RBT–L: A LOCATION BASED APPROACH FOR SOLVING THE MULTIPLE SEQUENCE ALIGNMENT PROBLEM
TECHNICAL REPORT 626

JAVID TAHERI, ALBERT Y. ZOMAYA,
AND BING BING ZHOU
SCHOOL OF INFORMATION TECHNOLOGIES
THE UNIVERSITY OF SYDNEY

JULY 2008
RBT–L: A Location Based Approach for Solving the Multiple Sequence Alignment Problem

Javid Taheri, Albert Y. Zomaya1, and Bing Bing Zhou

School of Information Technologies, J12, The University of Sydney,
Sydney, NSW 2006, Australia.

{J.Taheri, A.Zomaya, B.Zhou}@usyd.edu.au

1 Also affiliated with Sydney Bioinformatics Centre and the Centre for Mathematical Biology, the University of Sydney, Sydney, NSW 2006, Australia.
Abstract — This paper presents a novel approach to solve the Multiple Sequence Alignment (MSA) problem. The Rubber Band Technique: Location Base (RBT-L) introduced in this paper, is inspired by the elastic behavior of a Rubber Band (RB) on a plate with poles. RBT-L is an iterative optimization algorithm designed and implemented to find the optimal alignment for a set of input Protein sequences. In this technique, the alignment answer of the MSA problem is modeled as a RB, while the answer space is modeled as the plate with several poles resembling locations in the input sequences that are most likely to be correlated and/or biologically related. Fixing the head and tail of the RB at two corners of this plate, the RB is free to bend and finds its best configuration, yielding the best answer for the MSA problem. RBT-L is tested with one of the well-known benchmarks (BALiBASE 2.0) in this field. The obtained results show the superiority of the proposed technique even in the case of formidable sequences.

Keywords: Algorithms, Multiple Sequence Alignment, optimization.

1. INTRODUCTION

Almost half a century ago, in 1953, based on X-ray diffraction images taken by Rosalind Franklin, James D. Watson and Francis Crick suggested the first, now accepted, structure of the DNA (Watson and Crick 1953). Since that time, biologists always have tried to understand the basic roles of nucleotides and genes. One popular approach in trying to understand the function of a newly found gene or protein is to compare it with already known genes/sequences. As a result, it is now a very common practice to attempt to find one or more sequences in existing literature or databases that are reasonably close to the sequence in question. However, due to the fact that the number of known sequences is rapidly increasing every year, new classes of algorithms that are able to search massive data sets are needed.
Sequence Alignment algorithms are techniques that are used to find similarity among several DNA/Protein sequences. These algorithms are classified into two main categories: Pair-wise and MSA algorithms, each designed for special purposes. In Pair-wise algorithms, the main goal is to find the similar or closely related parts (motifs) of two sequences; whereas in MSA, the main goal is to find the consensus parts of more than two sequences. Therefore, Pair-wise algorithms are mainly used to find similar sequences in a database; MSAs are mainly used to find the relationship among several sequences.

Several algorithms and techniques have already been suggested to solve this problem in each of the above two categories. Among these techniques, there exist several classical methods, like Dynamic Programming, that can always find the optimal alignment for any two sequences (Pair-wise). However, these techniques cannot always be generalized to MSA cases (due to the excessive computation that is incurred after the addition of each extra sequence). Therefore, using classical methods in the MSA case is almost impossible. In fact, because it has been shown that MSA is NP-Complete (Wang and Jiang 1994), heuristics are mainly used to solve this problem.

Regardless of the solving technique, MSAs can be categorized into three main solution categories: exact, progressive and iterative (Abdesslem et al. 2006). In exact methods, which are usually the generalized methods of the Needleman and Wunsch algorithm (Needleman and Wunsch 1970), all sequences are aligned simultaneously to find the optimal answer. The main drawback of this class of algorithms is their massive computational need, usually impossible to find the answer in polynomial time. In progressive algorithms, sequences are first aligned two-by-two (using an appropriate Pair-wise algorithm) before finding the final alignment. Then, an alignment guidance tree is generated based on these Pair-wise alignment scores. Starting from
the two closest ones, sequences are combined step by step to find the optimal answer. In this case, current sequences are modified to get the best fit for new combining sequences. Although this class of algorithms usually manages to find reasonable alignments (especially to generate phylogeny trees), their main disadvantage is their sensitivity to the local minima as they easily get trapped by them. Despite progressive approaches, in the iterative methods, all sequences are aligned simultaneously. Here, using one or more heuristic algorithms, an initial answer is calculated first. Then, this initial answer is improved iteratively using intelligent routines designed for this type of MSAs. Although these algorithms are not as sensitive as progressive algorithm to the local minima, they have their own drawbacks. For example, achieving a reasonable final answer for these algorithms is greatly related to their initial answers.

