Network Modeling Analysis in Health Informatics and Bioinformatics, December 2015, Volume 4, Issue 25.
The final publication is available at Springer via http://dx.doi.org/10.1007/s13721-015-0096-z.

SEAL: A Divide-and-Conquer Approach For Sequence Alignment

Harini Kandadi, and Ramazan Savas Aygün[1]

[1]Computer Science Department, University of Alabama in Huntsville.

{harinikandadi@gmail.com, aygunr@uah.edu }

ABSTRACT

Sequence similarity search and sequence alignment methods are fundamental steps in comparative genomics and have a wide spectrum of application in the field of medicine, agriculture and environment. The dynamic programming sequence alignment methods produce optimal alignments but are impractical for a similarity search due to their large running time. Heuristic methods like BLAST run much faster but may not provide optimal alignments. In this paper, we introduce a novel *se*quence *al*ignment algorithm, SEAL. SEAL is a parallelizable algorithm that does not require gap penalty parameter as in heuristic methods. SEAL uses a combination of divide-and-conquer paradigm and the maximum contiguous sub-array solution. SEAL is also improved by the use of borders in every contiguous segment. The alignment scores obtained by SEAL are consistently higher than those obtained by heuristic methods. Since the dependencies are minimized among intermediate steps, the complexity of SEAL can be reduced to $\Theta(\log^2 n)$ in the presence of satisfactory number of parallel processors.

1 INTRODUCTION

Comparative genomics enables functional annotation of genes by comparing genes of different species. A sequence similarity search is an integral step in comparative genomics and proteomics. Sequence similarity search helps us identify similar DNA or protein sequences from the same or different species.

A sequence similarity search lines up sequences using sequence alignment methods to compare them and identify regions of similarity between a given query sequence and chosen database. Dynamic programming is a well-known method for sequence alignment and gives the highest scoring alignment between two sequences. Heuristic methods are less time-consuming and give good alignments. The most popular heuristic tool is BLAST (Basic Local Alignment Search Tool). BLAST gives good alignments in a reasonable amount of time but misses some sequences in the search process due to its strict parameters.

Dynamic programming algorithms provide the highest scoring alignments (Shpaer; 1996) and the number of false positives and false negatives is proven to be significantly lower than heuristic methods (Pearson; 1995). Therefore, there is a high risk that many sequences that are readily detected by Dynamic Programming algorithms may be missed by heuristic approaches like BLAST. Dynamic programming algorithms give optimal alignments between two sequences whereas BLAST search results may not necessarily be optimal and heuristic approaches give more than one alignment for a single database sequence compared. In cases where the search is focused on remote homology, heuristic methods may miss certain sequences. But time complexity for dynamic programming algorithms is high since an optimal alignment is obtained only after the whole matrix is filled (the number of cells in the matrix is the product of the lengths of the two sequences).

As sequence databases are increasing rapidly on a daily basis, parallelizing alignment methods to increase the speed and performance of search is gaining importance. There are various approaches used for parallel sequence alignment and search. Parallel approaches can be used at various stages: processing of input query sequences (i.e., each processor works on a subset of the query set), alignment algorithm (i.e., parallel version of the alignment algorithm is developed), and/or searching database sequences (Mathog; 2003) (i.e., each processor searches in a specific portion of the database). Most methods are feasible by the use of unique hardware like shared memory multiprocessors, systolic arrays (White; 1991), Blue Gene/P architecture (Lin; 2008), grid computing (Krishnan; 2005) and more. BLAST++ (Wang; 2003)[6], SOAP-HT Blast (Wang, Mu; 2003) and similar methods employ parallel processing of input query sequences. pioBLAST uses dynamic partitioning of databases (Lin; 2005). HBLAST (O'Driscoll et al., 2015) utilizes Hadoop architecture based on MapReduce framework using virtual partitioning concept for parallel sequence alignment. In (Dai; 2012), a cloud based service is provided using the Map-Reduce Framework for the short read mapping and storing reference genomes in Hbase. Bwasw-Cloud (Sun; 2014) extends BWA-SW algorithm using open source implementation of Map-Reduce framework. The Map phase performs alignment with respect to each reference chunk by extending around the reference; the Shuffle phase clusters alignment locations for each read output; and the Reduce phase combines the alignment locations with the same reads. BWA-ST (Li; 2010) follows seed-and-extend paradigm but finds the seeds using dynamic programming. Parallel methods based on BLAST also have the low sensitivity problem of BLAST.

There are a number of pairwise alignment algorithms. AlignMe (Stamm et al., 2014) provides pair-wise sequence-to-sequence alignment using the standard Needleman-Wunsch algorithm. It has four optimized parameter set: AlignMe PST, AlignMe,PS, Align Me P, and Fast. The letters (P, S, T) along with the algorithm correspond to the required inputs for the alignment where P, S, and T indicate position-specific substitution matrix, a secondary structure prediction, and a transmembrane prediction, respectively. AlignMe PST, PS an7d P work with distantly related proteins (with sequence identity <15%), low-homology proteins sequence similarity (~15%-

45%), and closely related proteins (>45%), respectively. According to their experiments, AlignMe PST provided 1.8-7.5% more correctly aligned positions than HMAP (Soding, 2005) or HHalign (Huang and Miller, 1991). AlignMe produced 6.5% more correct alignment than HMAP, and AlignMe P provides 4.1% more correct aligned positions than HMAP. AlignMe P, PS, and PST use PSI-BLAST to generate a position-specific substitution matrix. AlignMe Fast avoids the PSI-BLAST search in the other versions. WHAM (Li et al., 2012) is designed for short-read alignment problem. The sequence is represented in binary format as bits by mapping each nucleic acid to a binary number. The pairwise alignment is applied on binary representation. Since scoring is not part of this algorithm, it is assumed that WHAM targets exact match with errors of substitutions, insertions, and deletions. Choi et al. propose PROVEAN (Choi, 2012), a fast computation of pairwise sequence alignment based protein sequence variations. Their proposed algorithm takes *O((n+l).m)* time where *n* and *m* correspond to the length of sequences and *l* is the number of variations if the length of variations is constant. It is assumed that two sequences differ in a small contiguous region. MC64-NW/SW (Díaz et al., 2011) method redesigns Needleman-Wunsch/Still-Waterman (NW/SW) algorithm that obtains optimal sequence alignment in quadratic time and space cost for parallelization to yield in *O(m+n)* complexity.

