Pairwise Sequence Alignment using Bio-Database Compression by Improved Fine Tuned Enhanced Suffix Array

Arumugam Kunthavai[1], SomasundaramVasantharathna[2], and Swaminathan Thirumurugan[1]
[1]Department of Computer Science and Engineering/IT, Coimbatore Institute of Technology, India
[2]Department of Electrical and Electronics Engineering, Coimbatore Institute of Technology, India

Abstract: *Sequence alignment is a bioinformatics application that determines the degree of similarity between nucleotide sequences which is assumed to have same ancestral relationships. This sequence alignment method reads query sequence from the user and makes an alignment against large and genomic sequence data sets and locate targets that are similar to an input query sequence. Existing accurate algorithm, such as smith-waterman and FASTA are computationally very expensive, which limits their use in practice. The existing search tools, such as BLAST and WU-BLAST, employ heuristics to improve the speed of such searches. However, such heuristics can sometimes miss targets, in which many cases are undesirable. Considering the rapid growth of database sizes, this problem demands ever-growing computation resources and remains as a computational challenge. Most common sequence alignment algorithms like BLAST, WU-BLAST and Sequance Comparasion Tool (SCT) searches a given query sequence against set of database sequences. In this paper, Biological Data Base Compression Tool using Minimum Perfect Hash Function (BioDBMPHF) tool has been developed to find pair wise local sequence alignment by preprocessing the database. Preprocessing is done by means of finding Longest Common Substring (LCS) from the database of sequences that have the highest local similarity with a given query sequence and reduces the size of the database based on frequent common subsequence. In this BioDBMPHF tool fine-tuned enhanced suffix array is constructed and used to find LCS. Experimental results show that hash index algorithm reduces the time and space complexity to access LCS. Time complexity to find LCS of the hash index algorithm is $O(2+\gamma)$ where 'γ' is the time taken to access the pattern. Space complexity of fine-tuned enhanced suffix array is 5n bytes per character for reduced enhanced Longest Common Prefix (LCP) table and to store bucket table it requires 32 bytes. Data mining technique is used to cross validate the result. It is proved that the developed BioDBMPHF tool effectively compresses the database and obtains same results compared to that traditional algorithm in approximately half the time taken by them thereby reducing the time complexity.*

Keywords: *Sequence alignment, enhanced suffix array, compression, minimum perfect hash function, data mining.*

Recevied October 25, 2012; accepted January 6, 2013; published online August 17, 2014

1. Introduction

In the recent years, advances in molecular biology and the data available for research in biological field have facilitated the increasingly rapid sequencing of large portions of the genomes of several species. Biology has increasingly turned into a data-rich science, so the need for storing [9] and communicating large datasets has grown tremendously. The apparent examples are the nucleotide sequences and the protein sequences. Bioinformatics provide solutions for these issues, challenges, new possibilities and new algorithms created by these databases. Nucleic acid sequences cover the majority of such databases. The human genome project and many other efforts in molecular biology aim to sequence chromosomes of humans and other species are to expose the genetic information contained in these sequences and to find the homology between the sequences by finding similarity with the help of sequence analysis.

Sequence analysis is the most common task in bioinformatics. It is a way of arranging the primary sequences of DNA, RNA or protein to identify regions of similarity that may be a consequence of functional, structural or evolutionary relationships between the sequences. This paper is mainly focused on reducingthe time complexity of DNA sequence analysis. DNA consists of only four letters A, G, C and T. Sequence analysis is useful for identifying sequence similarity, producing phylogenetic trees or evolutionary tree which is a branching diagram or "tree" showing evolutionary relationships among various biological species or other entities based upon similarities and differences in their genetic characteristics and developing homology models of protein structures. There are two types of alignments local and global alignment.

Local sequence alignment plays a main role in the analysis of DNA and protein sequences [4, 5, 6, 17]. It is the basic step of several other applications like detecting homology, finding protein structure and function, deciphering evolutionary relationships, etc., there exist a number of local sequence alignment programs that use well-known algorithms BLAST and FASTA [4, 13] or their heuristic versions PSI-BLAST, FASTA3 [4, 5, 14, 15, 16]. Database search is a special case of pair wise local sequence alignment, where the second sequence is a database which consists of many sequences. Recently, there have been many improvements in alignment program features [7] using difference blocks, new parameters and multiple scoring matrices, in an attempt to incorporate more biological features in the alignment algorithm. Considering the rapid growth of database sizes, this problem demands ever-growing computation resources, and remains as a computational challenge. Existing Sequence Comparison Tool (SCT) [7] preprocesses the database by reducing the size of the database using extended suffix tree. The space consumption of the extended suffix tree is a bottleneck in large-scale applications such as genome analysis.