Based on these methods, a number of alignment algorithms are designed to solve the MSA problem, such as MULTALIGN (Barton and Sternberg 1987), MULTAL (Taylor 1988), PILEUP (Devereux et al. 1984) and CLUSTALX (Thompson et al. 1997), which provides a graphical interface for CLUSTALW (Thompson et al. 1994). They all use a global alignment algorithm (Needleman and Wunsch 1970) to construct an alignment of the entire length of the sequences. Their main difference is in the order they combine the input sequences. MULTAL deploys a sequential branching method to align the two closest sequences before building up the final alignment by subsequently aligning the next closest sequence to it. MULTALIGN and PILEUP construct a guide tree using UPGMA (Sneath and Sokal 1973). This tree is then used to align larger and larger groups of input sequences. CLUSTALX that uses the alternative Neighbor-Joining algorithm (Saitou and Nei 1987) to construct a guide tree has one of the most sophisticated scoring systems. It considers sequence weighting, position dependant gap penalties, and the automatic switching among scoring matrices based on the degree of similarity among the
input sequences. PIMA (Smith and Smith 1992) uses a local DP algorithm to align only the most conserved motifs. Two versions of this method, ML_PIMA and SB_PIMA, are different based on their order of combining input sequences, maximum linkage and sequential branching algorithms, respectively. DIALIGN (Morgenstein et al. 1996) focuses on a local alignment based on segment-to-segment comparison to construct the final alignment. Then, an iterative procedure is deployed to combine these segments toward generating the final alignment. PRRP (Gotoh 1996) iteratively divides the input sequences into two groups and then subsequently realign them using a global group-to-group alignment algorithm. SAGA (Notredame and Higgins 1996) involves evolving a population of alignments in a quasi evolutionary manner to gradually improve their fitness. MAFFT (Katoh et al. 2002) identifies the homologous regions by a Fast Fourier Transform (FFT) approach. Using its simplified scoring matrix, MAFFT manages to significantly reduce the CPU time and increases the accuracy of alignments even for sequences having large insertions and extensions as well as distantly related sequences of similar length. ProbCons (Do et al. 2005), which computes posterior-probability matrices and expected accuracies for each Pair-wise comparison, applies the probabilistic consistency transformation, and then computes an expected accuracy guide tree to progressively generate the final alignment. T-Coffee (Notredame et al. 2000) pre-processes a data set of all pair-wise alignments between the input sequences to generate a guide tree for the progressive alignment. T-Coffee not only does focus on the next aligned sequences but also on the whole set of input sequences. MUSCLE (Edgar 2004) as one of the very fast algorithms in this field has three stages: draft progressive, improved progressive, and refinement. At the completion of each stage, a multiple alignment is available and the algorithm can be terminated. The first stage builds a progressive alignment, the second stage that might be iterated attempts to improve the tree and builds a new progressive
alignment according to this tree, and, the third stage performs iterative refinement using a variant of tree-dependent restricted partitioning. MUMMALS (Pei and Grishin 2006) uses probabilistic consistency and improves its alignment quality by using Pair-wise alignment Hidden Markov models (HMMs). Parameters for such models have been estimated from a large library of structure-based alignments. There are also other HMMs methods that use statistical models of the primary structure consensus to align input sequences (Baldi et al. 1994; Krogh et al. 1994). HMMT (Eddy 1995) uses a simulated annealing method to maximize the probability that an HMM represents the sequences to be aligned.

In this paper, a novel approach, RBT-L, is presented to solve the MSA problem. The rest of the paper is structured as follows. Section 2 described the problem statement. Section 3 explains the main concepts of the innovated approach presented in this paper. In section 4, it is described in details how this technique is used to solve the MSA. Section 5 represents the simulation results. Discussion and analysis followed by conclusion are represented in sections 6 and 7, respectively.