European Bioinformatics Institute (EMBL) (Li et al., 2015; McWilliam et al., 2013) provides a number of database, tools and services. Their sequence alignment services (https://www.ebi.ac.uk/Tools/psa/) include Lalign, EMBOSS tools (matcher (based on Lalign), needle (Needleman-Wunsch), stretcher (modified Needleman-Wunsch), water (modified Smith-Waterman)), and the Wise2 tools (GeneWise, PromoterWise and Wise2DBA). Their sequence similarity search services (https://www.ebi.ac.uk/Tools/sss/) include FASTA, SSEARCH (based on Smith-Waterman) PSISearch (using PSI-BLAST and Smith-Waterman), GGSEARCH (global-global alignment), GLSEARCH (global query -local database alignment), FASTM/S/F, NCBI BLAST, PSI-BLAST, WU-BLAST, and ENA Sequence Search (faster than BLAST for large datasets but with marginal loss of sensitivity).

Divide-and-Conquer methods have been applied to sequence alignment in the past. The recurrence relation of a divide-and-conquer algorithm can generally be represented as *T(n)=aT(n/b)+f(n)*. While *(n/b)* is related to the size of the sub-problem to be solved, the coefficient *a* indicates the number of sub-problems to be solved. The function $f(n)$usually indicates the time to combine the solutions of sub-problems. The function $f(n)$ plays critical role in the complexity of a divide-and-conquer method. If $f(n)$ has a polynomial complexity of $\Theta(n^c)$, the best complexity (using Master Theorem) is obtained when $\log_b a < c$ and the complexity becomes $O(n^c)$. Preferably, fast parallel algorithms should have a low value for the coefficient *a* (e.g., 1), and a high value for the coefficient *b* to reduce the size of the sub-problems. However, even for the best combination of coefficients *a* and *b*, the complexity of the method is dictated by $f(n) \in O(n^c)$. Ultimately, to make divide-and-conquer efficient, the complexity of $f(n)$ should be reduced.

In the literature, there are a good number of applications of divide-and-conquer for sequence alignment. One application area is multiple sequence alignment. One of the key ideas in algorithms proposed in (Stoye, 1998), (Stoye et al., 1997), (Tönges et al., 1996), (Stoye, 1997) is that alignment problem can be divided or sliced based on the center or a reference index position of one of the strings. Then the question becomes finding out the position where other strings meet with the center of the first string. After those positions are found, the alignment can be divided into two sub-problems: alignment up to the mid-point of the first string and alignment after the mid-point of the string. These algorithms are based on dynamic programming. If no optimization has been done, the time complexity of these algorithms is $O(n^2)$ related to the cost of finding corresponding positions in the other strings. A speed-up (Jones and Pevzner, 2004) is provided by block alignment at the coarser level and using a lookup table. The matrix is divided into *(n/t)* by *(n/t)* blocks. Mini-alignment is applied to each block at the cost of $O(t^2)$. However, this still leads to complexity of $O(n^2)$. The complexity of mini-alignment is reduced to $log\ n$ using a lookup table and setting *t* to $log\ n$. The complexity is then reduced to $O(n^2/logn)$. At its core, it is still a dynamic programming approach but starts at the coarser block level. Ideally, to improve divide-and-conquer methods, $f(n)$ should be minimized without affecting coefficients *a* and *b* in the recurrence relation.

Although a number of algorithms are developed to parallelize the alignment algorithm, they still face some limitations. For example, in dynamic programming methods, the computation of a value in the matrix depends on the computation of values in the neighboring cells. Due to this, parallelization is significantly limited. Heuristic approaches have also limitations. For example, parameters such as gap penalty should be defined by the user. Developing faster approaches based on these methods will again face similar problems. Heuristic approaches find hits and explore these hits to expand the matching sequence heuristically based on some parameters provided by the user. For divide-and-conquer problems, there is a limit on how further the problem can be divided into sub-problems. Even with the availability of all the processing power, there is a theoretical limit because of the number of levels a problem can be divided into sub-problems due to the dependency between sub-problems. Therefore, an alternate reasoning about alignment is required.

This work introduces a novel parallelizable and sensitive method for sequence alignment called SEAL (SEquence ALignment). SEAL integrates the advantages of divide-and-conquer paradigm and the maximum contiguous sub-array solution. Divide-and-conquer divides the alignment search space between two sequences into smaller parts and the maximum contiguous sub-array solution finds locally optimal sequences. Given an array of integers, the maximum contiguous sub-array solution attempts to find the sub-sequence of consecutive integers which have the highest sum. The method finds the maximum contiguous segment in the whole search space initially and then uses divide-and-conquer to recursively find such segments in the prefix and suffix spaces. It provides a complete alignment of two sequences where locally optimum sub-sequences are joined with gaps. In an improved version of SEAL called iSEAL, each contiguous segment is further divided using borders and the segments outside the borders are re-explored. This work focuses on comparing protein sequences. Since our method minimizes dependencies in the intermediate steps, the time complexity can be reduced to $\Theta(\log^2 n)$ in the presence of satisfactory number of parallel processors. The major advantage of our method its paralleizable nature. Three components of SEAL are parallelizable: a) submatrices, b) the diagonals, and c) maximum contiguous subarray. In our method, the sub-problems (the maximum contiguous sub-array) can further be solved as divide-and-conquer problems. Our major contribution is

to increase the parallel components of the alignment algorithm without sacrificing the accuracy of the algorithm.