Hence, in this paper Biological Data Base compression tool using Minimum Perfect Hash Function (BioDBMPHF compression tool) has been developed to perform local pairwise sequence alignment. This method reduces the size of the database using improved fine-tuned enhanced suffix array. Objective of this paper is to improve the computation time of the alignment compared to the existing methods while preserving the accuracy by reducing the size of the database.

Instead of comparing user given sequence withthe large database, the size of the database is reduced with the help of improved fine-tuned enhanced suffix array using Minimum Perfect Hash Function (MPHF) using newly developed hash function hash index. The main idea is to find matched patterns of the query sequence and identify sequences in the database which share a large number of these matched patterns, thereby compressing the size of the database to very few sequences. Consistency of the result is verified using 7 fold cross validation technique.

The paper is organized as follows: Pair wise sequence is explained in section 2. Section 3 presents the basic definition of suffix array and enhanced suffix array followed by how to find Latest Common Substring (LCS) using MPHF which is used for bio data base compression. Bio database compression is explained in section 4. Experiments and results are presented in section 5.

2. Pairwise Sequence Alignment

Sequence alignment is simply a special case of string matching, a research field with a long history in computer science. The alignments are simply the mathematical models whose behaviours can be modified using parameters. Different models exist, which are designed to encapsulate a variety of physical characteristics of biological sequences.

Pair wise sequence alignments are used to find diagnostic patterns that characterize the two DNA families; to detect or demonstrate homology between new sequences and existing families of sequences. Two general models view alignments in different ways: The first considers similarity across the full extent of the sequences (a global alignment); the second focuses on regions of similarity in parts of the sequences only (a local alignment). It is important to understand these distinctions, to appreciate that sequences are not uniformly similar and there is no value in performing a global similarity on sequences that have only local similarity.

In many biological applications, two DNA sequences may not be highly similar in their entire length, but may contain regions that are highly similar, because only some internal sections of those strings may be related. While comparing such DNA sequences, local alignment becomes critical because local similarity finds out highly conserved regions in the DNA sequence and it is the preferred choice for biological applications. The reason for choosing local alignment, algorithm is it highlights conserved regions between two sequences and it yields more homological information. Database search is a special case of pair wise local sequence alignment, where the second sequence is a database which consists of many sequences. Every sequence in the database is aligned with the user given query sequence. If the size of the database is large, time taken to align all the sequence is also high. So, instead of comparing user given sequence with the large database, the size of the database is reduced using the developed BioDBMPHF compression tool using improved fine-tuned enhanced suffix array.

3. Enhanced Suffix Array

In recent years, the enhanced suffix array [3, 12] became the data structure of choice for indexing biological data and solving versatile tasks. This is due to its reduced memory consumption compared to the suffix tree and its improved cash performance. Enhanced suffix array is constructed using suffix array with additional tables Longest common prefix table (Lcptab) and child table.

3.1. Basic Definition

Let S be a string of length ‘n’ over an ordered alphabet ‘∑’. It is assumed that the size of the alphabet is a constant, i.e., ‘n’<232. The latter implies that an integer in the range [0: n] can be stored in 4 bytes and also assumed that the special symbol ‘\$’ is an element of ‘∑’ (which is larger than all other elements) but does not occur in S. S[i] denotes the character at position ‘i’ in S, for $0\leq i< n$. For $i \leq j$, S [i...j] denotes the substring of S starting with the character at position ‘i’ and ending with the character at position ‘j’. The substring S [i...j] is also denoted by the pair of positions (i, j).

3.2. Suffix Array

Suffix array is designed for efficient searching of a large text. It requires only 4n bytes (4 bytes per input character) in its basic form. Searching a text can be performed by binary search using the suffix array. Suffix arrays will prove to be better than suffix trees for many genome applications [3]. The suffix array (denoted by Ssuftab) of the string S is an array of integers in the range ‘0’ to ‘n’, specifying the lexicographic ordering of the ‘n+1’ suffixes of the string S$ as shown in the column Ssuftab. That is, Ssuftab [0]; Ssuftab [1], …, Ssuftab[n] is the sequence of suffixes of S$ in ascending lexicographic order. For example the suffixes generated for the string S=acaatatacat$ is listed in Table 1.

