Search Space Reduction Technique for Distributed Multiple Sequence Alignment

Manal Helal[1, 2], Lenore Mullin[3], John Potter[2], Vitali Sintchenko[1]
[1]Centre for Infectious Diseases and Microbiology, University of Sydney, [2]School of Computer Science and Engineering, University of New South Wales, NSW, Sydney, Australia, [3]National Science Foundation, VA, USA
mhelal@usyd.edu.au, lmullin@nsf.gov, potter@cse.unsw.edu.au, Vitali.Sintchenko@swahs.health.nsw.gov.au

Abstract

Software modeling needs to take advantage of the several High Performance Computer (HPC) architectures through multi-threaded and distributed computing. This work extends the dynamic programming algorithm for Multiple Sequence Alignment (MSA) to be suitable for parallel simultaneous execution, and then reduces the search space using a novel definition of a hyper-diagonal through a tensor space. Experiments demonstrates that scoring less than 1% of the search space produces the same optimal results as scoring the full search space. The alignment scores are better than some of the heuristic methods and capable of aligning more divergent sequences.

1. Introduction

Distributed and parallel application development is more complex than sequential application development. The basic difference is the ability to run on more than one processing element, whether in the same computer for parallel applications, or over a network for distributed applications. The complication stems from design, implementation, debugging, testing, and deployment issues. Basic design complication is based on data partitioning of a sequential solution to run simultaneously, which is not a trivial task. Many problem spaces will require communication between the data partitions, and optimizing the communication overhead versus employing more processors is an open area of research. Using a novel tensor indexing system, a generic partitioning model have been developed in [5] and [6] for the high dimensional Multiple Sequence Alignment (MSA) in computational biology. A smaller partition size produces more independent partitions on a single wave of computation (and hence can employ more processors), but more communication in-between waves of computations. The models presented have been tested on a network of computers of homogenous architecture, and on a single machine with up to 8 cores. The communication is managed by standard Message Passing Interface (MPI) libraries, where the underlying architecture is hidden in the communication APIs. Transforming a sequential application to the tensor indexing scheme to benefit from the partitioning methods used, and analyzing the dependency requirements were presented in [5] and [6].

This paper demonstrates a search space reduction method for the distributed Dynamic Programming (DP)-MSA solutions presented in [5] and [6]. Next section introduces the MSA problem and existing methods, and search space reduction methods in the literature. Section 3 explains the distributed DP-MSA, the novel hyper-diagonal through a tensor space definition, and the various distance measures. Results are presented in Section 4, and conclusions are drawn in Section 5.

2. Dynamic Programming for MSA

MSA is a significant problem in computational biology. The accuracy of MSA is important for various biological modeling methods like phylogenetic trees, profiles, and structure prediction. Simultaneous alignment methods have been proven to be both mathematically optimal and more biologically relevant than the heuristic methods that are commonly used. However, the computational complexity makes the simultaneous methods only feasible for small number of sequences of short lengths.

MSA Existing Methods: Simultaneous alignment methods are based on a DP algorithm [1]. The method is based on stretching each sequence on an axis forming a hyper-lattice, or hyper-cube, or tensor of cells. Each cell index joins the residues at a position corresponding to its element value at the dimension corresponding to the sequence order, forming a set of residues at each cell index. The score given to cells is based on a recursive function that takes into account the gaps inserted from the origin cell down to the cell being scored, and uses a sum-of-pairs score for all cell's set of residues. The alignment generated from tracing the maximum scores in the cells from the highest cell index up to the origin cell index, is optimal because all possible alignments are considered, and the optimal path with highest matching residues and minimum gap insertions, according to the scoring scheme used, is the one with the highest mathematical score. It has been proven to be an NP-hard optimization problem with no polynomial time solution due to the exponential growth in the number of cells to be scored with the number of sequences in the data set and their lengths. Heuristics developed instead. However, progressive and iterative solutions like ClustalW [15], Muscle [16], TCoffee [17] are all based on pair-wise alignments to build a guide tree to assemble the final MSA. These methods work well with sequences of assumed similarity of 90% and are known to suffer from bias [2] in the positioning of gaps and statistical uncertainty [3] in the produced conclusions.