This paper is organized as follows. The following section provides the overview of our methodology. Section 3 explains our algorithms. Section 4 analyzes experiments and complexity of our methods. The last section concludes our paper.

2 SYSTEM AND METHODS

The objective of our method is to design a parallelizable sequence alignment method which is much more sensitive to distant relationships than existing heuristic methods. SEAL is a novel sequence alignment method which combines maximum contiguous sub-array solution and divide-and-conquer approach. The maximum contiguous sub-array solution finds the longest consecutive positive integers given an array of integers. It begins by exploring the entire search space and then recursively explores sub-spaces. A maximum contiguous sub-array solution is applied to each diagonal array in a search space and scores are calculated. Scores of contiguous segments are compared to find the maximum segment for that search space. The prefix and suffix sub-regions of this segment are explored recursively for maximum scoring segments in those regions and sub-regions (Fig. 1). The process continues until the search space spans a single cell. The basic divide-and-conquer methodology used in SEAL is presented in Fig. 2. Borders are introduced to each maximum contiguous segment in an improved version of SEAL called iSEAL. BLOSUM62 amino acid scoring matrix is used in this method.

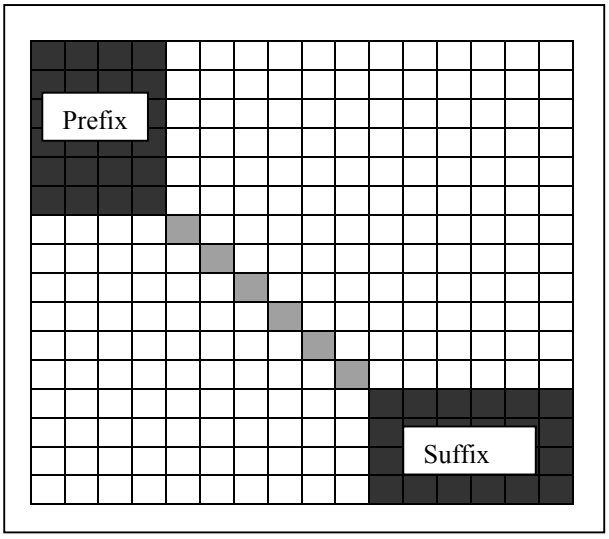

Fig. 1. Prefix and suffix regions to be analyzed after finding a good matching sequence.

Divide-and-Conquer

The challenging component of SEAL is to divide the alignment problem into subproblems such that the outcomes of these subproblems can be solved independently and then merged to find the alignment. We have the following observations:

1. A good matching sequence corresponds to a diagonal in the matrix whereas horizontal or vertical matching corresponds to a gap in the query or database sequence.
2. If there is a good matching sequence without any gaps, the corresponding segments from the sequences will not be used again for alignment.

The initial subproblem turns into finding the maximum subarray problem. The maximum subarray problem is the task of finding the subsequence that has the highest sum of its consecutive values. The number of diagonals for a query sequence of *m* and a database sequence of *n* is *(m+n-1)*. Note that each diagonal is a 1-dimensional sequence of values (scores between corresponding amino acids based on BLOSUM62 matrix).

After finding the best maximum contiguous array among all diagonals, this matching sequence divides the problem into 3 regions: diagonal region, suffix, and prefix. This further helps us eliminate all the regions that can be aligned with the identified matching region. The suffix and prefix regions are the next areas to be explored (Fig. 1).

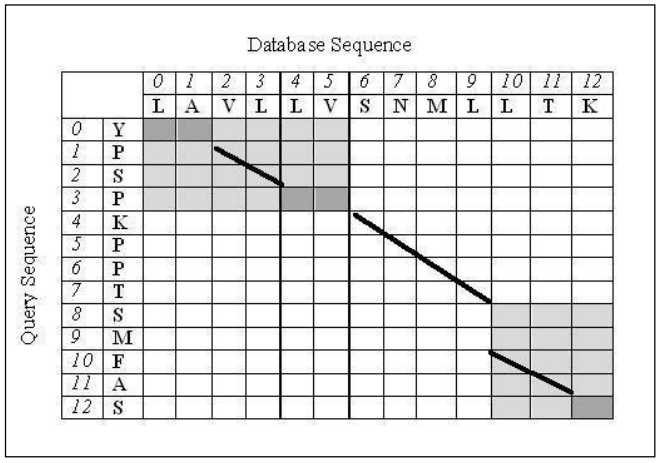

Fig. 2. Divide-and-Conquer exploration with maximum contiguous subsequence in each sub-matrix.

Note that our algorithm introduces gaps naturally into the result so that the user does not need to worry about selecting a gap parameter.

3 ALGORITHM

The basic steps involved in SEAL are as follows:

SEAL (M X N Matrix)
#Input: A M X N matrix. M and N are the lengths of query and #database sequences respectively
#Output: An alignment of the query and database sequence.

Step 1. Create a matrix of BLOSUM scores for the given query and database sequence. Divide M X N matrix into diagonals.
Step 2. Find the maximum contiguous sub-array for each diagonal of M X N matrix.
Step 3. Compare the scores of all the maximum contiguous segments for each diagonal and find the highest scoring segment for that matrix.
Step 4. Divide the matrix into two sub-matrices covering the prefix and suffix search spaces of the highest scoring segment. Sub-matrix1 spans cell [0, 0] to the start coordinates of the highest scoring segment. Sub-matrix2 occupies the search space from end coordinates of highest scoring segment to cell [M, N]. Repeat Step 1 to Step 3 for sub-matrices.

Step 5. Repeat Step 1 to Step 4. The recursion stops when the matrix or sub-matrix has no positive scores or it reaches the ends of the matrix.

Step 6. The highest scoring segments from all the sub-matrices are joined and padded with gaps to make the final alignment.

