H(i-1,j), and the up-left neighbor, H(i-1,j-1). Each PE also receives one code (2 bits) from each of the two sequences (Q and D). The PE has four given parameters (marked in yellow blocks in the figure) which are fixed for all PEs. These parameters are the scores of match (M), mismatch (MS), gap in sequence Q ($G_x$), and gap in sequence D ($G_y$). The PE calculates the outcome of the matrix element H(i,j) using XOR gate, one multiplexer, three adders, and one comparator. The multiplexer chooses match or mismatch score based on the similarity of the two codes compared, and sends the score to the adder. The comparator selects the maximum value of the three adders according to Eq. 2.
We implement our design on a Xilinx Zynq-7000 All Programmable SoC (XC7Z020-CLG484) Artix-7 FPGA from Digilent [26]. Each PE utilizes 19 slices. The XC7Z020 has 13300 slices (1.3 M ASIC gates), i.e., we can fit a maximum of 700 PEs on this FPGA chip. In our experimental results, we utilize only 400 PEs as the sequences we align are of length 400 nucleotides, as explained in the next section.

## V. Experimental Results

In this section, the experimental results of our technique are presented. To evaluate our technique, DNA sequences of the database DNA Data Bank of Japan (ddbj) [24] are used. 100 sequences of BCT and CON divisions are selected as case study. Each sequence has length of 400 nucleotides. The accession numbers of the selected sequences start with the ID ”AB”, ”AF”, ”AJ”, ”AM”, ”AY” and ”DQ”. We compare our technique with the traditional methods which use the first widely used program for optimal sequence alignment

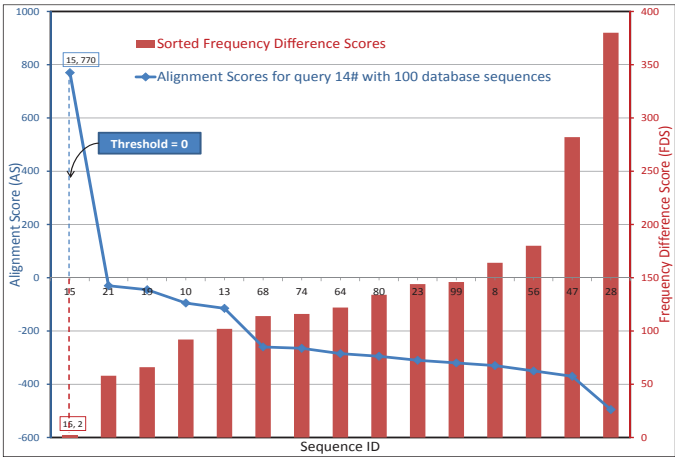


Fig. 6: Alignment and frequency difference scores for the query (sequence number 14) with 100 database sequences

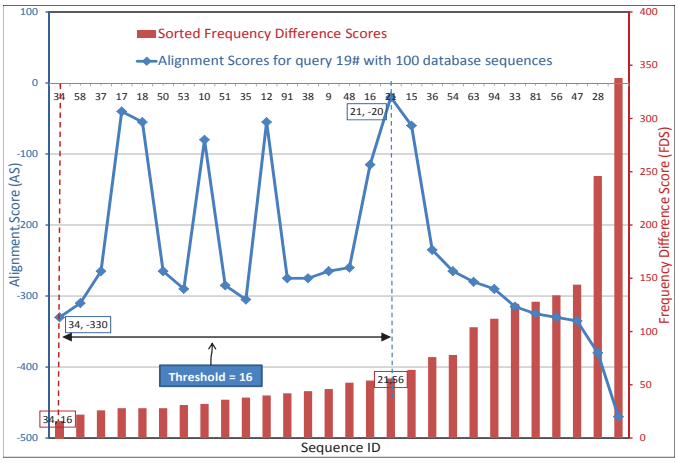


Fig. 7: Alignment and freuency difference scores for the query (sequence number 19) with 100 database sequences

Needleman-Wunsch Algorithm [25]. The score of match, mismatch, gap open, and gap extend are selected to be +2, -3, 0, -4, respectively.