Table 1. Suffixes of S=acaatatacat$

Index	$S_{suftab[i]}$
0	aatatacat$
1	acaatatacat$
2	acat$
3	at$
4	atcaat$
5	atatacat$
6	caatatacat$
7	cat$
8	tacat$
9	tatacat$
10	t$
11	$

The basic enhancement of the suffix array is the longest common prefix table denoted by Lcptab shown in Table 2. The Lcptab is an array of integers in the range 0 to n. lcptab[0]=0 and Lcptab[i] is the length of the Longest Common Prefix (LCP) of Ssuftab [i-1] and Ssuftab[i], for 1≤ i≤ n. Since, Ssuftab[n]=$, it always have Lcptab[n]= 0. The Lcp-table can be computed as a by-product during the construction of the suffix array. The Lcp-tab for the string S=acaatatacat $ is shown in Table 2. In this Table 2 Lcptab [0]=0 and Lcptab [1]=LCP of ‘aatatacat$’ and ‘acaatatacat$’=‘a’=1. Lcptab [2]=Lcp (‘acaatatacat$’ and ‘acat$’)=‘aca’=3.

Table 2. LCP table of S=acaatatacat$.

Index	Suftab	Lcptab	S(suftab[i])
0	2	0	aatatacat$
1	0	1	acaatatacat$
2	7	3	acat$
3	9	1	at$
4	5	2	atcaat$
5	3	3	atatacat$
6	1	0	caatatacat$
7	8	2	cat$
8	6	0	tacat$
9	4	2	tatacat$
10	10	1	t$
11	11	0	$

An interval [i...j], where 0≤ i≤ j - n, in an Lcp-array is called an Lcp-interval of Lcp-value ℓ (denoted by ℓ-[i...j]) if:

- Lcptab[i]< ℓ.
- Lcptab[k]≥ ℓ for all k with i+1≤ k≤ j .
- Lcptab[k]= ℓ for at least one k with i+1≤k≤ j .
- Lcptab [j+1]< ℓ.

Every index ‘k’, i+1≤ k≤ j, with Lcptab [k]=Ssuftab is called ‘ℓ’ index. The set of all ‘ℓ’ indices of an ‘ℓ’ interval [i...j] will be denoted by ‘ℓ’ Indices (i...j). If [i...j] is an ℓ-interval such that ω=S [suftab [i], ..., Suftab[i]+ℓ-1] is the LCP of the suffixes Ssuftab [i]; Ssuftab[i+1], … , Suftab[j], then [i...j] is also called ω - interval [3].

The parent-child relationship between the LCP-intervals generated in Table 2 constitutes a conceptual (or virtual) tree which is called LCP-interval tree of the suffix array. The root of this tree is the 0-interval [0-n] as shown in Figure 1 where all the LCP-intervals are plotted, along with arrows representing the parent-child relationship between them. Each individual suffix Ssuftab[l] of the suffix array can be considered as a singleton interval [l..l] and can be virtually placed as a leaf in the LCP-interval tree. The LCP-interval tree of S=acaaacatat $ is shown in Figure 1.

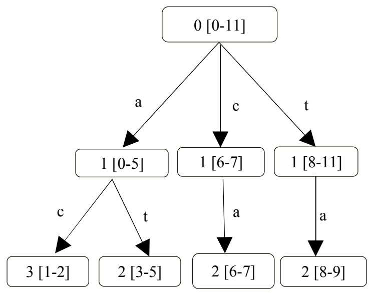

Figure 1. The Lcp-interval tree of S=acaaacatat$.

In Figure 1 Ssuftab[i], Ssuftab[i +1], ..., Ssuftab[j] of an ℓ-[i..j] share the prefix ω=S[suftab[i], ..., suftab[i]+ℓ-1] which is a branching character of Lcp-interval tree. As an example, consider the column Lcpinterval in Table 3. The interval [0-5] is a 1-interval because Lcptab[0]=0<1, lcptab[5+1]=0< 1, lcptab[k]≥1 for all k with 1≤ k≥ 5 and lcptab[3]= lcptab[10]=1. Furthermore, 1-[0-5] is the a-interval i.e.,) all the suffixes share the prefix ω=a.

