BLOSUM Trie for Faster Hit Detection in FSA Protein BLAST

M Anuradha
Research scholar
Department of Computer Science & Systems Engineering, Andhra University
Visakhapatnam - 530 003

K Suman Nelson
Software Engineer
Google India
Hyderabad

P V G D Prasad Reddy
Professor
Department of Computer Science & Systems Engineering, Andhra University
Visakhapatnam - 530 003

ABSTRACT
Basic Local Alignment Search Tool (BLAST) is one of the most widely used bioinformatics tools to determine similarities between genomic sequences. Ever since its inception several algorithmic improvements have been made to improve speed and runtime memory requirements without affecting the sensitivity and selectivity of the tool. Fast search algorithm (FSA) BLAST has been the most successful among such improvements with 20-30% faster processing rate. A DFA with hashed prefix word structures for the hit detection process in FSA BLAST has been proposed in the earlier work. Coding of the algorithms and testing on protein samples showed that the use of the new structure resulted in significant reduction in run time space but not the hit detection time. This paper proposes the use of a BLOSUM trie structure which eliminates the process of computing neighborhood words, resulted in a reduction of up to 75% in hit detection time. Tests were conducted with different BLOSUM matrices and threshold values and the proposed algorithm was found to be beneficial in terms of space, time complexity and accuracy without compromising on sensitivity and selectivity of the currently being used algorithm.

General Terms
Sequence Analysis, Protein BLAST, Hit Detection Algorithm, Time Complexity, Space Complexity, Sensitivity and Selectivity.

Keywords
Deterministic finite automaton, prefix word table neighborhood words, query pointers, hits and BLOSUM Trie.

1. INTRODUCTION
Basic local alignment search tool (BLAST) is the most widely used sequence similarity search tool used by computational biologists to understand the role, structure and function of genomic sequences. BLAST performs comparisons between a pair of sequences in order to find regions of local similarity [1]. The popular BLAST derivatives are NCBI-BLAST (web based and standalone versions are available), WU-BLAST, Paracel BLAST and fast search algorithm (FSA)-BLAST [2, 3, 4, 5, 6, 7]. Among them, NCBI-BLAST (standalone) and FSA-BLAST are open source programs and any one can download (ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/Latest, http://www.fsa-blast.org) and experiment with the program code and algorithms.

Because of its widespread usage (used over 120,000 times each day) and exponential growth in genomic databases, BLAST is becoming slower [8, 9]. In view of the above reasons, any improvement to the BLAST algorithm that can reduce runtime space and time without effecting sensitivity and selectivity would be very much desirable [10].

Over the years several modifications to the fundamental algorithms and new heuristics in BLAST were proposed to improve speed and minimize runtime space [3-7], [11-18]. Basically, BLAST program was designed to analyze both protein and DNA sequences. It has mainly four algorithmic steps namely finding hits, performing un-gapped alignments, performing gapped alignments and computing trace back and outputting the results [2, 3, 6, 7]. The main functional differences between NCBI BLAST and FSA BLAST are, one is the structure used for finding hits between a query sequence and database sequence during the hit detection process and the other is using semi-gapped and restricted insertion alignments during alignment stage of the algorithm [6,7]. A modified DFA proposed in the earlier work, reduced the runtime space significantly during the hit detection stage of the FSA protein BLAST algorithm [19]. But reduction in hit detection time is not significant. This paper proposes a method that eliminates the process of computing neighborhood words during hit detection process. It resulted in significant reduction in hit detection time of FSA protein BLAST algorithm.

Hits are short, fixed length high scoring matches between query sequence and database sequence. For protein search, hit is a match of word length 3 and inexact matches are permitted whereas for nucleotide search, hit is an exact match of word length 11. For finding hits, FSA-BLAST used an optimized deterministic finite automaton (DFA) which reduced the total BLAST search time by 6 to 30% compared with table look-up used in NCBI BLAST[3,7]. But study on this implementation revealed that, the number of query pointers used by the structure is fixed and dependent on alphabet size (a) and word length (w), and is equal to 'a^w '(for protein sequence, a=24 and w=3), i.e., 24^3=13,824, not on the length of the query. In the modified structure, it is made dependent on length of query sequence and the number of neighborhood words to each query word. This reduces the run time space of algorithm by a considerable amount, by initializing only the necessary query pointers [19].