This work is based on the DP simultaneous algorithm for MSA. The 2D-DP recurrence is:

$$\xi_{i,j} = \max\begin{pmatrix} \xi_{i-1,j-1} + sub(a_i, b_j) \\ \xi_{i-1,j} + g \\ \xi_{i,j-1} + g \end{pmatrix}$$

where ξ ij is the cell in the matrix with index representing the ith row (ith residue in the first sequence, denoted by a_i), and jth column (jth residue in the second sequence, denoted by b_j), and g is the gap penalty value, and sub(a_i, b_j) is a function that returns the score of substituting a_i with b_j based on the scoring scheme used.

The algorithm for 3 sequences is described in [1]. The work in [5] introduced an MSA DP scoring recurrence that is invariant of dimension and shape based on the index transformations in the scoring tensor.

Carillo and Lipman Projections: Carillo and Lipman [4] noticed that scoring all the exponentially growing number of cells in the DP scoring hyper-cube is not needed to reach an optimal solution. The Carillo-Lipman DP optimal MSA search space reduction is based on the scores of its pair-wise projections. This is because of the observation that the score of the projection of an optimal MSA into any of its sequence pairs must be no greater than the score of their pair-wise alignment. The bounds in the region around the hyper-diagonal are defined by the projections of each pair-wise alignment on the diagonal. The Carrillo and Lipman bounds are extended to MSA using bounds on the scores of its projections into any lower dimensional space with two or more dimensions and less the original dimension. Carrillo and Lipman bounds have been observed to be over-estimated. Different methods were proposed in the literature to optimize Carrillo and Lipman bounds.

The MSA tool in [7] presented a simultaneous method that is a heuristic variant of the Carrillo and Lipman method where alignments are scored as the cost of an evolutionary tree instead of the standard sum-of-pairs scoring scheme. The work in [8] proposing a branch and bound algorithm for maximum weight trace that merges weighted pair-wise alignments to form an MSA, using minimum sum of pairs alignment problem as a special case. The study in [9] explored the lattice efficiently by means of Dijkstra single-source shortest path algorithm. Using the work in [8], [10] formulated an Integer Linear Programming (ILP) branch and cut algorithm. The methods in [11] applied divide-and-conquer techniques by slicing the input sequences into segments that are later aligned using smaller lattice spaces. Using a goal-directed unidirectional search (the A* algorithm), the methods in [12] transformed the edge weights without losing the optimality of the shortest path and was able to exclude more nodes from computation than the Carrillo and Lipman bounds. Then [13] combined the divide-and-conquer approach of [11] with the bounding strategies of [10]. The work presented in [14] generalized the ILP formulation of [10] using arbitrary gap cost in a branch-and-cut algorithm for MSA. The Carrillo and Lipman approach uses the known topologies for a high-dimensional space which all describe the external points of the space, and only projection on these surface points can be measured.

3. Distributed Optimal MSA

The work in [5] and [6] have employed parallel processing to enable optimal simultaneous methods to be applied to larger data sizes by using novel tensor indexing methods and partitioning scheme. The master/slave cubical partitioning method provided in [5] divides the scoring tensor space into waves of computations that contain partitions of a specified generic size on a perimeter of equal distance from the origin, where the number of partitions on each subsequent perimeter is more than the previous one. The master process were consuming resources in the partitioning and scheduling of partitions over processors. Each perimeter is considered a wave of computation that can employ as many processors as there are partitions. There was communication overhead within and across the waves of computation. The Peer-to-Peer (P2P) diagonal partitioning method in [6] aligned the partitions on equal distances from the origin for each partition, forming true diagonals and less partitions on equal distances (waves of computation), and eliminating the communication between the partitions on the same wave. The P2P partitioning used a hash function that maps the processor index to a partition index. The later property eliminated the dedicated scheduling process and reduced the scheduling communication significantly.

The search space reduction approach described in the following subsections utilizes the new indexing scheme of the tensor internal points presented in [5] and [6], and attempt to provide measures of the distance of each internal partition and the middle partition of the encapsulating wave (distance from origin). This measure is used to define the bounds (absolute distance from the middle) on the band that can be scored to reduce the search space.

Hyper-Diagonal Definition: Hyper-diagonal is defined as a line through the middle partitions in each wave. Then a measure between the partitions on the same wave is developed in next subsection. Opposite to Carrillo and Lipman approaches, which is based on pair-wise alignments that is less sensitive than the simultaneous alignments, this work is based on simultaneous alignment. Also this work reduces the search space by deciding the distance around the middle diagonal to score, as opposed to deciding how much to trim from the edges in the Carrillo and Lipman approaches. The presented approach is based on the P2P partitioning scheme described in [6].