Table 3. Reduced enhanced Lcp table of S=acaaacatat$.

Index	Suftab	Lcptab	S(suftab[i])	Lcpinterval
0	2	0	aatatacat$	[0-5]
1	0	1	acaatatacat$	[0-1]
2	7	3	acat$	[1-2]
3	9	1	at$	[3-5]
4	5	2	atcaat$	
5	3	3	atatacat$	
6	1	0	caatatacat$	[6-7]
7	8	2	cat$	
8	6	0	tacat$	[8-9]
9	4	2	tatacat$	[9-10]
10	10	1	t$	
11	11	0	$	

3.3. Enhanced Suffix Array

Enhanced suffix array, a new data structure is constructed with the help of suffix array, LCP-interval table and an additional table-child-Table [2, 3]. This enhancement is used to determine all child interval at constant time for any ℓ-interval [i...j]. The child-table is a table of size n+1 indexed from 0 to n and each entry contains three values: up, down and next ℓ

Index. Space complexity to construct enhanced suffix array with Ssuftab, Lcptab and child table is 6n bytes for each entry [2]. For a given pattern 'p' in the string 'S' the time complexity of enhanced suffix array to find all the occurrences of the pattern is O(m+z) where 'z' is number of occurrences of 'p' in S and 'm' is the length of the pattern [2].

In the developed BioDBMPHF tool, the enhanced suffix array is constructed using MPHF with the help of bucket table instead of using child table. This method reduces the time complexity by O(2+γ) where 'γ' is the number of occurrences. Space complexity of enhanced suffix array is 5n bytes + 32 bytes.

3.4. Accessing the LCP Table using MPHF

It is possible to achieve the time complexity by O(log|∑|) time by reorganizing the child-table [10]. Fischer and Heun [8] achieved the same complexity but based on a data structure supporting range minimum queries. Abouelhoda and Dawood [1] solves exact pattern matching using fine-tuned enhanced suffix array in O(m) time and O(n) space where 'm' is the size of the pattern and 'n' is the size of the string. In this paper enhanced suffix array is fine-tuned using developed new hash index algorithm. Time complexity to find a given pattern is O(2+γ) access time where 'γ' is the time taken to access the pattern within the lcp-interval.

MPHF are widely used for memory efficient storage and fast retrieval of items from static sets such as words in natural languages, reserved words in programming languages or interactive systems, Universal Resource Locations (URLs) in web search engines or item sets in data mining techniques. There are applications for minimal perfect hash functions in information retrieval systems, database systems, language translation systems, electronic commerce systems, compilers, operating systems, biological data and the enhanced suffix array among others.

Though DNA consists of long sequence of string, it consists of only four letters A, G, C, and T. If these letters are arranged as a two letter combination there are only 16 different combinations for the substring Ssuftab[i] which forms the enhanced suffix array. The idea of the developed algorithm is to store for each LCP-interval a perfect hashing data structure containing the list of branching characters suf-start and the respective LCP-interval index. hash index algorithm as shown in Algorithm 1 is used to find a hash value which is attached to each LCP-intervals.

ASCII value of first character of S (suftab[i]) is shifted right and moved to ShrFtr and the ASCI value of the second character of S (suftab[i]) is shifted right and moved to shrsnd. Then, hash index is calculated using HashInd function as shown if Algorithm 1. Bucket table is formed with the help of this hash index as shown in the Table 4. Because of DNA two letter combinations consists of only 16 rows, each entry in bucket table requires 2 bytes. So, space complexity to store bucket table is only 32 bytes using developed hash index algorithm.

Algorithm 1: Hash index algorithm

```
Input: C1 and C2 ← first and second of S (suf [i])
Output: Hash index for S (suf [i])

Function HashIndex (C1, C2)
      {
        Ftr ← ASCI (C1);
        ShrFtr  ←shr (Ftr);
        Snd ← ASCI (C2);
        Shrsnd ←shr(Snd);
        HashInd=(Ftr % 97) + (Snd % 97);
                {
                  if(Ftr>Snd)
                    {
                      HashInd=(HashInd+ShrFtr+ Shrsnd)
                      % 98;
                    }
                  HashInd = HashInd % 16;
                }
      }
```

Table 4. Bucket Table.