Borders

The original SEAL algorithm introduces challenges about the length of the segment. Usually longer matching sequence is preferred to shorter sequence matching sequence. However, this may also eliminate some good segments to be matched. Assume that a sequence such as [1, 1, 1, 5, 0, -1, 1] is given. The sum of the maximum subarray is 8. The last 3 values do not increase the score of the sequence but using those returns a longer matching sequence. The search regions for prefix and suffix start from the beginning and end of the diagonal, respectively. The second best matching sequence may intersect with the best matching subsequence. However, since the corresponding portions of the query or database sequence are used for the best subsequence, those portions cannot be used again and this may lead that the second best matching subsequence is no longer a good matching subsequence.

Fig 3 represents this scenario. The sum of scores are as follows for three diagonals, *A*, *B*, and *C*: sum(A)=7, sum(B)=3, and sum(C)=10. Diagonal *C* has a better matching sequence than diagonal *A*. The last two cells of *A* intersect with the first two cells of *C*. If diagonal *C* is selected, those cells will not appear in the prefix space to be further analyzed. Since those cells are ignored, diagonal B is selected instead of diagonal A. On the other hand, the first two cells of diagonal C could be ignored and then diagonal A with a good score of 7 could be selected.

This problem can be avoided or minimized if special consideration is given to the borders of the contiguous subarray. The contiguous segment of a matrix may include high and low scores. It is clear that the highest score values should certainly be included in the final alignment. A method which separates the good scoring part of the maximum contiguous segment from the rest of the weak scoring segment is designed. Borders separate the good scoring part of the maximum contiguous segment. Beginning from the end coordinates of a segment and moving inwards by deducting the current cell score from the total score of a segment, the border is defined at the point

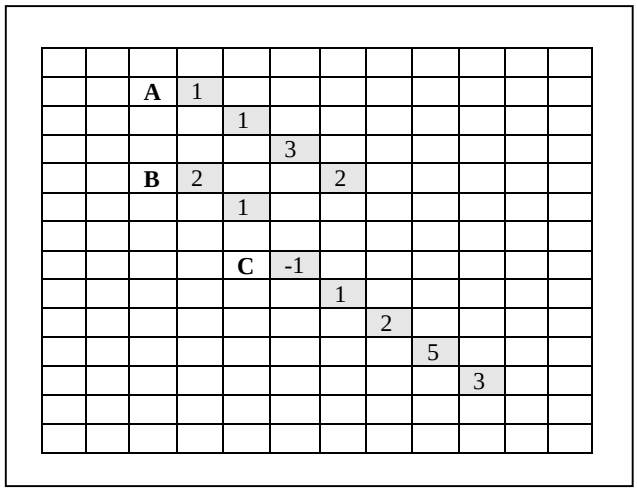

Fig. 3. The effect of low scores at the end of matching subsequence.

where the score of the segment falls by at most 2. This is repeated for both ends of a segment to define hard borders. An example of a maximum contiguous segment and the defined borders is given as follows.

Maximum contiguous segment: 1 1 1 5 0 -1 1
Segment with borders(marked as red lines):1 1 | 1 5 | 0 -1 1

The portions outside the hard borders can be reused in further stages of divide-and-conquer method. The algorithm is improved and named as iSEAL. The basic steps involved in iSEAL are as follows:

iSEAL (M X N Matrix)
#Input: A matrix M X N with query sequence and database
#sequence. M and N are the lengths of query and database
#sequence respectively.
#Output: An alignment of query and database sequences.

Step 1. Create a matrix of BLOSUM scores for the given query and database sequence. Divide M X N matrix into diagonals.

Step 2. Find the maximum contiguous sub-array for each diagonal of M X N matrix.

Step 3. Compare the scores of all the maximum contiguous segments for each diagonal and find highest scoring segment for that matrix.

Step 4. For each highest scoring segment define borders.

Step 5. Repeat Step 1 to Step 4 for two sub-matrices on either ends of the borders of the maximum contiguous segment.

Step 6. Repeat Step 1 to Step 5 until the sub-matrix has no positive scores or ends of matrix are reached.

Step 7. The highest scoring segments from all the sub-matrices are joined and padded with gaps to make the final alignment.

4 IMPLEMENTATION

4.1 Experiments

We have implemented our algorithm using Perl. A database of 45 transcription factor protein sequences downloaded from NCBI serves as a basis for all experiments. Each sequence from the database is aligned with a test database sequence, using all the alignment methods developed, so as to get a comparative view of alignments.

Preparing Test Database. To prepare a test database, protein sequences of transcription factors of all organisms are downloaded in FASTA format from NCBI. A database of these sequences was created with standalone BLAST version using *makeblastdb* command. The query is searched against the database using the BLASTP command. The additional parameters such as the threshold value can be specified according to the requirements. Gap penalty is not a subject under focus for the present research, so the lowest values supported by BLAST are used. For the experiments, default values are threshold = 10.0, gapopen = 6 and gapextend = 2.