This paper is organized into seven sections. The hit detection process used in FSA protein BLAST is described in section 2 using a small sample query as an example. Description of DFA with hashed prefix word structures and trie used in this study are given in section 3. In Section 4, implementation details and testing of code are explained.

Results are analyzed and discussed in Section 5. Conclusions are given in section 6 and acknowledgements in section 7.

2. HIT DETECTION PROCESS

In this stage, BLAST compares query sequence with each sequence of the database using Wilbur and Lipmann algorithm [20]. This process is done in two steps; one is structure construction which is unique for a query, and second is processing the database sequence to find word matches between query sequence and database sequence.

During the structure construction, fixed length overlapping words of length 'w' are extracted from the query sequence. For example, let 'BABBC' be the query sequence made up of an alphabet: {A, B, C} and w=3, then the fixed length overlapped words extracted from the query are: BAB, ABB, and BBC. For each query word, neighborhood words of length 'w' are generated. A neighborhood word is a word obtaining a score greater than or equal to some threshold value 'T' (default values used by NCBI-BLAST for protein search are T=11 and w=3) [21-24], using a selected scoring matrix. For example for query word BAB, BAC is neighborhood word. When B is matched with B, score is 6, when A is matched with A, score is 5 and when C is matched with B, score is 0. Then the word score is sum of the individual scores and given below.

B A B

| | |

B A C 6+5+0=11 which is equal to T.

The default scoring matrix used for protein BLAST is BLOSUM62 [25]. Each query word along with its neighborhood words will be associated with a query position and are stored in a structure. In the above example, query positions of query words BAB, ABB, and BBC are 1, 2, and 3 respectively.

While processing the database sequences, each sequence is processed sequentially, that is each sequence is read from the database, parsed into words of length 'w' and searched for

query word match in the structure. If a match is found, corresponding query word position 'i' and database sequence word position 'j' are recorded as hit, which will be the input to the alignment stage of the BLAST algorithm.

3. STRUCTURES USED FOR FINDING HITS

3.1 Existing FSA-DFA Structure

FSA-DFA consists of states, and transitions between the states. A state is a prefix word of length (w-2). The total number of states is equal to $a^{(w-2)}$ (24 for a protein sequence). FSA-DFA shown in Figure.2 for an input given in Figure.1 is constructed as follows.

Position:	1	2	3	4	5
Query:	B	A	B	B	C
Subject:	C	B	A	B	B
Threshold:	11				

(a)

	A	B	C
A	5	-1	2
B	-1	6	0
C	2	0	4

(b)

QP	QW	NHWs
1	BAB	BAC, BBB, BCB, CAB
2	ABB	ABC, ACB,BBB, CBB
3	BBC	BBA, BBB

(c)

Figure 1: (a) Example Query and Database sequences constructed with an alphabet:{A,B,C};word length(w)=3 (b) Scoring Matrix (c) List of query words(QW) and their neighborhood words (NHWs) along with their query positions (QP) for the given query

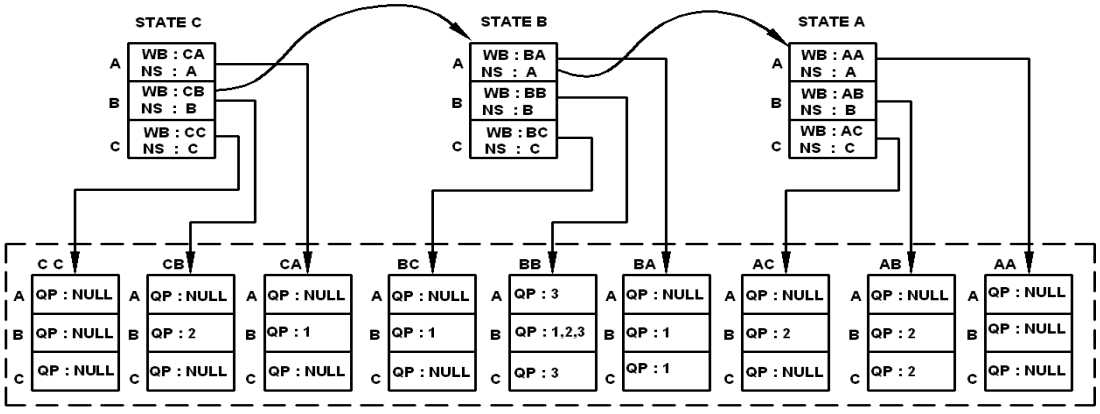


Figure 2: Existing FSA-DFA constructed for the input given in Figure 1(a) and (b)

- $a^{(w-2)}$ states are initialized in such a way that each state consists of 'a' transitions and each transition is associated with two pointers, one to the next state and other to a collection of words that share common prefix of length '(w-1)'. Each word in the prefix table is associated with a query pointer pointing to a list of query positions which is initially initialized to NULL.
- For each query word, neighborhood words given in Figure 1(c) are computed based on the given threshold T using alignment scoring matrix, and their query positions are stored in the structure.