Suf-start	
aa	
Ac	
At	
Ag	
Ca	
Ct	
Cc	
Cg	
Ta	
Tc	
Tg	
Tt	
Ga	
Gt	
Gc	
Gg	

Index	Suf Tab	Lcp tab	S(suftab[i])	Lcpinterval
0	2	0	aatatacat$	[0-5]
1	0	1	acaatatacat$	[0-1]
2	7	3	acat$	[1-2]
3	9	1	at$	[3-5]
4	5	2	atcaat$	
5	3	3	atatacat$	
6	1	0	caatatacat$	[6-7]
7	8	2	cat$	
8	10	0	tacat$	[8-11]
9	6	2	tatacat$	[9-10]
10	4	1	t$	
11	11	0	$	

Given a pattern 'p' of length 'm', one traverses the bucket table over the fine-tuned enhanced suffix array. The crucial part of this algorithm is to locate a pattern with a branching characters AA, AC, AG, AT=P [x, y] for some x, y Є [0, ..., m-1]. P [0, 1]=S (suftab[i]). First two character of the pattern such that P [0, 1]=S [suftab[i]] is identified. During the next access S (suftab[i]) is searched against the next alphabetical combination because S(suftab[i]) is arranged in alphabetical order. For example, if first two characters from the pattern are AA then second access in the S (suftab[i]) is searched against either AC or AG or AT. Interval between the first and second access is called child-interval. So, the pattern exists only in this child-interval. If the child-interval is empty then the given pattern does not exist.

In this paper, enhanced suffix array is fine-tuned using new hash index algorithm as shown in Algorithm 1. Time complexity of the above algorithm is O(2+γ) where 'γ' is the time taken to access the pattern within the child-interval. Each entry to space complexity of fine-tuned enhanced suffix array is 5n bytes, for reduced enhanced Lcptab and to store bucket table it requires 32 bytes.

4. Bio Database Compression

The fine-tuned enhanced suffix array is generated for all the sequences oryza sativa DNA sequences and is named as GFTES Array. The substring of a string S in GFTESArray is used to identify similarity between homologous sequences, because similar sequences contain conserved regions. Significantly similar sequences are identified with the help of a score. A given pattern with high score carries important information that belongs to a family of sequence with a highly conserved region. It is calculated based on the length of the pattern and frequency i.e., number of occurrences in the database. The given pattern p is classified as significant if it satisfies the following constraints:

- The length of $p \geq a$ given length-threshold: A significant pattern must be sufficiently long to carry important biological information.
- The score of $p \geq a$ given score-threshold: A Significant pattern must have a sufficiently high score.

GFTESArray is constructed for the entire sequences in the database. While constructing GFTESArray, at each node i, the length $L(P_i)$ and the frequency $F(P_i)$ (i.e., the number of occurrences of p in the database) is stored for the corresponding pattern p_i this frequency is incremented for every new node. Then the score function $W(P_i)$ is calculated using the Equation 1:

$$W(P_i) = \sqrt{F(P_i) * L(P_i)} \Big/ |DB| \qquad (1)$$

Where:

- $W(P_i)$=Function value of the pattern.
- $F(P_i)$=Frequency of the node.
- $L(P_i)a$=Length of the label of prefix.
- i=Node number.
- $|DB|$=Database size.

The score function $W(P_i)$ is used as a measure to determine whether a particular pattern is significant to be included in the post-processing for evaluation of sequences, to determine the closest set of sequences. Then, the query sequence and number of sequences to be selected from the database is read from the user temporarily added onto GFTESArray.

This enables to determine which suffixes of the query are shared by the sequence in the database. The query sequence is only temporarily added to the tree so, that BioDBMPHF compression tool is not affected for future sequence searches. Initially all the nodes in the GFTESArray are 0. When the query sequence is added as a suffix, the nodes visited are set to 1. This expedites the search for common patterns within the GFTESArray because only those paths in the tree for patterns that contain substrings of the query sequence are examined. In depth-first manner, starting at the root all the nodes is visited to check the value 1. If the current node has no child whose value is 1, then the search backtrack to its parent node. During this traversal all the significant patterns are collected. The sequence may have other common patterns that are not significant. An optimal alignment between these two sequences in an ideal case contains all significant patterns. After this process the query sequence from GFTESArray is deleted. Top ten significant patterns are selected and stored. The sequence that contains significant patterns is extracted and stored. Reverse check is made to obtain the accuracy of the results; it computes how many chosen patterns are being shared by each of the sequences extracted already. Higher the number (weight), greater will be the similarity of the corresponding sequence to the query. Based on this weight, the sequences are ranked. A top ‘n’ sequences are transferred to new database. Algorithm for the developed BioDBMPHF compression tool is shown in Algorithm 2.