As our database has 100 sequences, 100 cases are tested considering different query sequence for each case, i.e., in the first case, we consider the first sequence as query sequence and the remaining sequences as database. In the second case, the second sequence is considered as query sequence and the remaining sequences are considered as database, and so on. The results are presented for the three similarity functions, frequency (Freq), Standard Deviation (STD) and combination of both functions (Freq + STD):

**1- Using the frequency similarity function (Freq):**

The experimental results when our technique uses (Freq) are presented in Figures 6-9.

Figures 6, 7, and 8 show the results for three selected tested cases, case 14, case 19 and case 40, respectively. In these figures, the left y-axis shows the alignment score (AS) using Needleman-Wunsch Algorithm. The right y-axis shows the frequency difference score (FDS) of our technique. The x-axis shows the ID of the database sequences.

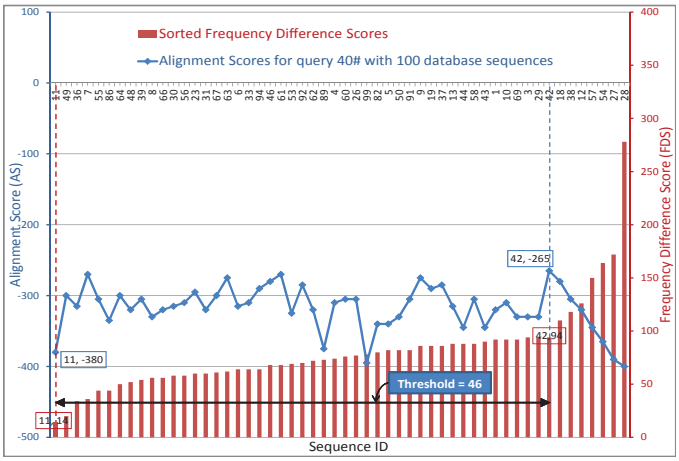


Fig. 8: Alignment and frequency difference scores for the query (sequence number 40) with 100 database sequences

The alignment score (AS) and the frequency difference score (FDS) in these figures are computed for the query sequence (sequence number 14 in Fig. 6, sequence number 19 in Fig. 7, and number 40 in Fig. 8) with each sequence of the database. Then, The sequences are sorted based on their frequency difference score in ascending form, i.e., from the sequence which has the smallest difference score to the one which has the highest score.

The common result in these three figures is, when the frequency difference score (FDS) increases across the sequences, the alignment score (AS) decreases (or vice versa). This result shows that the criterion we use in our technique, for selecting the sequences to which we may apply Needleman-Wunsch Algorithm on instead of the whole database sequences, is correct. This is because the sequences which have low difference scores have high alignment scores. In figures 6, 7, and 8, we define new parameter called ”Threshold”. The threshold for any test case is the number of sorted sequences between the sequence which has the lowest frequency difference score (FDS) and the highest alignment score (AS). In other words. The threshold refers to the number of sequences we need to apply Needlman-Wunsch Algorithm on them (using our technique) instead of applying it on the whole database sequences (using the traditional methods).

In Fig. 6, the frequency difference score (FDS) curve is marked with a red label ”15,2”. This label means that the minimum frequency difference score, ‘2’, occurs at the sequence number ‘15’. The alignment score curve is marked with the blue label ”15,770”. It means that the maximum alignment score ‘770’ occurs at the sequence number ‘15’. In other words, the sequence number ‘15’ has the lowest frequency difference score (FDS) and the highest alignment score (AS) and the threshold in this case is ‘0’. From this figure, we conclude that using our technique implies that applying Needlman-Wunsch Algorithm on the sequence number 15 is enough to get the optimal alignment instead of applying it on the whole database sequences (using the traditional methods). This is the best case we get, but unfortunately it is not always the same for all cases.