Identifying Motifs. Multiple sequence alignment of the database of transcription factors sequences provides a useful insight into the evolutionary relationships. Since all the proteins in the database are transcription factors, they are expected to have similar functional domains. Multiple sequence alignment method lines up a number of sequences optimally by bringing the greatest number of similar characters into alignment. Most popular tools for multiple sequence

```
  BLOCK1
A  GSHMGIQETDPLTQLSLPPGFRFYPTDEELMVQYLCRKAAGYDFSLQLIAEIDLY  95
B  GSHMEEPSDLEELEQFAKTFKQRRIKLGFTQGDVGLAMGKLYGNDFSQTTISRFE 100
C  MRYLELGCISKTNKLFQKLQDLNPLLNIEIEAYSCKSSRRQRGRFVEKPLGYLLS  87
D  MTVSGSSGRAKRSTTQAKAAEQMATKPQARRASEAGTSAVVKGFSHIPHGNTALS  78
E  MAVGPPTGGSGNPPQIPVQPHPILAPSPLFALPTLNFTASQVATVCETLEESGDI  71
F  MGRRKIEIKRIENKSSRQVTFSKRRNGLIDKARQLSILCESSVAVVVVSASGKLY  71
G  MSLVGGFPHHPVVHHEGYPFAAAAAAAAAAAAASRCSHEENPYFHGWLIGHPEMSP  43
H  MNSFSAFSEMFGSDYESPVSSGGDYSPKLATSCPKKPAGRKKFRETRHPIYRGVR  77
I  MSLVGGFPHHPVVHHEGYPFAAAAAAAAAAAAASRCSHEENPYFHGWLIGHPEMSP  43
J  MSRKLKKTNLFNKDVSSLLYAYGDVPQPLQATVQCLDELVSGYLVDVCTNAFHTA  77
K  MAGHLASDFAFSPPPGGGDGSAGLEPGWVDPRTWLSFQGPPGGPGIGPGSEVLGI  67
```

```
  BLOCK2
A  KFDPWVLPNKALFGEKEWYFFSPRDR 100
B  ALNLSFKNMAKLKPLLEKWLNDAENL  74
C  ALELRFPDYDFCGESWGSFRRKTLAE  87
D  VKADEARLASITDEKERKRLKRLLRN  67
E  ERLARFLWSLPVAHPNISELDRSEAV  74
F  DSSSVIVSTGKYKNFTIFLTIPFLHV  77
G  PDYSMALSYSPEYASGAAGLDHSHYG  40
H  QRNSGKWVCELREPNKKTRIWLGTFQ  84
I  PDYSMALSYSPEYASGAAGLDHSHYG  40
J  QNSQRNKLRLEDFKFALRKDPIKLGR  52
K  SPCPPAYEFCGGMAYCGPQVGLGLVP  90
```

Fig. 4. Sample output of BLOCKS leading two blocks on sequences A: gi|47169278; B: gi|42543138; C: gi|19069247; D: gi|116000610; E: gi|108884304; F: gi|2829920; G: gi|12545384; H: gi|47605752; I: gi|47117899; J: gi|6323540; K: gi|125490392.

```
Query: gi|6323540|ref|NP_013611.1| TFIID subunit (19 kDa), involved in RNA polymerase II transcrip-
tion initiation, similar to histone H4 with atypical histone fold motif of Spt3-like transcription
factors [Saccharomyces cerevisiae]

Test Database Sequence: gi|125490392|ref|NP_038661.2| POU domain, class 5, transcription factor 1
[Mus musculus]
```

```
Gapped BLAST output:(Score=36)

Query  2    SRKLKKTNLFNKDVSSLLYAYGDVPQP-LQATVQCLDEL  39
            +RK K+T++ N+   SL   +   P+P LQ      ++L
Sbjct  222  ARKRKRTSIENRVRWSLETMFLKCPKPSLQQITHIANQL  260
```

```
iSEAL output: (Score = 202)
Query: m s - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Db   : l s l k n m c k l r p l l e k w v e e a d n n e n l q e i c k s e t l v q a

Query: r k l k k t n l f n k d v s s l l y a y g d v p q p - l - q a t v q c l d e l v s g
Db   : r k r k r t s i e n r v r w s l e t m f l k c p k p s l q q i t - - - - - - - - - -

Query: y l v d v c t n a f h t a q n s q r n k l r l e - d - - - - - f - - - - - k f a l
Db   : - - - - - - - - - - h i a - - - - - n q l g l e k d v v r v w f c n r r q k - g -

Query: r k d p i k l g r a e e l i a t n k e q q v t d d d e e a - k k q f n e t d n q n s l k r y
Db   : k r s s i e y s q r e e y e a t g - - - - - - - - - - - a v s - - - - - - - - - - - - - - - -

Query: r e e d e e g d k q g p k q f n e t d n q n s - - - - - l k r y r e e d
Db   : - - - - - - - - - - - - - - h f g - t p g y g s p h f t t l - - y - - - -

Query: e - - - - e g d - - - e m v t d d d e e a a g r n s a k q s t d s k a t k i r k q g p - - k
Db   : s v p f p e g e a f p s v v t - - - - - a l g - - - - - - - - - s - - - - - - - - - p m h s
```

Fig. 5. Sample alignment comparison between gapped BLAST and iSEAL.

	Id. No. (gi no)	Dynamic Prog. Score	Gapped BLAST-score	iSEAL score	Std. BLOCKS motifs	Motifs detected by gapped Blast	Motifs detected by iSEAL
1	47169278	439	22	164	2	2	2
2	42543138	557	430	516	2	2	2
3	19069247	417	36	248	2	2	2
4	116000610	414	32	252	2	1	2
5	6323540	427	36	202	2	1	2
6	108884304	555	70	265	2	2	2
7	2829920	464	31	222	2	1	2
8	12545384	498	33	175	2	1	0
9	47605752	516	34	317	2	0	2
10	47117899	498	33	175	2	1	0
11	4885665	501	36	243	2	0	1
12	12643786	526	28	259	2	2	2
13	4504573	444	35	245	2	0	2
14	68989258	531	41	227	2	1	1
15	31982933	380	24	156	2	0	2
16	13958612	417	38	190	2	2	1
17	112253397	395	24	223	2	2	1
18	54039792	432	39	214	2	2	2
19	3913130	533	30	261	2	2	2
20	127704	415	36	136	2	2	1
21	31317299	450	26	147	2	1	1
22	31317297	452	2	152	2	1	1
23	113594633	463	30	218	2	2	2
24	124360101	350	19	176	2	1	2
25	124359419	497	26	114	2	2	2
26	88963532	358	26	162	2	2	2
27	88963530	350	26	156	2	2	2
28	60498987	376	19	248	2	0	2
29	124360009	489	25	167	2	1	2
30	124359882	410	41	274	2	2	2
31	89113792	350	26	158	2	1	2
32	89113790	352	24	151	2	1	2
33	89113784	323	26	134	2	2	2
34	89113782	351	27	177	2	2	2
35	89113780	358	26	158	2	2	2
36	89113778	343	27	174	2	2	2
37	124054218	281	26	140	2	2	2
38	124013584	431	33	255	2	2	2
39	729811	380	27	182	2	2	2
40	729810	376	27	163	2	2	1
41	81673105	451	29	162	2	2	2
42	47117699	498	33	175	2	1	1
43	122934930	428	27	206	2	2	2
44	17981708	454	40	228	2	2	2
45	122053927	376	38	196	2	2	2