Algorithm 2: BioDBMPHF compression tool

```
// Input:  set of DNA Sequences
// Output:  Compressed set of DNA sequences

S 1: Read DNA database.
S 2: Construct GFTESArray for the input sequences, Set node visit=0.
S 3: While constructing the suffix array, store the information of label-length, frequency at nodesand sequences traversing through the branches.
S 4: Use the Equation 1 to calculate the degree of similarity of patterns in the form of prefixes.
S 5: Read query sequence Q and the number of sequences n to select from the user.
S 6:Temporarily add the suffixes of query to the generalized GFTESArray.
S 7: While adding the suffixes highlight nodes of the paths which are traversed by the query sequence.
S 8:Post process the GFTESArray to extract patterns shared by the query sequence, which lies above a defined threshold on the function-value.
S 9:Pick the top ten of these patterns and store.
S 10: Do a reverse check to compute the weight of each sequence in the subset.
S 11: Rank the sequences according to these weights.
S 12: Pick top n sequences from the subset and write to a new database.
```

5. Results and Conclusions

Cross-validation, is the statistical practice of partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while other subset(s) are retained for subsequent use in testing and validating the initial analysis. In this paper. 7-foldcross validation is used to validate the consistency of the result.

5.1. Cross Validation

The real world DNA (GSS, EST) databases from oryza sativa group is extracted from NCBI website and enhanced suffix array is formed for all the sequences in the database. Based on the user given query sequence, the significant patterns are generated. The sequences in the DNA database are given weights according to the number of patterns they contain. The compressed database is formed by selecting top ‘n’ sequences with highest ranks and written into a new database.

The main idea of 7 fold cross validation approach is to “train on 6 folds and test on 1 fold’. The data set is divided into 7 parts. Among the 7 parts 6/7 of the data are used for training data set and the remaining 1/7 is used for testing data set. BioDBMPHF compression tool is applied on training data set and then on WU_BLAST for single user given query sequence. For the same query sequence WU-BLAST alone is applied on the testing data set. The average of this seven runs is computed for analysis. In this paper such seven queries are taken and analyzed. 49 sequences from oryza sativa GSS gi: 288881557 to gi: 288881606 are taken as a database set, 42 sequences are used for training data set and 7 sequences are used for test data set. Single query sequence is applied first on BioDBMPHF compression tool, data base is compressed. WU-BLAST is performed on the new data base and the same query sequence. Then, for same sequence WU-BLAST alone is applied. The sequences in the training and test data set are interchanged and the above steps are repeated until every fold is used for training. The average result from 7 runs are calculated and stored. This is experimented for 7 different queries. The objective is to test whether the results are consistent for all the queries on a particular database in terms of computational time.

Reduced database from the developed BioDBMPHF compression tool is cross validated using 7 fold cross validation approach. The cross validation result of seven different queries for DNA database set GSS gi: 288881557 to gi: 288881606 are shown in Figure 2 and EST gi: 288886142 to gi: 288886191 are shown in Figure 3. Running time of developed BioDBMPHF Compression Tool is measured on a PC equipped with a 3.0 GHz Core 2 Duo 32 bit Processor and 3 GB DDR2 RAM. The PC is operated by windows 7.

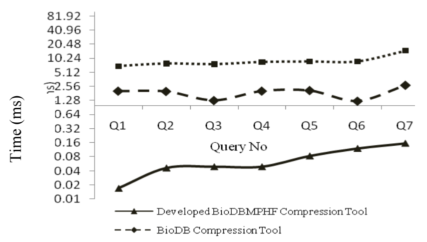

Figure 2. Computation time for BioDB compression tool, BioDBMPHF compression tool and WU-BLAST for the data set GSS gi: 288881557 to gi: 288881606.