The basic idea is to score the partitions around the hyper-diagonal that connects between two points: the origin of the hyper lattice and the last point in the lattice that is indexed as the lengths of the corresponding sequences lengths on each axe. Waves are defined in [6] as groups of partitions for which the sum of its partition index elements are equal, i.e. on the same distance from the origin. A new wave is created for each subsequent increment. Therefore, a partition on one wave has exactly one index element incremented once to get to the higher neighboring partition on the next wave. Waves are calculated by all the permutations of the indices for which the elements sum is equal to the wave number. This refers to all permutations of all integer partitions of the specified wave number, over the dimensionality k.

The hyper-diagonal is defined in the same way as a 2-D diagonal; namely, (0, 0), (1, 1), (2, 2), ... (l_1, l_2), where l_i is the length of sequence i. In higher dimensions, the diagonal connects the origin to all cells of equal index elements, i.e. (0, 0, 0, ... 0), ..., (1, 1, 1, ..., 1), ... (2, 2, 2, ... 2), ... (l_1, l_2, l_3, ... l_k). There are variable paths that can be taken to connect these points across the middle waves in between each two consecutive diagonal points. The intermediate points that define the middle partitions in the waves in between (0, 0, 0, ... 0) and (1, 1, 1, ..., 1), up to (l_1, l_2, l_3, ... l_k), can be defined in different ways. However, a simple way is selected and described below.

The points connecting the hyper-diagonal line through the tensor space constitute the middle partition on each wave. These points are calculated as the fair division of the wave number over the indices; with the highest remainder distributed over the middle indices. The middle partition index is calculated in the following procedure:

```
q = floor(w/k)
r = w mod k
for i = 0 to k-1:
if  floor((k-r)/2) <= i < floor((k+r)/2)
                mp(i) = q + 1,
else
        mp(i)  = q,
```

Where mp(i) is the middle partition index at i^th^ dimension, k is the dimensionality, and w is the wave number. Examples are shown in Table 1.

Table 1: Middle partition index examples

w	k	mp
5	6	{1, 1, 1, 1, 1, 0}
9	6	{1, 2, 2, 2, 1, 1}
12	6	{2, 2, 2, 2, 2, 2}
35	8	{4, 4, 5, 5, 5, 4, 4, 4}
40	12	{3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3}

Then, the values in mp are multiplied by the partition size S to get the actual partition index. The latter falls in the middle line that connects the origin cell to the last cell in the tensor; thus, passing through all computation waves.

Distance Measures: Now that the diagonal line points are well defined, the distance between the middle partition index to each partition in the same wave is to be specified. This is necessary in order to decide whether it is within the specified Epsilon value, and therefore, can be included in the scoring, and its higher neighbor in subsequent waves can be explored. Several geometrical methods are discussed, and several problems are addressed. However, it is still an open area of research to explore.

The main objective is to uniquely discriminate between partitions in the same wave. This is achieved by dividing the partitions into 2 categories. The first category contains those on the left hand side of the middle partition in the current computation wave. The second category contains those partitions on the right hand side of the middle partition. The partitions are expected to be evenly distributed on a line connecting all partitions in the same wave. Therefore, distance between each partition and the middle partition index is measured.

The traditional geometry does not scale with dimension; moreover, it does not handle symmetry in the partition indices in an efficient manner. The first test is to calculate the Pythagoras distance between the middle partition index vector and the other partition indices in the same computation wave. To measure the Pythagoras distance between two points x and y in the tensor of dimensionality k, the following equation is applied:

$$\sqrt{\sum_{i=0}^{k-1}(y_i - x_i)^2}$$

Table 2 lists some examples. The first 2 rows show an example of symmetry problems. They measure the distance between the middle partition at wave 6 and two edge vector indices in the opposite directions on the tensor graph. The distance measure is the same. Then the next 2 row shows the distance between the same middle partition and other non-symmetric vector indices on the same wave, but the sum of their elements are the same. Both still produced the same distance measure because the elements differences compensated each other at the final summation. This is because the distance calculation is neither sensitive to the directions in each dimension, nor to elements positions..