In Fig. 7, the lowest frequency difference score, ‘16’, occurs at the sequence number ‘34’ which has alignment score equal to ‘-330’ (this alignment score is not the highest one). On the

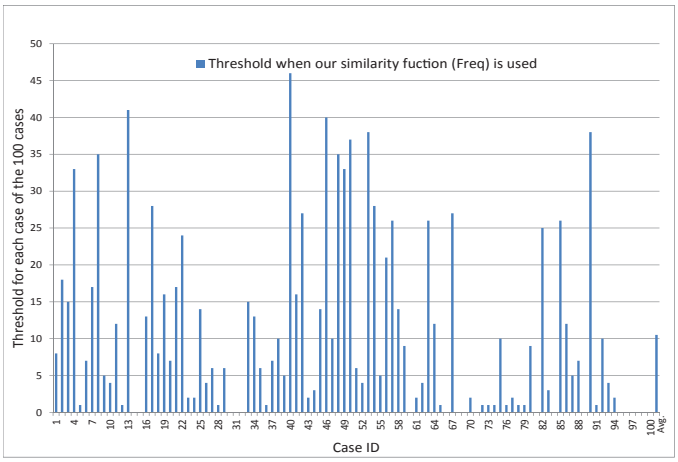


Fig. 9: The threshold for each case of the 100 cases when our similarity function (Freq) is used

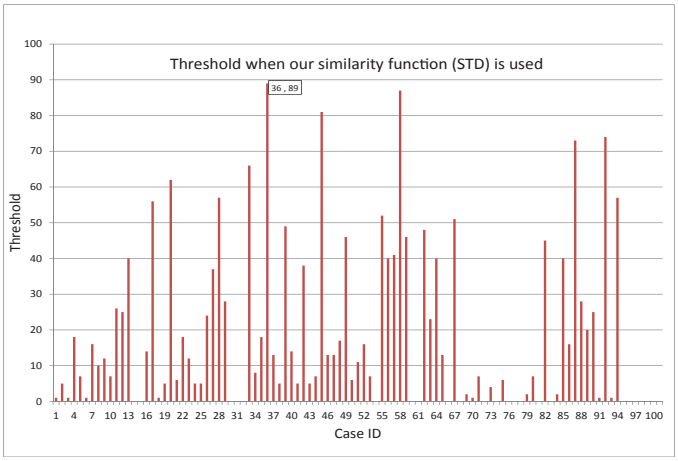


Fig. 10: The threshold for each case of the 100 cases when our similarity function (STD) is used

other hand, the highest alignment score, '-20', occurs at the sequence number '21' which has frequency difference score equal to '56'. The threshold in this case is '16'. In other words, we need to apply Needlman-Wunsch Algorithm on the lowest 16 frequency difference score sequences instead of applying it on the whole database sequences. In Fig. 8, the lowest frequency difference score, '14', occurs at the sequence number '11' which has alignment score equal to '-380' (this alignment score is not the highest one). On the other hand, the highest alignment score '-265' occurs at the sequence number '42' which has frequency difference score equal to '94'. The threshold in this case is '46', i.e., we need to apply Needlman-Wunsch Algorithm on the lowest 46 frequency difference score sequences instead of applying it on the whole database sequences. From the previous three cases, we notice that the threshold is not fixed. To find the maximum threshold, the test has to be done for all cases. Fig. 9 shows the thresholds for all 100 cases. The last bar shows the average threshold through all cases which is equal to '10.5'. In this figure, all 100 cases are tested to find the highest alignment score and the lowest frequency difference score for each case (one case for each different query), and then the threshold were computed. From Fig. 9, we notice that the threshold differs from one case to another one based on the query sequence. And, the maximum threshold among all other cases is '46' which appears in the case number 40 (as shown in Fig. 8). This is the worst case in which we need to apply Needleman-Wunsch Algorithm on 46 sequences. In other words, when our technique is applied only on the top 46% of the database sequences (or in general case 50%), then the maximum AS, in each case, will be included in this top part, i.e., applying Needleman-Wunsch Algorithm on this 50% of the database sequences will be enough to find the maximum alignment score instead of applying the algorithm on the whole database sequences as done by traditional methods.