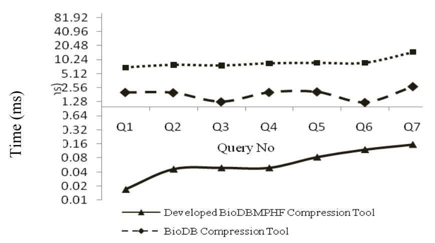

Figure 3. Computation time for BioDB compression tool, BioDBMPHF compression tool and WU-BLAST for the data set EST gi: 288886142 to gi: 288886191.

The idea is to test whether the results are consistent for all queries on a particular database in terms of computation time. First series in the Figures 4 and 5 represents the computation time obtained from WU-BLAST, second series represents the result of BioDB compression tool [11] and the third one represents the result of applying developed BioDBMPHF compression tool. Suffix array enhanced with child table is used in BioDB compression tool. The Figure shows that the results obtained from BioDB compression tool [11] and developed BioDBMPHF compression tool are consistent and perform sequence comparison with a good accuracy and a practical time improvement is achieved over WU-BLAST.

Existing BioDB compression tool [11] is constructed with the help of suffix array enhanced with child table. The space complexity of enhanced suffix array is 6n bytes to store Ssuftab, Lcp table and child table for single entry [2, 9]. The time complexity of enhanced suffix array for a given pattern ‘p’ of the string ‘S’ is O(m+z) where ‘z’ is number of occurrences of ‘p’ in S and ‘m’ is the length of the pattern.

The developed BioDBMPHF compression tool is implemented using fine-tuned enhanced suffix array using new hash index algorithm. Time complexity of fine-tuned enhanced suffix array is O(2+γ) where ‘γ’ is the Time taken to access the pattern within the child-interval. Space complexity of fine-tuned enhanced suffix array is 5n bytes for reduced enhanced Lcptab and to store bucket table it requires 32 bytes.

In this paper, the BioDBMPHF compression tool using improved fine-tuned enhanced suffix array has been developed. BioDBMPHF compression tool pre-processes the database to create a generalized fine-tuned enhanced suffix array and extended by adding frequency and length information for the patterns. BioDBMPHF compression tool distinguishes patterns by computing significance-scores. A pattern is regarded as important if it is lengthy enough and it appears frequently enough in the database. The scoring function takes into account a pattern’s length and frequency, the given threshold values and determines if a pattern is important. Using these, for a given query sequence BioDBMPHF compression tool compresses the database to only a few sequences that share the most significant patterns with the query. This compression in database size speeds-up the local alignment of the query sequence against the database. Experimental results have shown that BioDBMPHF compression tool provides a speed-up over WU-BLAST. It is able to reduce the time of a database search to nearly five times originally taken by WU-BLAST. Results from WU-BLAST have shown that this method is experimentally effective, as the results obtained by BioDBMPHF compression tool produces accurate alignment. Combined with the extended suffix array, BioDBMPHF compression tool has the advantage of using WU-BLAST to do the local sequence alignment. Latest data mining technique, 7 fold cross validation is applied to attain a greater accuracy in the results and the results are satisfactory. The fine-tuned Enhanced suffix array algorithm used in BioDBMPHF compression tool requires 5n bytes + 32 bytes/entry where SCT requires 20n bytes which uses suffix tree and BioDB compression tool requires 6n bytes/entry. So, space complexity is approximately five times less than SCT and also less than BioDB compression tool. Experimental results show that the running time of BioDBMPHF compression tool is much better than SCT and BioDB compression tool.

In this paper, a small domain of sequences have been selected from the DNA database and experimentally proved that, fine-tuned enhanced suffix array reduces space complexity by five times and time complexity is also reduced with the help of developed hash index algorithm. BioDBMPHF compression tool can also be applied to global sequence alignment and multiple sequence alignment. This work can also be applied to protein sequences by modifying the hash index algorithm.

References

[1] Abouelhoda M. and Dawood A., "Fine Tuning the Enhanced Suffix Array," *in Proceedings of the 4th Cairo International Biomedical Engineering Conference*, *IEEE Explore*, Cairo, Eygpt, pp. 1-4, 2008.

[2] Abouelhoda M., Kurtz S., and Ohlebusch E., "Replacing Suffix Trees with Enhanced Suffix Arrays," available at: http:// www.vmatch.de/ AboKurOhl2004.pdf, last visited 2004.