2. MULTIPLE SEQUENCE ALIGNMENT

Let \( \{S_1, S_2, \ldots, S_N\} \) be \( N \) sequences over the alphabet set \( \Psi \), which contains 4 and 20 characters for DNA and Proteins sequences, respectively. Also, let \( \Psi' = \Psi \cup \{-\} \) be the superset of \( \Psi \) with and extra character for a ‘gap’. The MSA problem can be defined as finding \( \{S'_1, S'_2, \ldots, S'_N\} \) with the following properties:

1. \( S'_i = S_i \) for all \( i = 1,2,\ldots,N \) providing that all gaps are removed from \( S'_i \).
2. \( |S'_1| = |S'_2| = \cdots = |S'_N| \) where \( |S'_i| \) denotes the length of \( S'_i \).
3. The alignment score, \( \text{Score}(S') = \sum_i \sum_j \text{sim}(S'_i, S'_j) - \sum_i g(S'_i) \) is maximized where 

\[ \text{sim}(S'_i, S'_j) \] denotes a quotation of similarity between \( S'_i \) and \( S'_j \), and, \( g(S'_i) \) is related to gaps of \( S'_i \).

Based on these properties, the MSA is defined as an optimization problem. However, regarding the fact that the complexity of the stated problem is exponentially increased by adding every extra sequence to the input sequences set, finding the optimal answer is not always possible. This is why classical methods like DP and Needleman’s algorithm can be applied only for small number of short sequences.

3. **Rubber Band Technique**

The Rubber Band Technique: Location Base (RBT-L) is an iterative heuristic to solve the MSA. In this approach, which is inspired by the general behavior of a rubber band on a plate with poles, an initial answer is generated before launching the main optimization procedure. Using several operators, this initial answer is modified iteratively to obtain better alignment scores. In this paper, to simplify the explanation of these operators, they are all explained in the two-dimensional space (the Pair-wise case), although they can be easily generalized to the multi-dimensional space by allocating an extra dimension for every extra input sequence. The following definitions are essential before describing the main optimization procedure.

**A. Grid Answer Space**

The Grid Answer Space (GAS), which is the extended version of the grid table used in DP for Pair-wise alignment, is a multi-dimensional table with each sequence in one of its axes. In each GAS, the total number of grid points is related to the length of its input sequences and is equal to
\((|S_1| + 1) \times (|S_2| + 1) \times \cdots \times (|S_N| + 1)\). Fig. (1) shows a sample GAS for the Pair-wise alignment case of \{TWACTFW, TTWTFW\} sequences. The arrowed line shown in this figure represents the answer of this Pair-wise alignment.

The use of this table provides a unique one-to-one relationship between any possible answer of a MSA and its associated arrowed line. However, to satisfy conditions (1) and (2) of a MSA the following conditions must be met:

1. Each arrowed line must start from the upper left corner \((0,0)\) and finishes at the lower right corner \((|S_1| + 1, |S_2| + 1)\).

2. There cannot be any backward section in this arrowed line. That is, each section can only be diagonal or parallel to the one of the table’s axes.

B. Rubber Band
Any answer for a MSA can be presented by one and only one arrowed line. This unique arrowed line for each answer is called a Rubber Band (RB) as shown in bold in Fig. (1).

C. Primary Pole
A Primary Pole (PP) refers to a fixed point in GAS that the RB is ought to pass by. In fact, PPs are the sections of the GAS that force the optimization procedure to align a predefined number of characters (for each sequence) with each other. Although the RB is free to choose any set of grid points between each two PPs, it is not allowed to miss any of them. For example, Fig. (2) shows two PPs of an alignment problem. Three possible RBs for this alignment are shown in Figs. (2a)–(2c). As it can be seen, they all pass through these two PPs regardless of their other grid points. The sequence alignment answers of these three RBs are also shown in this figure. Note
that, the two PPs in this figure force the final alignment to align the marked areas in these answers.

**D. Secondary Pole**
Secondary Poles refer to grid points in GAS that a RB passes one at a time to generate the final answer. Despite PPs that can be far apart from each other, secondary poles are adjacent to each other. This type of poles is only used to connect PPs to each other. For brevity purposes, secondary poles are referred to as ‘poles’ for the rest of this paper.

**E. Primary Pole Score**
As described earlier, each PP points out predefined locations of input sequences that need to be aligned with each other. If the characters regarding these locations are augmented in a single string, the Primary Pole Score for that particulate PP is defined as the alignment score of that augmented string (with respect to the scoring matrix used for the whole alignment). For example, based on BLOSUM62 scoring matrix, the Primary Pole Scores of the Poles #1 and #2 in Fig. (2) are 11 and 6, respectively.