Another method attempts to measure the angle between the index vectors of the partitions as calculated in the following equation:

$$\theta = \cos^{-1}\left(\frac{x.y}{\|x\| \quad \|y\|}\right)$$

Where $x.y = \sum_{i=0}^{k-1}(x_i y_i)$ and $\|z\| = \sqrt{\sum_{i=0}^{k-1} z_i^{\,2}}$ for any vector z.

However, this method results in non-uniform angle distances (no unit distance between each 2 consecutive partitions, on both sides of the diagonal) as shown in Table 2, and produces same problems.

A third approach is the Manhattan distance, or city block distance. This method measures a route along non-hypotenuse sides of a triangle, i.e. grid-like layout of points. It is calculated as follows:

$$\sum_{i=0}^{k-1}|y_i - x_i|$$

Table 2: Distances measures between partition indices on the same wave examples.

First Partition Index	Second Partition Index	Pythagoras distance	Angle distance	Manhattan distance
(1,1,2,1,1)	(6,0,0,0,0)	5.66	0.99	10
(1,1,2,1,1)	(0,0,0,0,6)	5.66	0.99	10
(1,1,2,1,1)	(0,0,2,3,0)	2.65	-0.88	5
(1,1,2,1,1)	(0,1,1,3,0)	2.65	0.73	5

The solution used attempts to handle all the problems mentioned above by ranking the dimensions of the measured indices. The idea is based on the Gray Codes, in which each index change counts. For example, the distance between (0, 0) and (1, 0) is equal to 1, while the distance between (0, 0) and (1, 1) is equal to 2. Furthermore, the sign problem is solved by taking the first different index sign. If the first index element is different in a dimension that is lower than the middle dimension, therefore the sign is positive. If the first different index is at a dimension higher than the middle dimension, then the sign is negative. The problems of permutations and symmetry are solved by assigning different weights to each dimension, using the following equation:

$$\sum_{i=0}^{k-1}(y_i - x_i) \quad \times \quad i^2$$

Table 3 shows the distances based on Equation 5 for the partition indices examples used before in Table 2.

Table 3: Index ranking distances measures examples.

First Partition Index	Second Partition Index	Index ranking distance
(0, 0)	(1, 0)	-4
(0, 0)	(1, 1)	-1
(1, 1, 2, 1, 1)	(6, 0, 0, 0, 0)	-682
(1, 1, 2, 1, 1)	(0, 0, 0, 0, 6)	106
(1, 1, 2, 1, 1)	(0, 0, 2, 3, 0)	58
(1, 1, 2, 1, 1)	(0, 1, 1, 3, 0)	51

The Epsilon value in index ranking method is a percentage of the distances to include in the search space. However, the unit partition and the full wave length should be calculated. The unit partition is defined as distance between the middle partition and its next permutation. The full wave length is the distance between the middle partition and the first partition in the wave. The first edge partition is the vector index where the wave number is the value of the first dimension index and the rest of the dimension indices are all zeros. Using this distance measure, the reduced search space area takes the shape of a cone whose slope increases at a fast pace close to the origin of the tensor graph space, while decreases at a low pace across the waves which are close to the bottom of the tensor graph. However, the distance measure between the partitions is still not uniformly distributed over the wave length. Moreover, no fixed percentage of all partitions in each wave is included in the search space. This is because more edge partitions are scored on the top of the tensor graph. As the shape of the tensor increases, and the number of dimensions increases, the reduced searched space area becomes too much larger than what is actually required.

A more intuitive approach is followed to decide a specific constant band around the middle partition in each wave. This is done by having a fixed distance for the allowed variation in each dimension index from the middle partition in each wave. This requires the Epsilon value to be a fixed number: it designates the maximum absolute distance between each index element value in the partition index to the corresponding index element in the middle partition. This method produces a fixed band with symmetrical increasing and decreasing curves at both the top and the bottom of the tensor graph (tensor lattice points as vertices of the graph, connected by neighbouhood on single index element change) as shown in Figure 1. The number of partitions per wave grows till it reached a fixed number that is a function of the dimensionality and the Epsilon value, and then it remains constant for all middle waves, till it starts decaying again in the same pace as the initial growth. The maximum number of partitions for the dimension and the Epsilon values are shown in Table 4 and their exponential growth is very fast. This fast growth makes this distance measure not as sensitive as would have been desirable, but it scores very well due to its ability to define the partitions around the middle band through the tensor space efficiently. The function that describes the growth in Table 4 is the function for all multidimensional neighbours of a cell which are (3^k – 1) neighbours for one stride around the cell (E = 1) and (5^k-1) for E = 2, and generally ($((E*2)+1)^k$-1) for any arbitrary E value, retrieving only the neighbours that fall on the same wave number, i.e. the sum of the neighbour's index elements are equal the wave number of the given middle partition cell index. The neighbours of a cell spans +k and –k waves around the cell's wave number, and the number of the neighbours falling on each of these waves, vary with the shape of the dataset, and the position of the cell itself. So, it is recursively computed without a generic closed formula on the bounds.