Table 1. Comparison of alignment methods on a database of 50 transcription factors.

alignment are ClustalW [13] and PileUp [14]. ClustalW is a progressive method of multiple sequence alignment in which most related sequences are aligned and then progressively less related sequences or groups of sequences are added. Alignments based on localized sequences give information about domains and motifs. The common blocks or motifs are extracted from unaligned sequences based on previously calculated motifs from known gene families. There are widely used web tools like Profile [14] and BLOCKS [14]. Profile identifies highly conserved portion of the sequence alignment and constructs a score profile which includes score for substitutions and gaps. BLOCKS tool concentrates on the conserved regions of the alignment with substitutions without gaps. Since SEAL involves placing gaps between locally optimal sequences, BLOCKS is a good tool to provide standard motifs without gaps. In Table 1, the column for Std. BLOCKS motifs indicates the number of motifs found by the BLOCKS tool. Multiple sequence alignment of some sequences by BLOCKS is shown in Fig. 4 to give a measure of the number of significant motifs (protein segments which can function independently) identified by alignment methods.

The Dynamic Programming alignment (Needleman- Wunsch) serves as a standard for each alignment as it gives optimal alignment with the maximum score. BLAST (without gaps) and gapped BLAST are very popular tools and therefore can serve as good benchmarks. The methods developed in this research SEAL and improved SEAL (with borders) are used for alignment. The scores of the alignments produced with each method are compared. Fig. 5 provides an example of comparison of alignments between gapped BLAST and iSEAL.

In a total of 45 sequence alignments, all the scores obtained by iSEAL are higher than those obtained by gapped BLAST. There are 14 cases in which motifs detected by iSEAL are undetected by gapped BLAST, 6 cases where motifs detected by gapped BLAST are undetected by iSEAL. In the remaining cases, both BLAST and iSEAL detected the same motifs. Therefore, according to the results obtained, iSEAL gives 87% performance efficiency in detecting motifs and BLAST gives 75%. The range for the length of query sequences is [81..217]. In our independent dataset, we have tried sequences whose length between about 60 and 700.

We have looked into a number of newer alignment programs. In essence, they maintain the similarity of the traditional algorithms. However, methods such as AlignMe, can use additional information such as PSSM for alignment with low sequence identity. In our case, we apply plain alignment comparison and compare with AlignM-Fast. For the example provided in Fig. 4, AlignMe-Fast barely hits both motifs. The size of alignments is almost half of what iSEAL detects in Fig. 5. The aligned parts with respect to the blocks are highlighted below:

```
query  MSRKLKKTNLFNK-----DVSSLL----------------
db     MAGHLASDFAFSPPPGGGDGSAGLEPGWVDPRTWLSFQGP

query  ----TAQNSQRNKL---RLEDFKFALRKDPIKLG------
db     KVEPTPEESQDMKALQKELEQFAKLLKQKRITLGYTQADV
```

SEAL combines the divide-conquer method with the maximum contiguous sub-array solution. Locally best alignments are found and combined to give a complete alignment between two sequences. In this work, each sequence of a database of 45 query sequences is aligned with a test database sequence using Dynamic Programming, BLAST (without gaps), gapped BLAST, SEAL and iSEAL sequence alignment methods. The alignment scores obtained by SEAL and iSEAL are consistently higher than BLAST in all 50 cases. In 60% of cases, all the motifs detected by the BLOCKS program are detected by both SEAL and gapped BLAST. In 28% of cases, SEAL detects motifs which are undetected by BLAST and in 12% of cases BLAST detects motifs undetected by SEAL. Therefore, SEAL is a better scoring method than BLAST and produces good quality alignments. Table 1 provides the comparison of alignment methods for each sequence. We applied one-tailed t-test, since we are interested in detecting motifs or sensitivity of detection rather than accuracy. We have obtained a p-value of 0.048 and this is enough to reject the null hypothesis.

4.2 Complexity Analysis

In this section, we provide the complexity analysis for running algorithm sequentially and in parallel. Since our method minimizes the dependencies between steps of the procedure, it is possible to reach low complexity when run in parallel with enough number of

processors. We provide time complexity of sequential algorithm to explain the complexity of the parallel algorithm.

Sequential Complexity

The complexity analysis includes two major computations. These include finding the maximum contiguous sub-array of all the diagonals in a matrix and dividing the matrix into two sub-matrices. The time complexity for finding the maximum contiguous sub-array for a single diagonal is $O(n)$ using Kadane's algorithm (Bentley; 1984).The complexity for finding the maximum contiguous sub-array for all the diagonals in a *n x n* matrix and the highest scoring segment among them is $O(n^2)$.

SEAL finds the highest scoring segment and splits the matrix into 3 parts: the sub-matrix that holds the highest-scoring segment, the sub-matrix before the segment and the sub-matrix after the segment. In the worst case, matrix is split into two when the length of segment is very small and the best maximum contiguous sub-array appears close to the boundaries of the matrix. On the average, each sub-matrix has the quarter of the matrix size before the split. The time complexity of SEAL denoted by T(n), can be represented by the following recursion:

$$T(n) = 2T\left(\frac{n}{2}\right) + cn^2 \qquad (1)$$

assuming that the split divides the matrix into roughly two equal-sized partitions and assuming the length of maximum contiguous array is 0 (i.e., in reality, it cannot be 0, this is just a case of worst case for split). The time complexity of SEAL, after solving this equation is $O(n^2)$. Actually, in the recurrence part, the coefficient b of $T(n/b)$ can be more than 2 if the length of the common subsequence is a function of *n*.