For the given example, structure consists of 3 states A, B and C. Each state consists of 3 transitions A, B and C, and each transition is associated with corresponding prefix table. For a given query, we start with word BAB. Transition A of state B has pointer to next state A and pointer to prefix word table BA. In the prefix word table, for the word BAB, the query position is marked as 1 because it is available in query sequence at position 1. The neighborhood words to BAB, listed in fig 1(c), are now computed and their positions are also stored in the structure. This process repeats for every query word.

While processing the database sequence 'CBABB', the first character 'C' is read. That is we are starting with the state C. Then the next character 'B' is read, transition 'B' from state C is followed. Here transition B of state C has, one pointer to next state B and other pointer to prefix table of word CB. Since the next character read is A, and for the word CBA there is no match in the query sequence the search advances to next state B. Now we are at state B, and next character read is A, then transition A from state B is followed which has pointers to next state A and prefix word table BA. When the next character B is read, transition 'B' from prefix table 'BA' is followed. Here the transition has a query position 1, hence outputs the hit as (1, 2), i.e., match occurs at query position '1' and database sequence position '2' and search advances to next state 'A'. This process continues until the data base sequence is exhausted.

3.2 FSA-DFA with hashed prefix word structures

The limitation of currently being used FSA-DFA is, whatever may be the query length, size of the DFA and the number of query pointers initialized during the structure construction stage is fixed (a^w). For a protein sequence, it is equal to 24^3=13,824. Indeed, for a given query we may not be using these many pointers. Whether we use or not, 4 bytes of memory is allocated to each pointer which will be a considerable overhead on run-time space utilization. It can be overcome by using an array of states and prefix word hash table which makes the number of query pointers to be initialized dependent on the size of query and the number of neighborhood words each query word is associated with, instead of initializing fixed number of pointers. This section presents construction of such structure with an example and is explained below.

It consists of array of states, where each state is a word block of (w-2) length. The size of the array is equal to $a^{(w-2)}$ (24 for a protein sequence). Each state is associated with a hashed prefix word table with query or neighborhood word as key and pointer to list of query positions as value. Initially the size of each hash table is zero and it grows by one key and one value for each query word and its neighborhood words, while constructing FSA-DFA.

A portion of the modified FSA-DFA shown in Figure.3, for the given query sequence 'BABBC' is constructed as follows.

Let the query word be 'w1w2w3'. For each query word, w1 is the current state, w2 is the next state and w3 is the transition that maps query word w1w2w3 from current state into corresponding query position in prefix word hash table w1w2. In this example, for query word 'BAB', B is the current state, A is next state and B is the transition that maps query word BAB into query position 1 in prefix word hash table BA. Similarly neighborhood words to word BAB are also mapped into query position 1. Similarly the remaining query words, ABB and BBC, and their neighborhood words are also mapped into their corresponding query positions in corresponding prefix word hash tables. So whenever a query word or neighborhood word is mapped into corresponding query position, two pointers are initialized. One maps into the hash table of that state and other to the next state. Hence the number of query pointers initialized is less than or equal to the sum of the number of query words and their neighborhood words.

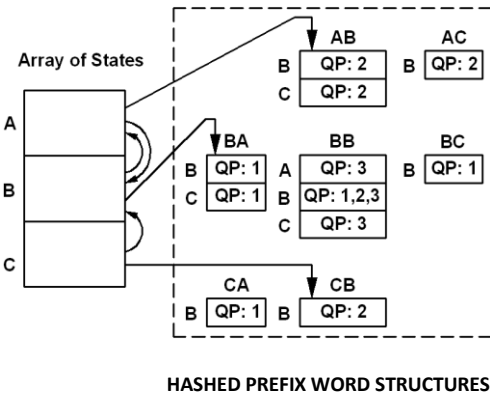


Figure 3: FSA-DFA with hashed prefix word structures constructed for the input given in Figure 1 (a) and (b)