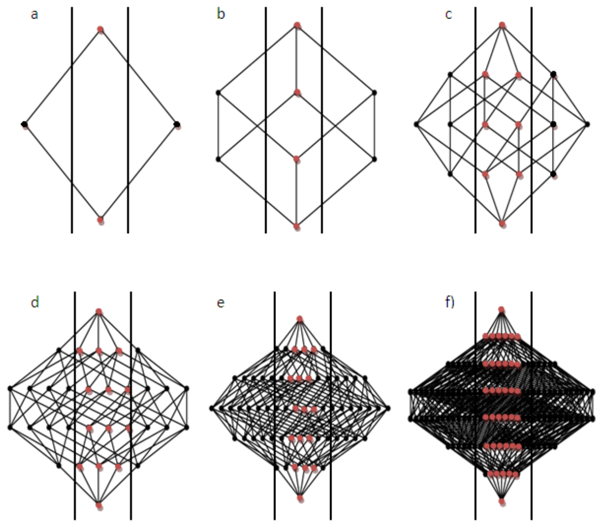


Figure 1: Fixed Epsilon search space reduction on the tensor graph Representation

Table 4: Maximum partitions in all waves per E value and dimensionality k.

k / E	3	4	5
1	7	19	51
2	19	85	381
3	37	231	1451
4	61	489	3951

5	91	891	8801
6	127	1469	13651
7	169	2255	20851
8	217	3281	31321
9	271	4579	46211
10	331	6181	66901

4. Experimental Results

The machine used is a SunFire X2200 with 2xAMD Opteron quad processors of 2.3 GHz, 512 Kb L2 cache and 2 MB L3 cache on each processor, and 8GB RAM. Randomly created sequences have been selected to test the performance of varying the E value. Table 5 describes the data selected, where K is the dimensionality (number of sequences), L is the sequence lengths, C is the computation cost represented in the number of cells of the scoring tensor, which is the product of the sequence lengths each incremented by 1 for the initial gap, T is the number of waves, and P is total number of partitions all over the waves.

Table 5: Data sets used in the experiments.

Set #	K	L	C	T	P
1	2	20,20	441	12	100
2	2	30,31	961	29	225
3	3	21,21,21	9261	28	1000
4	3	31,31,31	29791	43	3375
5	4	31,31,31,31	923521	57	50625
6	4	41,41,41,41	2825761	77	160000
7	5	41,41,41,41,41	115856201	96	3200000

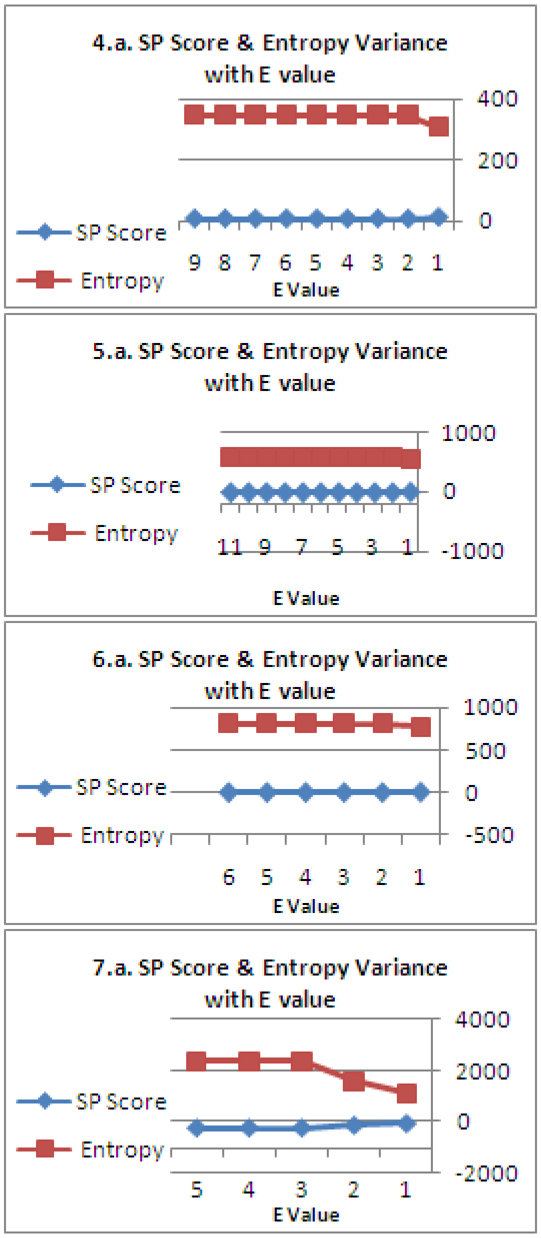
Table 6 shows the performance measure produced for varying the E value for the chosen data sets, where E is the epsilon value, % is the percentage of partitions computed from the total partition shown in Table 5 for this data set, ST is the system time (cpu time) in seconds, UT is the user time (wall time) in seconds, M is the process memory size, SP and Ent. are the Sum-of-Pairs score and Entropy respectively of the produced alignment, and AL is the alignment length.