**F. Sticky Poles**
Sticky Poles (SPs) are imaginary poles in the system related to possible locations in a GAS with potentially high Primary Pole Scores. That is, the optimization procedure can have a pole with a high Primary Pole Score if it places a PP on that special place. Each Sticky Pole is located in a GAS to represent a column from the input sequences with a high score, resembling aligning identical or closely related nucleotides from input sequences with each other. SP of the alignment problem in Figs. (1) and (2) are given in Fig. (3). Here, a Pole is considered ‘sticky’ if its score is higher than 4 with respect to the BLOSUM62 scoring matrix.
G. Alignment Score
In each MSA problem, an Alignment Score is defined to evaluate the quality of each answer. The Sum-of-Pairs Score (SPS) with Penalized Gap Opening is the criterion used in this approach. In SPS, each column in an alignment is scored by summing the scores of all pairs of characters in that column. The score of the final alignment is then summed over all column scores. In the Penalized Gap Opening scheme, there are two factors to calculate the score/cost of a gap: opening and extension. Gap opening applied for each gap once and the gap extension corresponds to the length of each gap. The cost of a gap opening is usually considered to be 5 to 10 times more than that of a gap extension. Considering two factors for a gap is related to a well-known biological fact that having few long gaps is more probable than having several short gaps in an alignment.

4. RUBBER BAND TECHNIQUE IN MULTIPLE SEQUENCE ALIGNMENT
This section presents the details of the proposed algorithm for the solution of the MSA problem. The following sections are main procedures/routines used in this optimization process.

A. Locate Sticky Poles
As explained earlier, a SP points out to a location in the GAS that if a PP is placed in that location, it will get a high Pole Score. In fact, these locations are positions in the alignment to align identical or closely related characters from different input sequences to each other. However, simulation shows that, not all of them are possible from the biological point of view, and hence, an acceptable/feasible point as a SP. For example, for the sample alignment in Fig. (1), it is more probable that the first and second ‘W’ s in sequence No.1 are related to the first and second ‘W’ s in sequence No.2, respectively. Therefore, assigning SPs to relate the first ‘W’ in sequence No.1 to the second ‘W’ in sequence No.2, and vice versa, is considered to be
unnecessary. Furthermore, by not considering these kinds of unfeasible PPs, not only is the calculation time reduced, but also several inappropriate SPs that can mislead the final alignment to local minima are eliminated.

In this approach (RBT-L), the Amino Acid (AA) locations are used to separate a biologically meaningful SP from others. To generalize this hypothesis, a routine is designed to assign SPs to align similar AAs from different input sequences to each other (based on their position in different proteins); i.e., similar AAs (like ‘W’ in Fig. (1)) from the beginning/middle/end of sequence No.1 to the similar AAs from the beginning/middle/end of sequence No.2 and etc.

In the following procedure.

Step 1: Generate an Amino Acid Index Table (AAIT) for all input sequences.

Step 2: Generate SPs based on aligning wideness.

To generate the AAIT, each input sequence is scanned once and the index of AAs that are similar is stored in a database as shown in Fig. (4). In AAIT, each column is assigned to an input sequence. In each column, there exist 20 rows for the 20 different AAs. The array attached to each row (R) of column (C) contains the indexes of all AAs in sequence No.C that are similar to AAs No.R based on the alignment scoring matrix. For example if the similarity factor is set to 2 and the BLOSUM62 is the alignment scoring matrix, then the index of all ‘D’, ‘Q’, and ‘E’ AAs will be stored in row No.7 that is assigned for AA ‘E’ for each input sequence/column.

After generating AAIT, arrays of each row are expanded by finding appropriate AAs in different proteins to have identical lengths; because, same row for different columns may contain different number of elements, as they represent the number of a particular AA (and closely related AAs)
in different input sequences. To do this, a search algorithm is launched to find all SPs that are approximately at the same positions in different proteins (regarding their individual lengths). This procedure starts from the longest array in each row. For each element on the longest array, which contains the location of a specific AA in a protein, this procedure finds the closest index in other arrays regarding their array lengths. If these different indexes are close enough to each other (represented by factor called ‘Closeness’ and compare to preset threshold, $\gamma$), their indexes are stored as a PP, otherwise their relative indexes are stores. For example, assume one row of the AAIT is as Fig. (5a). For the marked AA #14 in sequence No.1, the relative index in sequence No.2 will be $(14/45) \times 42 = 13.07$. Because, the closest index in sequence No.2 to 13.07 is 10, the $Closeness = |13.07 - 10|/40 = 0.073$. Here, if ($\gamma = 0.15$), this index is accepted from sequence No.2, i.e., AA #14 from sequence No.1 is aligned to AA #10 from sequence No.2. Having the similar calculation for sequence No.6 yields to a different scenario. In this case, relative index for AA#14 from sequence No.1 will be $(14/45) \times 37 = 11.51$. Here, closest AA is 20; therefore, $Closeness = |11.51 - 20|/37 = 0.230$ that is greater than $\gamma(= 0.15)$. Therefore, it is assumed that AA #14 from sequence No.1 is aligned to AA #12 from sequence No.6 instead. Fig. (5b) shows the generated SP after performing the described procedure for all the elements of this example. Elements marked in gray represents indexes that did not pass the closeness test; thus, their relative index is used instead.