The database sequence 'CBABB' is processed as follows. When the first character 'C' is read, we are starting with state C. Then the next character 'B' is read, a pointer to next state B is followed. When next character 'A' is read, since there is no match for the word CBA in the query, search advances to next state B. At state B, a pointer to next state A and other to the prefix word hash table BA of state B is followed. When the next character B is read, word BAB will be hashed into slot B of prefix word hash table 'BA' of state B and outputs a hit (1, 2). Now we are at state A. when character B is read a pointer to next state B and other to the prefix word hash table AB of state A is followed. When the next Character B is read, word ABB is hashed into slot B of prefix word hash table AB of state A (where match is found for the word ABB in the query sequence at position 2, that outputs a hit (2, 3)). The process continues until the data base sequence is exhausted.

3.3 BLOSUM trie structure

Finding neighborhood words to each query word during the structure initialization stage is most time consuming because each word needs to be compared with a^w words where each

query word will be having very few neighborhood words out of a^w words and the process repeats for every query. Cost of finding neighborhood words can be reduced by using a BLOSUM trie structure consisting of pre-computed lists of neighborhood words to each possible word of length 'w' for a given alphabet and scoring matrix. This section presents construction of such structures with an example.

Let 'Q' be set of query words where q_i is query word available at i^{th} position of query sequence, 'Q_i' be the set of neighborhood words to query word q_i and 'S' be the set of all possible words of fixed length w with the given alphabet of size 'a'. Let $n(Q_i)$ and n(S) are number of elements in the sets Q_i and S respectively where $n(S)=a^w$. For any query, Q is a proper subset of S and $n(Q_i)$ is very small when compared with n(S). For a specific scoring matrix if we compute the set of neighborhood words to each word in S once, they can be reused for all query words and any number of queries. By taking the above facts into consideration a BLOSUM trie is constructed as follows.

• For the scoring matrix given in Figure 1(b), alphabet: {A, B, C} and word length w=2, the word matrix 'W' of size (a^w x a^w) (9x9 in this example) is computed in such a way that the element w_{ij} of W is the score when the word in i^{th} row is matched with the word in j^{th} column.

• The lists of neighborhood words (words with $w_{ij} \geq T$) to each word in S are extracted from W and are arranged in descending order as shown in the Figure 4(a) and (b).

	AA	AB	AC	BA	BB	BC	CA	CB	CC
AA	10	4	7	4	-2	1	7	1	4
AB	4	11	5	-1	5	-1	1	8	2
AC	7	5	9	1	-1	3	4	2	6
BA	4	-1	1	11	5	8	5	-1	2
BB	-2	5	-1	5	12	6	-1	6	0
BC	1	-1	3	8	6	10	2	0	4
CA	7	1	4	5	-1	2	9	3	6
CB	1	8	2	-1	6	0	3	10	4
CC	4	2	6	2	0	4	6	4	8

(a)

AA	AA, AC, CA
AB	AB, CB
AC	AC, AA
BA	BA, BC
BB	BB
BC	BC, BA
CA	CA, AA
CB	CB, AB
CC	CC

(b)

Figure 4: (a) Pre-computed word matrix consisting of scores of all possible words of size w=2, obtained using alphabet: {A, B, C} and the scoring matrix given in Fig. 1 (b) Lists of neighborhood words to each possible word of size w=2 and word score ≥7 obtained from pre-computed word matrix

The lists of neighborhood words to each word in S are stored as a trie structure because trie is more space efficient, particularly when there is large number of known short length keys. Another advantage of trie structure is, searching and data retrieval is faster (of order of O(w)). Trie structure for the lists given in Figure 3(b) is shown in Figure 4. It is implemented as non-binary search tree, where each node contains at most 'a' pointers, corresponding to 'a' possible characters in each position of the word except leaf nodes. The root node corresponds to an empty string whereas the other nodes correspond to prefixes of words in S. Each path from root to a leaf corresponds to one word in S.

• The neighborhood words to each word in S are extracted from the trie by traversing from root node with the prefix of word, to a node where the word ends. All the leaf nodes under that node are nothing but the set of neighborhood words including the word itself.