Table 6: Experiments performance results.

Set #	E	%	ST	UT	M	SP	Ent.	AL
1	1	51.00	0.04	0.03	146432	5	93.59	21
1	2	75.00	0.06	0.04	146432	5	93.59	21
1	3	91.00	0.07	0.03	146432	5	93.59	21
1	4	99.00	0.06	0.07	146432	5	93.59	21
1	5	100.00	0.07	0.04	146432	5	93.59	21
2	1	36.00	0.07	0.06	146432	5	164.35	31
2	2	55.56	0.08	0.04	146432	5	164.35	31
2	3	71.56	0.09	0.04	146432	5	164.35	31
2	4	84.00	0.10	0.41	146432	5	164.35	31
2	5	92.89	0.11	0.05	146432	5	164.35	31
2	6	98.22	0.11	0.07	146432	5	164.35	31
2	7	100.00	0.11	0.06	146432	5	164.35	31
3	1	17.20	0.11	0.09	146432	16	165.97	21
3	2	40.20	0.15	0.12	146432	16	165.97	21
3	3	65.40	0.21	0.84	146432	16	165.97	21
3	4	86.60	0.26	1.22	146432	16	165.97	21
3	5	97.80	0.32	1.04	146432	16	165.97	21
3	6	100.00	0.32	2.94	146432	16	165.97	21
4	1	8.21	0.06	0.18	244736	10	308.72	34
4	2	20.36	0.10	0.95	244736	5	345.36	37
4	3	35.82	0.23	0.36	244736	5	345.36	37
4	4	52.77	0.28	7.21	277504	5	345.36	37
4	5	69.30	1.04	3.73	277504	5	345.36	37
4	6	83.59	0.35	1.56	277504	5	345.36	37
4	7	93.72	0.39	2.80	277504	5	345.36	37
4	8	98.64	0.39	6.43	277504	5	345.36	37
4	9	99.94	0.40	4.21	277504	5	345.36	37
5	1	1.94	0.17	1.14	174	-4	538.90	39
5	2	7.81	0.44	15.34	447	-11	575.49	41
5	3	18.81	0.97	66.36	995	-11	575.49	41
5	4	34.68	1.90	616.28	1951	-11	575.49	41
5	5	53.84	2.96	206.47	3034	-11	575.49	41
5	6	73.33	4.11	322.89	4213	-11	575.49	41
5	7	88.84	5.28	473.04	5408	-11	575.49	41
5	8	96.76	6.43	495.34	6587	-11	575.49	41
5	9	99.46	8.54	748.25	8750	-11	575.49	41
5	10	99.98	7.47	579.96	7644	-11	575.49	41
5	11	100.00	7.46	503.38	7643	-11	575.49	41
6	1	0.85	0.30	5.71	277504	3	779.27	52
6	2	3.53	0.92	36.01	278528	-2	816.79	54
6	3	8.84	1.90	95.96	280576	-2	816.79	54
6	4	17.09	1.39	69.15	281600	-2	816.79	54
6	5	28.17	6.31	439.75	285696	-2	816.79	54
6	6	41.56	10.13	756.10	287744	-2	816.79	54
7	1	0.14	1.25	114.12	283648	-41	1143.83	55
7	2	0.97	9.66	907.05	327680	-116	1611.69	72
7	3	3.39	36.27	8344.97	327680	-219	2361.11	96
7	4	8.35	112.41	50208.94	454656	-219	2361.11	96
7	5	16.62	320.66	214003.73	608256	-219	2361.11	96