**2- Using the standard deviation similarity function (STD):**
If the two sequences being compared/aligned have the same (or close) number of codes frequency but the codes are distributed in different way between the two sequences, then, the frequency difference score (FDS) will not be correct score

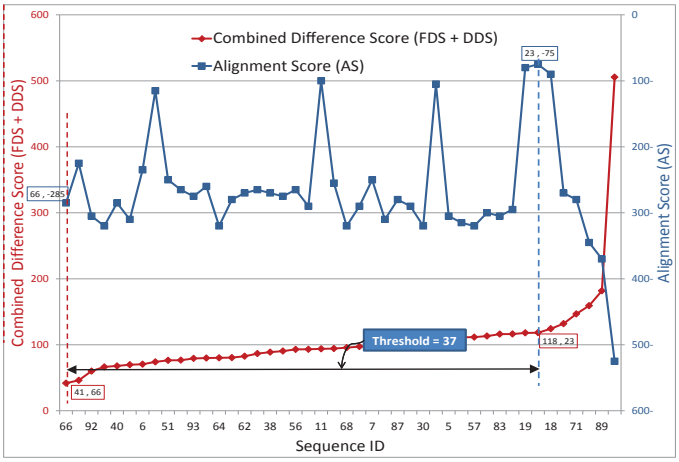


Fig. 11: The worst case threshold (the query is the sequence number 13) when our combined similarity functions (Freq + STD) is used

to measure the similarity. Therefore, we use the standard deviation similarity function (STD).
Figure 10 shows the thresholds for all 100 cases when our technique uses the STD similarity function. In this figure, the maximum threshold (worst case) among all other cases is '89' which appears in the case number 36. This is the worst case in which we need to apply Needleman-Wunsch Algorithm on 89 sequences which is not worthy. From this figure, we can conclude that using the standard deviation similarity function (STD) to measure the similarity is not also a good idea because the two sequences may have the same (or close) standard deviation (STD) but the code frequency for each sequence is different. Therefore, a combination of the two similarity functions (Freq) and (STD) can give better results as shown in the following section.

**3- Using the combination of frequency (Freq) and standard deviation similarity functions (STD):**
The experimental results when our technique uses the combination of the two similarity functions (Freq+STD) are presented in Figures 11-15.

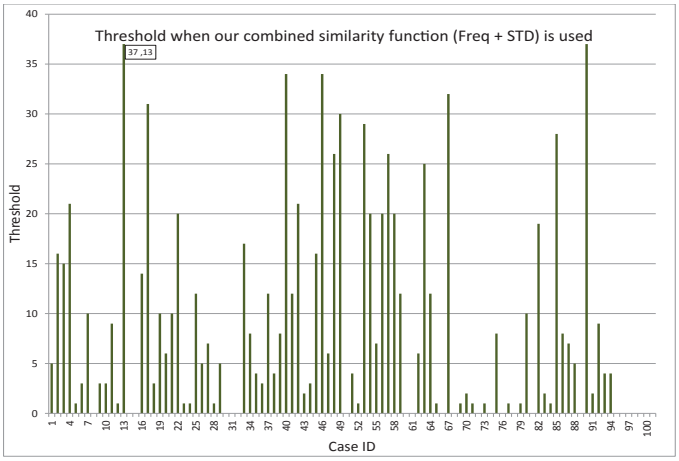


Fig. 12: The threshold for each case of the 100 cases when our combined similarity function (Freq + STD) is used