B. Initialize Primary Poles
After performing the procedure explained in the previous section for all rows of AAIT, several PPs (depending on the initial configuration of the algorithm) are selected as initial PPs to the start RBT-L algorithm with. The way by which these PPs are selected plays a major role in achieving a reasonable solution for the alignment in question. However, based on the complex
nature of the MSA problem, starting from appropriate PPs does not always guarantee a reasonable solution, but it usually confirms it. Therefore, two methods are designed to select the initial PPs as the initial phase of the algorithm: ‘random walk’ and ‘homogenous’. Random walk is mainly focused on aligning large segments of the input sequences with each other regardless of their position in the input sequences, for example aligning the beginning part of one sequence to the middle part of another. In contrast, in the homogenous mode, the main idea is to select PPs so that the beginning, middle, and end part of the input sequences are aligned with each other, respectively. As a result, random walk is more focused on local alignment in contrast to homogenous mode that emphasizes more on the global alignment.

The procedure for the random walk PP selection is as follows.

1. **Step 1:** Locate Sticky Poles in the GAS, namely $\text{ArrStickyPoles}$.
2. **Step 2:** Sort $\text{ArrStickyPoles}$ according to their location.
3. **Step 3:** Let $\text{ArrPrimaryPoles}$ be a copy of $\text{ArrStickyPoles}$.
4. **Step 4:** Delete several points in $\text{ArrPrimaryPoles}$ to make it acceptable/feasible.

In Step 2, based on several empirical results, the Norm-2 distance is selected as the sorting metric. That is, the SPs are sorted based on their Norm-2 distance from the origin. The Norm-2 distance is defined as follows.

$$\|X\|_2 = \|(x_1, x_2, \ldots, x_N)\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_N^2}$$
In Step 4, the following procedure is designed to eliminate several PPs to make the final array feasible. Since PPs are sites in the GAS that the RB has to pass through, their locations should not breach any of the two fundamental conditions defined earlier. If PPs are selected so that they follow each other and connect the upper-left corner of the GAS to its lower-right, then the RB that will follows this array of PPs will not violate any of the two fundamental conditions too. The following pseudo-code explains details of this procedure.

```plaintext
CorrFlag=FALSE;
While (not(CorrFlag)) {
    CorrFlag=TRUE;
    For all PP[i]s in ArrPrimaryPoles {
        If (any element of PP[i]>PP[i+1]) {
            Randomly delete PP[i] or PP[i+1];
            CorrFlag=FALSE;
        }
    }
}
```

In the homogenous version, a predefined number of PPs is set diagonally (in multi-dimensional space) from corner \((0,0,\ldots,0)\) to \((|S_1| + 1) \times (|S_2| + 1) \times \cdots \times (|S_N| + 1)\). The following procedure explains more.

- **Step 1:** Locate Sticky Poles in the GAS, namely \(\text{ArrStickyPoles}\).
- **Step 2:** Sort \(\text{ArrStickyPoles}\) according to their location.
- **Step 3:** Diagonally set a predefined number of PP in \(\text{ArrPrimayPoles}\).

**C.Move Blocks**
The aim of this procedure is to move PPs so that they force the final alignment to have a ‘Block’.

Here, a Block is defined as a section of the alignment without any gap \(-\}. Fig. (6) shows four sample Blocks in an alignment. In our approach, a Block can be simply constructed by placing
two PPs with equal distance in all dimensions. Note that, based on the nature of a RB that does not bend between any two Poles, if two PPs are placed cubically in a multi-dimensional space, the RB has no choice but passing straight from one to another; and hence, making a Block. The following procedure explains the details.

**Step 1:** Randomly choose PP[i] from ArrPrimaryPoles

**Step 2:** Randomly choose PP[i-1] or PP[i+1]

**Step 3:** Change the coordinates of PP[i-1]/PP[i+1] with respect to PP[i] to have a cube in the multi-dimensional space without violating PP[i-2]/PP[i+2]

**D. Move Primary Poles**

In this procedure, PPs are slightly moved around their vicinity. Moving PPs enables the final alignment to smoothly move from one answer to another without a drastic change in the final alignment. The following explains this procedure.