Parallel Complexity

The advantage of SEAL is its parallelizable nature. The traditional divide-and-conquer algorithms parallelize $aT(n/b)$ of the recurrence relation. If there are a number of processors, each processor should execute $T(n/b)$ and run concurrently. In such a case, if $f(n) \in \Theta(n^c)$, the time complexity becomes $\Theta(n^c)$. The SEAL algorithm does not only parallelize $aT(n/b)$ but also parallelizes $f(n)$. It parallelizes $f(n)$ at two levels. Therefore, three components of SEAL are parallelizable: a) submatrices (aT(n/b)), b) the diagonals, and maximum contiguous subarray of each diagonal. a) The divided sub-matrices can be processed independently by different processors. b) The diagonals in each matrix also can be processed independently for finding maximum contiguous sub-array. c) The maximum contiguous subarray can be computed in parallel. When SEAL is parallelized based on the parallel processing of sub-matrices, the time complexity can be represented by the following recurrence tree.

The function f(n) should have low time complexity to lower the overall time complexity. The complexity of finding maximum contiguous subarray is $O(\log n)$ in the presence of $(\frac{n}{\log n})$ parallel processors (Perumella; 1995). Since each diagonal can be executed in parallel, the complexity of finding the maximum contiguous subarray among all diagonals is $O(\log n)$. The previous recurrence relation in (1) turns into

$$T(n) = 2T\left(\frac{n}{2}\right) + c\log n.$$

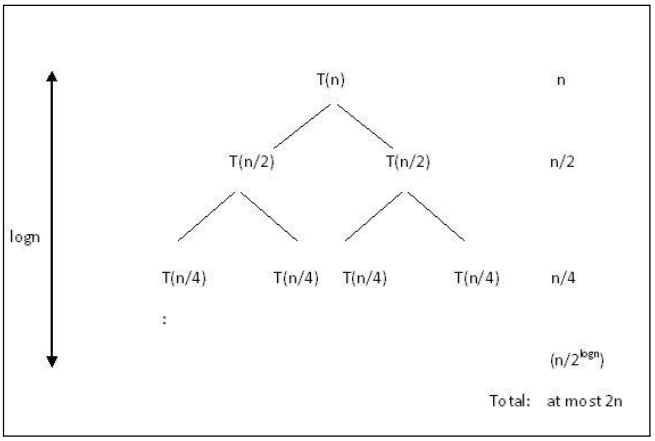

Fig. 5. Recurrence tree for SEAL

Each sub-matrix can be computed in parallel further. Then the recurrence relation becomes

$$T(n) = T\left(\frac{n}{2}\right) + c\log n$$

The complexity of this recurrence relation is $\Theta(\log^2 n)$ in the presence of satisfactory number of processors. In this expression, the coefficient b is 2. As long as $b>1$, this $\Theta(\log^2 n)$ time complexity still holds. This indicates that the submatrices do not even need to be split into half. The recurrence relation can be written as:

$$T(n) = T\left(\frac{n}{b}\right) + c\log n$$

where b>1 and the complexity is still $\Theta(\log^2 n)$. Our work shows the open area for the research to be conducted for sequence alignment by reducing the dependencies among steps.

In terms of space complexity, each diagonal is processed independently, and the total size of diagonals is the size of the complete matrix. At least $O(MxN)$ space is required. The maximum contiguous subarray problem requires linear space. The space complexity of this system is $O(MxN)$.

5 DISCUSSION

SEAL detects and aligns the major similar segments between two sequences and is also sensitive to small similar fragments in other parts of the alignment. It gives a better alignment and score for highly similar sequences when compared to other heuristic methods. It is sensitive to distantly related sequences and therefore helps in better function prediction of unknown proteins. SEAL provides an ease of usage without the burden of specifying gap penalties. Gap penalty parameters, namely, gap open and gap extension costs, need not be specified for SEAL as they are implicitly introduced within the alignment. This overcomes the disadvantages of using the gap penalty threshold value, as used in other heuristic methods, which may not be good for all alignments. The algorithm can be parallelized to reduce time complexity.

In our experiments, we have obtained promising results for SEAL. However, there is still space for improving SEAL. In the proposed approach, we tried to keep the interaction between subproblems minimal. We have used a Greedy approach: the segment with the highest scores should be part of the final alignment. The borders of the selected segment may eliminate some good matching seg-

ments. To avoid this problem, we introduced hard borders where regions outside the hard borders can be reused in the subproblems. This minimized the problem of intersecting diagonals. However, this component is a heuristic component of our algorithm. We believe that this heuristic can be avoided and optimal results can still be obtained. However, it may limit the parallelization of the algorithm. Further research has to be done to study intersecting diagonals. Parallel methods may be developed to compute maximum segments on each diagonal concurrently with independently processing sub-matrices. With enough number of parallel processors, it is possible to reduce the complexity to $\Theta(\log^2 n)$ using our method. We need high number of processors to truly evaluate the performance of our system. In basic parallel environments, our system does not outperform the other techniques in running time. As future work, the parallel implementation of our method should be evaluated on GPU-based architectures or high-performance computing servers. Our proposed algorithm can also be incorporated or used by other alignment algorithms.

REFERENCES

Jon Bentley. (1984) Programming Pearls: Algorithm Design Techniques, *Communications of the ACM*, **25**(9), 865-871.