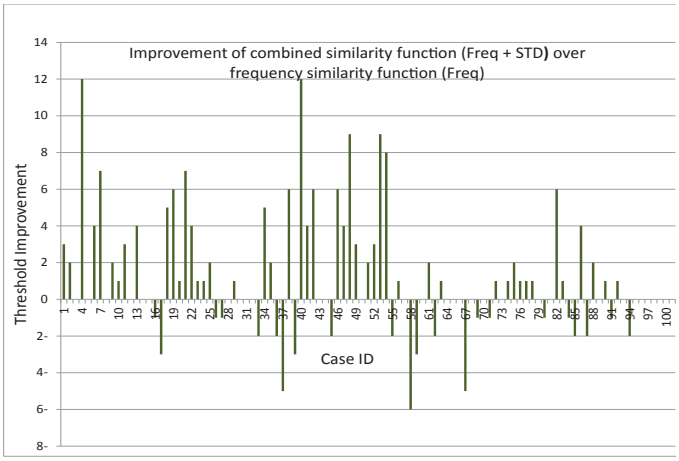


Fig. 13: Improvement of the threshold when the combined similarity functions (Freq + STD) is used instead of the sole frequency similarity function (Freq)

Figure 11 shows The results, for the query (sequence number 14) with 100 database sequences, when our combined similarity function (Freq + STD) is used. In this figure, the right y-axis shows the alignment score (AS) using Needleman-Wunsch Algorithm. The left y-axis shows the combined difference score (FDS + DDS) of our technique. The x-axis shows the ID of the database sequences. The combined difference score (FDS + DDS) curve, in this figure, is marked with a red label ”66, 41”. This label means that the minimum (FDS + DDS) difference score, ‘41’, occurs at the sequence ‘66’. The alignment score (AS) curve is marked with a blue label ”23,-75”. It means that the maximum alignment score ‘-75’ occurs at the sequence ‘32’. The number of sorted sequences on the x-axis which are located between the minimum frequency difference score sequence and the maximum alignment sequence (i.e. ”Threshold”) is 37.

Figure 12 shows the thresholds for all 100 cases when our technique uses the combined similarity function (Freq + STD). In this figure, the maximum threshold (worst case) among all other cases is ‘37’, which appears in the case number 13. This means, when our technique is applied only on the top 37% of the database sequences, then the maximum AS, in any query case, will be included in this top part, i.e., applying Needleman-Wunsch Algorithm on this 37% of the database sequences will be enough to find the maximum alignment score instead of applying the algorithm on the whole database sequences as done by traditional methods. Using the combined similarity function (Freq + STD) gives better results than using the sole frequency function (Freq). Figure 13 shows the threshold improvement when the combination is used. In this figure, 47 cases are improved (bars located in the positive area). The maximum improvement is ‘12’ which appears in the case number 40 (the threshold of this case number is ‘46’ for frequency similarity function (Freq) and it becomes ‘34’ for the combined similarity function (Freq + STD)). There are 32 cases where the threshold is ’0’ using the (Freq) function and they remain the same in the (Freq + STD) function. The remaining 21 cases are changed negatively (bars located in the negative area). When our technique applies Needleman-Wunsch Algorithm on less than 37 sequences of the database and repeated for 100 cases (each case with different query),

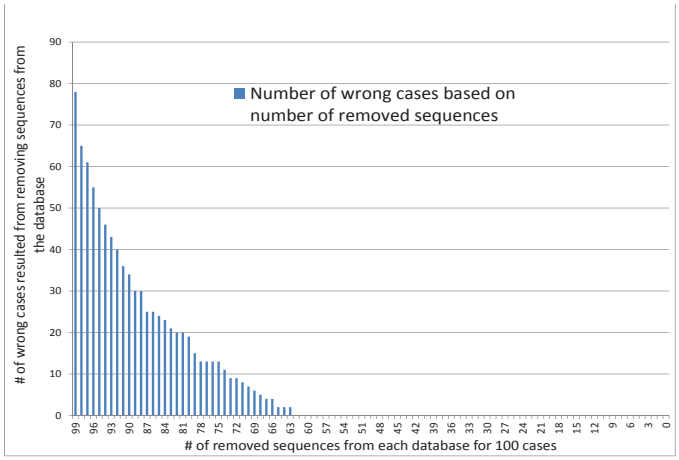


Fig. 14: The error rate resulted from removing sequences from the database

then the result will not be correct for all the 100 cases, i.e., the sequence which has the lowest difference score is not the same as the sequence which has highest alignment score. The results differ based on the number of removed sequences from the database.