**Step 1:** Randomly choose PP[i]

**Step 2:** Add different random numbers between [-3, +3] to different coordinates of PP[i] without violating PP[i-1] and PP[i+1]

**E. Jump to Sticky Poles**

In this procedure, which is one of the crucial operators in this optimization algorithm, a PP is selected and randomly moved to one of its closest SPs, which were formerly stored in ArrStickyPoles. Following this procedure will allow the algorithm to escape from several local minima. This operator usually leads to a drastic change in the final alignment, and hence a sudden change in the final alignment score. The following procedure explains the process.
Step 1: Randomly choose PP[i]

Step 2: Find an array of closest SPs from ArrStickyPoles

Step 3: Randomly jump PP[i] to one of the element in this array

Step 4: If the jumped PP violates other PPs, jam PPs to each other to correct the final RB.

F. Jam Primary Poles
The main object of this procedure is to deactivate the effect of several superfluous PPs in the algorithm. Here, instead of deleting a PP (from the ArrPrimaryPoles) to deactivate it, it is jammed into one of its neighboring PPs. In this case, although, the jammed PP still exists in the system, it cannot bend the RB anymore, and therefore, will not affect the final alignment. However, the jammed PPs can still be useful in exploring the solution space in future iterations. Fig. (7) shows how jamming of PPs may improve/change the final alignment as explained below.

G. Align Primary Poles
This operator is similar to Jam Primary Poles where several PPs are also deactivated, but in a different fashion. Here, a selected PP is aligned with respect to its predecessor and successor by letting its coordinates be the average of its predecessor and successor coordinates. As shown in Fig. (8), the aligned PP is not completely deactivated like Jam Primary Poles, however, deactivated for the current alignment. The aligned PP can still be useful in exploring the space between its predecessor and successor in future iterations. The following procedure explains this
process.

Step 1: Randomly choose $PP[i]$

Step 2: Let coordinates of $PP[i]$ be the average of $PP[i-1]$ and $PP[i+1]$

H. Connect Poles
This procedure is responsible to connect the selected PPs to each other. In this procedure, adequate number of poles are added to the selected PPs that may not be adjacent to make a feasible RB. The following pseudo-code explains details of this procedure.

```plaintext
ArrPoles=ArrPrimaryPoles;
ConnFlag=FALSE;
while (not(ConnFlag)) {
    ConnFlag=TRUE;
    For all PP[i] in ArrPoles {
        If (PP[i] is not adjacent to PP[i+1]) {
            Let P1 be a pole adjacent to PP[i] toward PP[i+1];
            Let P2 be a pole adjacent to PP[i+1] backward PP[i];
            Randomly insert P1 or P2 to ArrPoles between PP[i] and PP[i+1];
            ConnFlag=FALSE;
        }
    }
}
```

I. Fine Tuning
This procedure is launched as the final stage of RBT-L. Here, the restriction about passing through all PPs is removed from the Connect Poles procedure and it is invoked several times to tune the final answer. By using this operator, the Connect Poles procedure will get the chance to violate the PP positions and maybe align few characters differently. This is explained below.

Step 1: Let $ArrPoles$ represents the final RB (alignment answer)

Step 2: Randomly choose $P[i]$ from $ArrPoles$
Step 3: Delete a predefined number of predecessors and successors Poles from $P[i]

Step 4: Reconnect the ArrPoles by launching the Connect Poles procedure

Step 5: Repeat steps 1-4 until the final alignment cannot be improved anymore

J. Optimization Procedure
The following procedure explains the main optimization cycle of the proposed algorithm. Note that, after performing any of the above operations, the last alignment will be replaced by the resulting alignment (RB) if its alignment score is greater (i.e. better), according to the assigned score matrix (e.g. BLOSUM62).

Step 1: Initialize Primary Poles

Step 2: For a predefined number of Tries repeat Steps 3-7

Step 3: Move Blocks

Step 4: Move Primary Poles

Step 5: Jump Primary Poles

Step 6: Jam Primary Poles

Step 7: Align Primary Poles

Step 8: Fine Tuning

The only section of the above procedure that has not been explained yet is Step 2. This step refers to repeating Steps 3-7 for several times before terminating the optimization process. It is because of the nature of our optimization process and its operators that all have stochastic/random behavior. Therefore, it is not impossible to miss better solutions, based on the conditions of the final answer, even after not obtaining any improvements in an optimization
cycle. To prevent this, Steps 3-7 are repeated for several runs before terminating the optimization process.