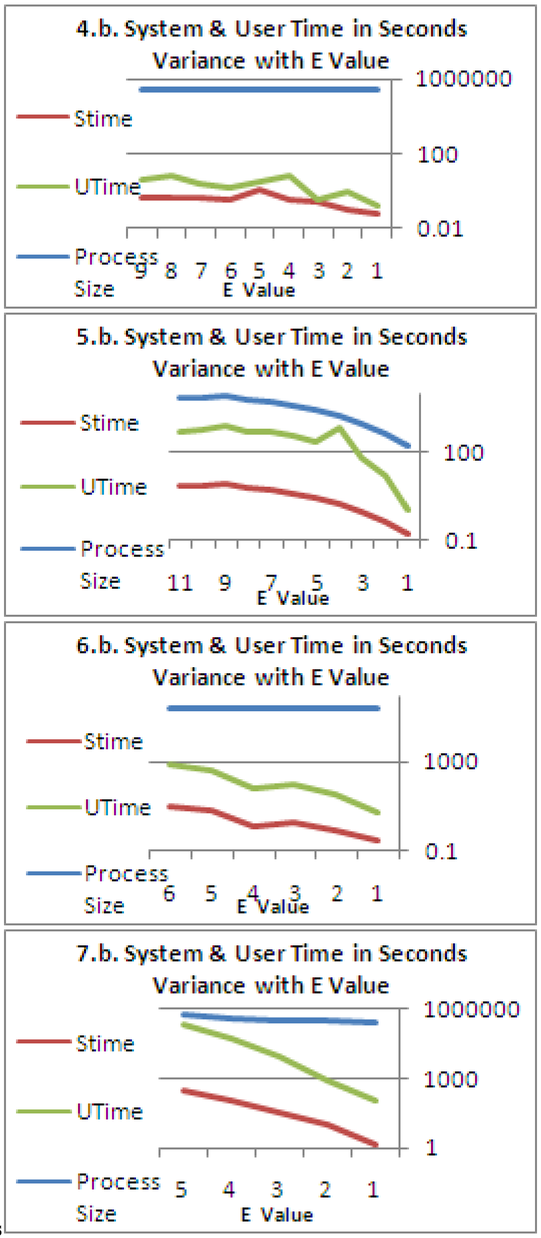
The results show that in the first 3 datasets, which are fairly small, the smallest search space produced exactly the same alignment scores (SP and Ent.) as the Full Search Space (FSS). As the data sets grows, in the fourth data set, the smallest search space value E = 1, (which is 0.06% of the FSS), produced 10 SP with 34 AL, which is better than the steady SP of 5 with 37 AL that is produced using E = 2 (which is 0.1% of the FSS) and up to the FSS. Same observation is shown in dataset 5, where E = 1 (0.17% of the FSS) produced -4 SP with 39 AL, which is better than the -11 SP with 41 AL from E = 2 (which is 0.44% of the FSS) and up to the FSS. In the largest dataset used, the E = 1 (0.14% of the FSS) produced -41 SP with 55 AL; E = 2 produced -116 SP with 72 AL, which is 0.97% of the FSS, and E=3 (which is 3.39% of the FSS) up to the FSS produced -219 SP score with 96 AL. The smallest band around the hyper-diagonal forces the alignment to use fewer gaps and more substitutions, which in turn causes shorter alignment length and higher SP score because of the less gap penalty incurred. As the band around the hyper diagonal increases, there is more room for gap insertions, and hence, longer alignment lengths, and lower SP score. Based on the size of the dataset, the increase in the search space, doesn't affect the produced alignment. In the first few data sets, there where no change at all, then steady results where reached at E = 2 for data set 4, 5 and 6, and E = 3 for dataset 7. These results are plotted in Figure 2 and Figure 3, and graphs' titles are labeled after the dataset they represent.

Figure 2. Performance results for the conducted experiments illustrating the alignment scoring variations over the change of the E value on the x-axis.