[3] Abouelhoda M., Stefan K., and Ohlebusch E., "The Enhanced Suffix Array and its Application to Genome Analysis," *in Proceedings of the 2nd Workshop on Algorithms in Bioinformatics*, Rome, Italy, pp. 1-15, 2004.

[4] Altschul S., "Gap Costs for Multiple Sequence Alignment," *the Journal of Theoretical Biology*, vol. 138, no. 3, pp. 297-309, 1989.

[5] Altschul S., Gish W., Miller W., and Lipman J., "Basic Local Alignment Search Tool," *the Journal of Molecular Biology*, vol. 215, no. 3, pp. 403-410, 1990.

[6] Altschul S., Madden T., Schäffer A., Zhang J., Zhang Z., Miller W., and Lipman D., "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389-3402, 1997.

[7] Divya R., Abdullah N., and Xindong W., "Using an Extended Suffix Tree to Speed-Up Sequence Alignment," *in Proceedings of IADIS International Conference Applied Computing*, Salamanca, Spain, pp. 655-660, 2006.

[8] Fischer J. and Heun V., "A New Succinct Representation of RMQ Information and Improvements in the Enhanced Suffix Array," *in Proceedings of the International Symposium on Combinatorics*, *Algorithms*, *Probabilistic and Experimental Methodologies*, Hangzhou, China, pp. 459-470, 2007.

[9] Mousa H., Moustafa K., Abdel-Wahed W., and Hadhoud M., "Data Hiding Based on Contrast Mapping using DNA Medium," *the International Arab Journal of Information Technology*, vol. 8, no. 2, pp.147- 154, 2011.

[10] Kim D., Kim M., and Park H., "Linearized Suffix Tree: An Efficient Index Data Structure with the Capabilities of Suffix Trees and Suffix Arrays," *Algorithmica*, vol. 52, no. 3, pp. 350-377, 2008.

[11] Kunthavai A. and Vasantharathna S., "Bio-Database Compression using Enhanced Suffix Array for Pair wise Sequence Alignment," *the Journal of Cell and Molecular Biology*, vol. 9, no. 1, pp. 45-52, 2011.

[12] Manber U. and Myers E., "Suffix Arrays: A New Method for On-Line String Searches," *the SIAM Journal on Computing*, vol. 22, no. 5, pp. 935-948, 1993.

[13] Needleman S. and Wunsch C., "A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins," available at: http://csb.stanford.edu/class/ public/readings/Bioinformatics_I_Lecture6/Needleman_Wunsch_JMB_70_Global_alignment.pdf, last visited 2012.

[14] Pearson W. and Lipman D., "Improved Tools for Biological Sequence Comparison," *the Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, no. 8, pp. 2444-2448, 1988.

[15] Pearson W., "Effective Protein Sequence Comparison," *Methods in Enzymology*, Elsevier, 2000.

[16] Pearson W., "Flexible Sequence Similarity Searching with the FASTA3 ProProgram Package," *Methods in Molecular Biology*, Springer, 2000.

[17] Smith T. and Waterman M., "Identification of Common Molecular Subsequences," available at: http://csb.stanford.edu/class/public/readings/Bioinformatics_I_Lecture6/Smith_Waterman_JMB_81_Local_sequence_alignment.pdf, last visited 1991.

Arumugam Kunthavai received BE degree in computer science and engineering from Madurai Kamarajar University, MS degree (by research) in 2008 and PhD degree in 2014 from Anna University, India. Currently, she is working as an Assistant Professor in computer science and engineering at Coimbatore Institute of Technology, India. She is life member of ISTE. Her research interest includes bioinformatics, distributed system and data mining.

Somasundaram Vasantharathna received BE degree in electrical engineering, ME degree in applied electronics and phD degree in electrical engineering all from Bharathiar University, Tamilnadu, India. Currently, she is working as a Professor and Head of electrical and electronics engineering at Coimbatore Institute of Technology, India. She is life member of ISTE. Her research interest includes operating systems, power quality and smart energy systems and image processing applications to food adulteration.

Swaminathan Thirumurugan received BE degree in computer science and engineering from Coimbatore Institute of Technology, India in 2012. Currently, he is working atInformatica. His research interest includes algorithm analysis and data mining.