Fig. 14 shows the error rate resulted from removing sequences from the database. The x-axis shows the number of removed sequences from each database for 100 cases (for clarity, we do not show all 100 cases). The y-axis shows the number of wrong cases resulted from removing sequences from the database. For example, when 99 sequences are removed from the database and our technique is repeated for the 100 cases, there will be ‘78’ wrong cases and only ‘12’ cases will have correct results, i.e., in each case of the 12 cases, the the sequence, which has the lowest difference score, has the highest alignment score. When the number of removed sequences decreases, the error rate will be decreased and the number of correct cases will be increased. When the number of removed sequences is ‘63’, i.e., only ‘37’ sequences are remained in the database, there will be no wrong cases. This is the best case in terms of the

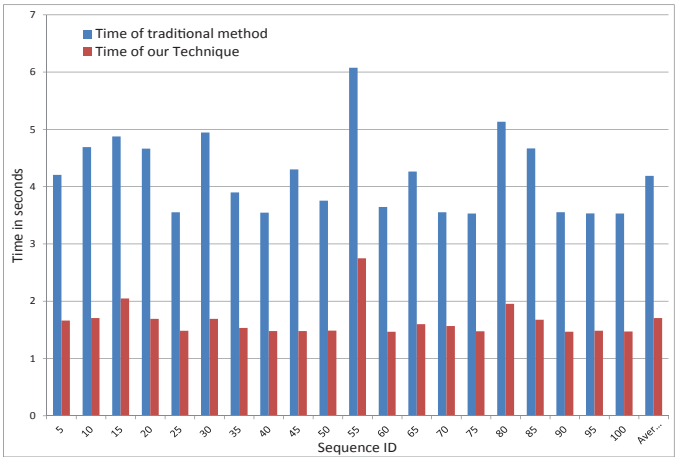


Fig. 15: Execution time comparison between traditional methods and our technique

size of the database and the execution time. Removing less number of sequences will not effect on the result but negatively will increase the size of the database and consequently the time required to analyze it. Fig. 15 shows comparison between the execution time of traditional methods and our technique. In this figure, The x-axis shows the 100 cases (not all cases are shown for clarity purpose). For each case, different query sequence used to be aligned with the remaining sequences of the database. The y-axis shows the execution time required for each case. The blue bar shows the time for traditional methods which apply NW algorithm on whole database sequences while the red one shows the time for our technique which applies NW algorithm on selected 37% of the database sequences. The last bars show the average time through all 100 cases.
In this figure, the execution time using our technique is 63% improved in comparison to the execution time required using the traditional methods. (the average time for traditional methods is 4.18 sec. while the average time for our technique is 1.8 sec.). This result we got because we have excluded selected 63% of the sequences from the process of applying Needleman-Wunsch Algoritm by using our technique.

## VI. CONCLUSIONS

We have presented new technique to accelerate the database sequence alignment. Our technique has the advantage of the heuristic and deterministic algorithms that can delivers optimal and fast solution We compared our technique with the traditional methods which apply the alignment algorithms on the whole database sequences and showed that our technique saves almost 63% of the time required to perform the sequence comparing. Merging our technique with the state-of-the-art database computing technique may further improve the execution time.

## REFERENCES

[1] European Bioinformatics Institute Home Page, FASTA searching program, 2003. http://www.ebi.ac.uk/fasta33/.

[2] National Center for Biotechnology Information. NCBI BLAST home page, 2003. http://www.ncbi.nlm.nih.gov/blast.

[3] S. Needleman and C. A. Wunsch. General method applicable to the search for similarities in the amino acid sequence of two sequences. Journal of Molecular Biology. Pages 443453, 1970

[4] T. F. Smith and M. S. Watermann. Identification of common molecular subsequence. Journal of Molecular Biology. Pages 196197, 1981

[5] G. Kondrak. Algorithms for Language Reconstruction. Ph.D. thesis, University of Toronto. 2002

[6] Farrar M: Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics 2007, 23:156-161

[7] A. Stivala, P.J. Stuckey, M.G. de la Banda, M. Hermenegildo, and A. Wirth. Lockfree parallel dynamic programming. Journal of Parallel and Distributed Computing, 70(8):839-848, 2010.