5. SIMULATION RESULTS

In MSA, the optimal answer is unknown and there is no concrete criterion to evaluate the quality of a given algorithm, unlike the case for Pair-wise alignment where an optimal solution can always be found. Therefore, standard benchmarks, like BALiBASE, are provided to gauge the efficiency of MSA algorithms.

The first version of BAliBASE (Thompson et al. 1999) was dedicated to the evaluation of multiple alignment programs and was divided into five hierarchical reference sets of: (1) equidistant sequences with various levels of conservation, (2) families aligned with a highly divergent ‘orphan’ sequence, (3) subgroups with < 25% residue identity between groups, (4) sequences with N/C-terminal extensions, and (5) internal insertions. For release 2.0 of BAliBASE, these alignments have been manually verified and corrected by superposition of all known three-dimensional structures, using the lsqman program (Kleywegt and Jones 1995). In this benchmark, an open source program is also provided to score the quality of each answer by comparing it with the one biologist found manually. The maximum score is 1.0 and is assigned to the alignments that are identical to the benchmark’s answer; minimum is 0.0 and is assigned to unrelated/unrealistic answers; and, a number between 0.0 and 1.0 for the others. The closer to the manually calculated answer, the higher would be the score.

To demonstrate the performance of the approach proposed in this paper, all benchmarks from Reference #2 and #3 form the BALiBASE 2.0 (Web-Site) have been selected. For all these benchmarks, the BLOSUM62 scoring matrix with the gap penalty of 10 and 1 for the Gap Opening and Gap Extension, respectively, is selected. Random walk and homogenous
initialization mode is selected for short/medium (like 1idy and 1tvxA in Reference #2) and long (like 2myr and 1lvl in Reference #2) sequences, respectively. Fig. (9) shows the bar graph representation of the performance of RBT-L compare with other approaches (including our previous approach, RBT-I (Taheri and Zomaya 2008)) formerly designed to solve the stated benchmarks in Reference #2. Fig. (10) shows the similar results for Reference #3. For each benchmark the RBT-L is executed ten times and its best result is considered as the final answer, and consequently, reflected in these figures. However, the other nine answers were not so apart from the best one, always less than %5 in relation to the final Alignment Score. The reason for obtaining different answers for different runs is significantly related to the nature of this optimization process and its operators, as all of them undertake their optimization steps randomly. Therefore, it is not surprising that they will always fall into different parts of the solution space, yet close enough to each other as a sign of the algorithm’s robustness and repeatability.

6. DISCUSSION AND ANALYSIS

The results obtained by using RBT-L were quite different and interesting, covering a vast variety of situations. In summary, similar to other approaches formerly presented to solve this problem, although RBT-L did not manage to find the identical alignments to benchmark answers, it was generally successful. The following sections explain this in more detail.

A. Alignment Score vs. BALiBASE score

The first observation made after analyzing the solving trajectory of the benchmarks was the weak relationship between Alignment Score, which is purely depends on the Scoring matrix (BLOSUM62 in this case), and the BALiBASE score, which is purely based on biological facts. However, they seemed to be fairly related. In several cases, gaining higher Alignment Scores
yields better BALiBASE scores; although, this cannot be always guaranteed. To investigate this relationship more, we executed the algorithm with different scoring matrices, gap opening and extension values. In almost all cases, the Alignment Score and BALiBASE score showed the same amount of un-correlation. Nevertheless, it seems that in most cases, alignments with higher Alignment Scores have better BALiBASE score as well. Fig. (11) shows a sample of this un-correlation for 1w1t from Reference#3.

B. RBT-L and Reference #2
Reference #2 of the BALiBASE 2.0 is dedicated to ‘orphan’ sequences. These sequences are significantly different in their number of sequences and their sequences’ length. For this class of sequences, RBT-L has shown different performances. As it was shown in Fig. (9), in several cases (like 1idy, 1csy, 1tvxA, 1ubi and 1ajsA), RBT-L managed to significantly outperform the existing pioneer methods. In several other cases, RBT-L’s performance was just fairly comparable to the others, like 1uky, 1tgxA and 2trx. And finally, there were cases that RBT-L did not significantly outperform the existing alignment methods, like 1cpt, 2myr and 1lvl.

C. RBT-L and Reference #3
Reference #3 of the BALiBASE 2.0 is dedicated to subgroups of sequences with < 25% residue identity between groups. Performance of RBT-L in this category was quite different compare to Reference #2. Here, although RBT-L manages to outperform few of the existing methods, it could not significantly outperform any of them. Overall, the performance of RBT-L was fairly better than the others.