The experiment conducted in [18] show that the proposed method score came third after TCoffee and Muscle, where similar sequences were aligned, while it scored the highest over all other methods tested for the alignment of the most divergent sequences. These results show that the presented method scores better when aligning sequences of large dissimilarity, and can be used to identify regions of high dissimilarity along the full length of the input sequences.

Figure 3: The resource consumption of the experiments illustrated on a logarithmic scale over the change of the E value on the x-axis



5. Conclusion

The novel tensor indexing scheme used in [5] and [6] for partitioning the scoring tensor for the parallel processing of the DP MSA problem, have enabled another view for the internal points of the tensor space. New distance measures have been implemented to logically diagonalize the tensor and measure the distance between the middle partition in each wave of computation and the near-by ones. The bounds created around the middle hyper-diagonal are the tightest presented in literature, and is not based on pair-wise projections that are less sensitive and trimming on outer tensor borders only and only useful for sequences of higher similarity. The presented method starts from the hyper-diagonal and increase the band around it till steady results are produced.

The initial results show that scoring less than 1% of the full search space can produce the exact optimal score as scoring the full search space. This is justified by the scoring function that calculates all possible paths from the origin up to the current scoring cell that fall within the search space considered, and average the missing neighbours. This DP scoring method is more sensitive than the sum-of-pairs method, and this is why a very small band around the middle of the tensor was sufficient for global alignments.

References

1. Gusfield, D. *Algorithms on Strings, Trees, and Sequences*. New York, NY, USA: Cambridge University Press, 1997.
2. Golubchik T, et al. *Mind the gaps: evidence of bias in estimates of multiple sequence alignments.* Mol Biol Evol 2007;24(11): 2433-42.
3. Wong KM, et al. *Alignment uncertainty and genomic analysis*. Science 2008;319(5862): 473-6.
4. Lipman, H. C. The *multiple sequence alignment in biology*. SIAM J. of App. Math., 1988:48, 1073-1082.
5. Helal M, et al. *Multiple sequence alignment using massively parallel mathematics of arrays*. In: Proceedings of the International Conference on High Performance Computing, Networking and Communication Systems (HPCNCS- 07). Orlando, FL. USA, 2007:120-7.
6. Helal M, et al. *Parallelizing Optimal Multiple Sequence Alignment by Dynamic Programming.* In Proceedings of the International Symp. on Adv. in Parallel & Dist. Computing Tech. (APDCT-08) in conjunction with 2008 (ISPA-08), Sydney, Australia, December 10-12, 2008:120-7.
7. Lipman, D. J. *A Tool for Multiple Sequence Alignment*, Proc. Nat. Acad. Sci. USA, 1989:86, 4412-4415.
8. Kececioglu, J. *The maximum weight trace problem in multiple sequence alignment*. Proceedings of the 4th Symposium on Comb. Patt. Matching. 1993:684, 106-119.
9. Gupta, S. K. *Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignments*. J. Comput. Biol. 1995:2 (3), 459–472.
10. Reinert, K. L. *A branch-and-cut algorithm for multiple sequence alignment*. In Proceedings of the First Annual International Conf. on Comput. Mol. Biol. (RECOMB-97). 1997:241–249.
11. Stoye, J. E. DCA: *an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment.* Comput. Appl. Biosci , 1997:13, 625–626.
12. Lermen, M. A. *The practical use of the A* algorithm for exact multiple sequence Alignment*. J. Comput. Biol. 2000:7, 655–671.
13. Reinert, K. S. *An iterative method for faster sum-of-pairs multiple sequence alignment*. Bioinf., 2000:16, 808–814.
14. Althaus, E. C. *Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics*. Bioinf., 2002:18 (Suppl. S.), S4–S16.
15. Thompson, J. D. *CLUSTAL W, improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice*. Retrieved from http://wwwbimas.cit.nih.gov/clustalw/clustalw.html.
16. Edgar, R. C. *MUSCLE: multiple sequence alignment with high accuracy and high throughput.* Nucl Acids Res 2004;32(5):1792-97.
17. O'Sullivan O, et al. *3DCoffee: Combining Protein Sequences and Structures within Multiple Sequence Alignments.* J Mol Biology 2004;340:385-95.
18. Helal, M., Sintchenko, V., *Dynamic programming algorithms for discovery of antibiotic resistance in microbial genomes*, in proceedings of Health Informatics Conference (HIC-09), to be held in 19-21 August, 2009, Canberra, Australia.