[8] Lukasz Ligowski and Witold Rudnicki. An efficient implementation of Smith-Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In IEEE International Symposium on Parallel & Distributed Processing. pp. 1-8. 2009.

[9] S. J. Greenhill, Q. D. Atkinson, A. Meade and R. D. Gray. The shape and tempo of language evolution. In proceedings of the Royal Society. Pages 2443-2450, 2010.

[10] Bonny, T., M. A. Z. and Salama, K. N. An adaptive hybrid multiprocessor technique for bioinformatics sequence alignment. In the International Conference on Biomedical Engineering. pages 112-115, 2010

[11] Z. Nawaz, M. Nadeem, J. van Someren, and K.L.M. Bertels. A parallel fpga design of the smith-waterman traceback. In Field-Programmable Technology (FPT), 2010 International Conference on, pages 454-459, Beijing, China, December 2010.

[12] T. Rognes. Faster smith-waterman database searches with inter-sequence simd parallelisation. BMC Bioinformatics, 12(1):221, 2011.

[13] Talal Bonny and Khaled Salama, Fast Global Sequence Alignment Algorithm in the Asilomar Conference on Signals, Systems, and Computers to be held in PACIFIC GROVE, CA, in November 6-9th, 2011

[14] Talal Bonny and Khaled Salama, ABS: Sequence Alignment By Scanning in the 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'11), on August 30 - September 3, 2011. Boston, MA, USA

[15] S. Nelson-Sathi, J. List, et. al. Networks uncover hidden lexical borrowing in Indo-European language evolution. In proceedings of the Royal Society. Pages 1794-1803, 2011

[16] M. Affan Zidan, T. B. and Salama, K. N. High performance technique for database applications using a hybrid gpu/cpu platform. IEEE/ACM 21st Great Lake Symposium on VLSI. pages 8590, 2011

[17] A. Chakraborty and S. Bandyopadhyay. Clustering of web sessions by FOGSAA. In IEEE Recent Advances in Intelligent Computational Systems (RAICS). Pages 282-287. 2013

[18] E. F. de O.Sandes and A.C.M.A. de Melo. Retrieving smith-waterman alignments with optimizations for megabase biological sequences using gpu. Parallel and Dis- tributed Systems, IEEE Transactions on, 24(5):1009-1021, 2013.

[19] H. Shah, L. Hasan, and N. Ahmad. An Optimized and Low-cost FPGA-based DNA Sequence Alignment. In the 35th Annual International Conference of the IEEE EMBS. Pages 2696-2699. 2013

[20] S. Kim, Y. J. Yoo, J. So, J. G. Lee and J. Kim., Design and Implementation of Binary File Similarity Evaluation System. International Journal of Multimedia and Ubiquitous Engineering, Vol.9, No.1. Pages 1-10, 2014

[21] J.M. Marmolejo-Tejada, V. Trujillo-Olaya, C.P. Renteria-Mejia and J. Velasco-Medina. Hardware implementation of the Smith-Waterman algorithm using a systolic architecture. In IEEE 5th Latin American Symposium on Circuits and Systems (LASCAS). Pages 1-4, 2014

[22] Manal Al Ghamdi and Yoshihiko Gotoh. Alignment of nearly-repetitive contents in a video stream with manifold embedding. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Pages 1255-1259, 2014

[23] P. Wang, V. Eglin, C. Largeron, J. Llads, A. Fornes And C. Garcia. A Novel Learning-free Word Spotting Approach Based On Graph Representation. In 11th IAPR International Workshop on Document Analysis Systems (DAS). Pages 207-211, 2014

[24] Website: http://www.ddbj.nig.ac.jp/

[25] Website: blast.ncbi.nlm.nih.gov/Blast.cgi

[26] Website: www.digilentinc.com/