D. RBT-L and BALiBASE Score
One of the noticeable facts about RBT-L answers is its non-zero BALiBASE score at all times. Examining Figs. (9) and (10), it can be seen that, several of the existing pioneer alignment
algorithms have significant performance diversity in their results. For example, Fig. (10) shows that PRRP and ML_PIMA received BALiBASE score 0.0 and 0.905 for 1idy and 1r69, respectively. In other words, in these two cases, the final alignment of these two algorithms manifests no biological relationship in one case and almost maximum biological relationship in the other case. In contrast, RBT-L was always able to identify some biological relationship in the sequences to align. In some cases, it almost found the identical answer, like 1idy in Reference #2. In other cases where the biological relationship in the input sequences was subtle and no method could find a reasonable answer, RBT-L was not a total failure too, like lajsA in Reference #3.

E. Overall Performance of RBT-L
It seems that the proposed algorithm in this paper, RBT-L, had a reasonable performance in almost all cases. Although there were cases that some of the existing methods found better solutions, with respect to BALiBASE score, in most cases, the quality of RBT-L’s alignments were as good as or better than the others. In some cases, it could even significantly outperform the existing methods, like 1idy and 1ubi in Reference #2.

Figs. (12) and (13) are provided to show the overall performance of RBT-L compared to other approached. In these figures, the alignment algorithms are sorted according to their average BALiBASE score throughout the whole benchmark. Fig. (12) shows that RBT-L significantly outperformed all other methods in Reference #1, almost %20 better than the second best, SAGA. Fig. (13) shows that RBT-L is ranked 3 based on its average BALiBASE score in Reference #3, although its average BALiBASE score is just %6 below the best approach, PRRP.

F. RBT-L and RBT-I
Comparing RBT-L with RBT-I shows that, although the main optimization procedure of these
two approached are very similar, their performance is quite different. Figs. (12) and (13) shows that, the average BALiBASE score of RBT-L is %26 and %17 greater for Reference #2 and #3, respectively. However, the minimum BALiBASE score RBT-L managed to obtain is significantly greater in both benchmarks. These observations imply that the way SPs are generated at the beginning of the main optimization routine is the key to better alignments. As a conclusion, the location of an AA is more important than its index in a protein.

7. CONCLUSION

In this paper, a novel approach based on the analogy of the behavior of an elastic Rubber Band (RB) on a plate with several poles, is presented to solve the Multiple Sequence Alignment (MSA) problem. The Rubber Band Technique: Index Base (RBT-L) is an iterative optimization algorithm that starts from a relatively reasonable answer, found by one of its operators, and gradually improves it to find the optimal alignment. In this approach, the plate, representing the solution space of a MSA, is equipped with several poles, resembling locations in the input sequences that are likely to be biologically related. Fixing the head and tail of the RB at two corners of the plate, it is free to twist around the poles and find the best configuration, leading to an optimal alignment of the sequences in question. RBT-L is tested by solving several benchmarks from BALiBASE 2.0. Results are promising and articulate the performance of the presented approach.

8. SOFTWARE AVAILABILITY

RBT-L is written in C++ under Microsoft Visual Net 2005 for Windows Operating System. This algorithm is a part of “ProteinAlignment” software that is developed to implement several multiple alignment algorithms. The web-based version of this software have not been
implemented yet, however, researchers can have a free setup package of this software by contacting the authors.

REFERENCES


Input Sequences
TWACTFW
TTWTFW

Aligned Sequences
-TWACTFW
TTW--TFW

Fig. 1: A sample Grid Answer Space with a Rubber Band
Fig. 2: Two Primary Poles with three possible Rubber Bands
Fig. 3: Sticky Poles for the alignment in Figs. 1 and 2
<table>
<thead>
<tr>
<th>Seq</th>
<th>1</th>
<th>2</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4: Amino acids index table (AAIT)
Fig. 5: A scheme for Sticky Poles generation
Fig. 6: Sample blocks in an alignment
Fig. 7: Jam primary poles procedure: (a) before and (b) after
Fig. 8: Align primary poles operator: (a) before and (b) after
Fig. 9: Bar-graph representation for Reference #2 results
Fig. 10: Bar-graph representation for Reference #3 results
Fig. 11: Alignment Score vs. BALiBASE Score for 1wit in Reference #3
Fig. 12: Overall performance of RBT-L in Reference #2
Fig. 13: Overall performance of RBT-L in Reference #3