Choi, Y. (2012). A Fast Computation of Pairwise Sequence Alignment Scores Between a Protein and a Set of Single-locus Variants of Another Protein. In *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine* (pp. 414–417). New York, NY, USA: ACM. http://doi.org/10.1145/2382936.2382989

Dong Dai; Xi Li; Chao Wang; Xuehai Zhou, "Cloud Based Short Read Mapping Service," Cluster Computing (CLUSTER), 2012 IEEE International Conference on , vol., no., pp.601,604, 24-28 Sept. 2012

Díaz, D., Esteban, F. J., Hernández, P., Caballero, J. A., Dorado, G., & Gálvez, S. (2011). Parallelizing and optimizing a bioinformatics pairwise sequence alignment algorithm for many-core architecture. *Parallel Computing*, *37*(4–5), 244–259. http://doi.org/10.1016/j.parco.2011.03.003

Huang, X., & Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, *12*(3), 337–357. http://doi.org/10.1016/0196-8858(91)90017-D

D. Mathog, "Parallel BLST on split databases", *Bioinformatics*, vol. 19(4), 2003.

Arun Krishnan. (2005) GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework, *Concurrency and Computation: Practise and experience*, **17**(13), 1607-1623.

Li, W., Cowley, A., Uludag, M., Gur, T., McWilliam, H., Squizzato, S., … Lopez, R. (2015). The EMBL-EBI bioinformatics web and programmatic tools framework. *Nucleic Acids Research*. http://doi.org/10.1093/nar/gkv279

Li, H. and R. Durbin, Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics, 2010. 26(5): p. 589-595

Li, Y., Patel, J. M., & Terrell, A. (2012). WHAM: A High-Throughput Sequence Alignment Method. *ACM Trans. Database Syst.*, *37*(4), 28:1–28:39. http://doi.org/10.1145/2389241.2389247

Heshan Lin *et al.* (2008) Massively parallel genomic sequence search on the Blue Gene/P architecture, *Conference on High Performance Networking and Computing*, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, article **33**.

H. Lin, *et al.* (2005) Efficient Data Access for Parallel BLAST, *IEEE International Parallel & Distributed Processing Symposium*

McWilliam, H., Li, W., Uludag, M., Squizzato, S., Park, Y. M., Buso, N., … Lopez, R. (2013). Analysis Tool Web Services from the EMBL-EBI. *Nucleic Acids Research*, *41*(W1), W597–W600. http://doi.org/10.1093/nar/gkt376

Jones, N.C., Pevzner, P., 2004. An Introduction to Bioinformatics Algorithms. MIT Press.

O'Driscoll, A., Belogrudov, V., Carroll, J., Kropp, K., Walsh, P., Ghazal, P., Sleator, R.D., 2015. HBLAST: Parallelised sequence similarity - A Hadoop MapReducable basic local alignment search tool. J. Biomed. Inform. 54, 58–64. doi:10.1016/j.jbi.2015.01.008

K. Perumalla and Narsingh Deo (1995), Parallel Algorithms for Maximum Subsequence and Maximum Subarray, *Parallel Processing Letters* 1995 05:03 , 367-373

W. R. Pearson. (1995) Comparison of methods for searching protein sequence databases, *Protein Sci*, **4**, 1147-1160.

Shpaer E. G. *et al.* (1996) Sensitivity and selectivity in protein similarity searches: a comparison of Smith-Waterman in hardware to BLAST and FASTA, *Genomics*. **38**(2), 179-191.

Soding, J. (2005). Protein homology detection by HMM-HMM comparison. *Bioinformatics*, *21*(7), 951–960. http://doi.org/10.1093/bioinformatics/bti125

Stamm, M., Staritzbichler, R., Khafizov, K., & Forrest, L. R. (2014). AlignMe—a membrane protein sequence alignment web server. *Nucleic Acids Research*, *42*(W1), W246–W251. http://doi.org/10.1093/nar/gku291

Soding, J., 2005. Protein homology detection by HMM-HMM comparison. Bioinformatics 21, 951–960. doi:10.1093/bioinformatics/bti125

Stamm, M., Staritzbichler, R., Khafizov, K., Forrest, L.R., 2014. AlignMe—a membrane protein sequence alignment web server. Nucleic Acids Res. 42, W246–W251. doi:10.1093/nar/gku291

Stoye, J., 1998. Multiple sequence alignment with the Divide-and-Conquer method. Gene 211, GC45–56.

Stoye, J., 1997. Divide-and-Conquer Multiple Sequence Alignment (Dissertation Thesis). Universität Bielefeld, Forschungsbericht der Technischen Fakultät, Abteilung Informationstechnik.

Stoye, J., Moulton, V., Dress, A.W., 1997. DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. Comput. Appl. Biosci. CABIOS 13, 625–626.

Tönges, U., Perrey, S.W., Stoye, J., Dress, A.W.M., 1996. A general method for fast multiple sequence alignment. Gene 172, GC33–GC41. doi:10.1016/0378-1119(96)00123-0

Mingming Sun; Xuehai Zhou; Feng Yang; Kun Lu; Dong Dai, "Bwasw-Cloud: Efficient sequence alignment algorithm for two big data with MapReduce," Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference on the , vol., no., pp.213,218, 17-19 Feb. 2014

Tang, C. L., Xie, L., Koh, I. Y. Y., Posy, S., Alexov, E., & Honig, B. (2003). On the role of structural information in remote homology detection and sequence alignment: new methods using hybrid sequence profiles. *Journal of Molecular Biology*, *334*(5), 1043–1062.

Hao Wang. *et al.* (2003) BLAST++: BLASTing queries in batches, *Bioinformatics*, **19**(17), 2323-2324.

Jiren Wang and Qing Mu. (2003) SOAP-HT-BLAST: high-throughput BLAST based on Web services, *Bioinformatics*, **19**(14), 1863-1864.

White C.T. (1991) BioSCAN: a VLSI-based system for biosequence analysis**,** *Computer Design: VLSI in Computers and Processors, ICCD '91. Proceedings, 1991 IEEE International Conference*, **14**(16), 504-509.