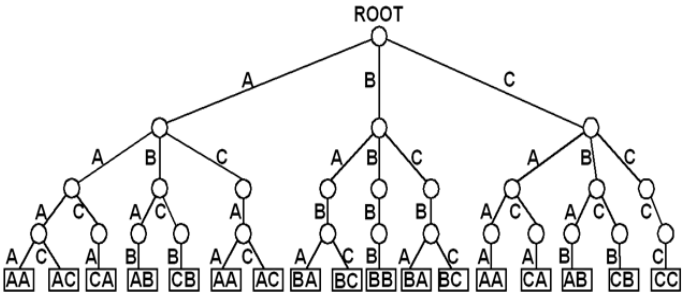


Figure 5: A Trie structure consisting of lists of neighborhood words represented in Figure 4(b)

4. TESTING

Proposed hit detection algorithm has been developed using C++ in visual studio 2010 environment. To evaluate the efficiency of proposed algorithm, hit detection process from FSA-BLAST has been taken as reference. To test the code, input and output files were created. The input file consisted of twenty five protein sequences, randomly extracted from the non-redundant protein database (a collection of sequences from several sources, including translations from annotated coding regions in GenBank, RefSeq and TPA, as well as records from SwissProt, PIR, PRF, and PDB) using Entrez, listed in Appendex A; BLOSUM 45, 62 and 80 scoring matrices and their corresponding pre-computed word matrices. The size of the proteins so selected varied from 110 to 6907 amino acids in order to study the effect of query length on the performance of hit detection algorithm. Here, the set of 25 protein sequences extracted are considered as database and each one of the 25, taken one at a time as a query to be searched against the database. Threshold value needed to find neighborhood words for each query word is taken as 9, 11 and 13. Trie is constructed from the pre-computed word matrix during structure initialization stage as explained in section 3.3. When each query is run on the database, the program outputs, pairs (i, j), that identify matches between the query and database sequences. The number of hits and the number of query pointers initialized for the structure are stored into the output file. This process is repeated for both the algorithms, existing FSA DFA and hashed FSA DFA with BLOSUM 45, BLOSUM 62 and BLOSUM 80 scoring matrices and with trie structure with threshold values 9, 11 and 13. Time taken to initialize the FSA DFA structure and processing the database sequence has been measured separately using a utility program known as performance counter. Quantitative and qualitative analysis on

Table 1. Comparison of hit detection time and number of query pointers used in FSA-DFA using trie and hashed FSA-DFA using trie with FSA-DFA for BLOSUM 62 scoring matrix and T=11

S.No.	*Query protein sequence*	*Gene – id*	*Sequence Length (aa)*	*Hit Detection Time (ms)*			*Number of Query Pointers used*	
				FSA-DFA	*FSA-DFA with Trie*	*Hashed FSA-DFA with Trie*	*without hash*	*with hash*
1.	B Chain B, Structure Of P. Citrinum	gi|20151022|pdb|1KRF|	511	49.89	4.75	16.78	13824	7338
2.	2-oxoglutarate dehydrogenase-like, mitochondrial isoform a [Homo sapiens]	gi|221316661|ref|NP_060715.2|	1010	101.64	9.54	27.21	13824	10393
3.	myosin [Arabidopsis thaliana]	gi|433663|emb|CAA82234.1|	1520	149.96	11.4	36.51	13824	11436
4.	chromodomain-helicase-DNA-binding protein 3 isoform 1 [Homo sapiens]	gi|52630326|ref|NP_001005273.1|	2000	192.21	19.89	47.9	13824	11557
5.	NOTC2_MOUSE RecName: Full=Neurogenic locus notch homolog protein 2;	gi|20138876|sp|O35516.1|	2470	254.64	30.93	73.84	13824	11076
6.	novel protein similar to vertebrate low density lipoprotein-related protein 2 (LRP2) [Danio rerio]	gi|169158323|emb|CAQ13433.1|	3091	309.36	37.18	93.42	13824	11865
7.	CUB and sushi domain-containing protein 3 isoform 1 [Homo sapiens]	gi|38045888|ref|NP_937756.1|	3707	365.9	42.67	96.22	13824	11861
8.	novel gene similar to vertebrate polycystic kidney and hepatic disease 1 (autosomal recessive)-like 1 (PKHD1L1) [Danio rerio]	gi|157886135|emb|CAP09460.1|	4191	409.01	37.92	91.63	13824	12314
9.	dynein heavy chain 64C, isoform C [Drosophila melanogaster]	gi|221330856|ref|NP_729034.2|	4638	450.12	44.38	93.38	13824	12512
10.	novel hemicentin protein [Danio rerio]	gi|55962332|emb|CAI11663.1|	5533	551.04	60.97	130.53	13824	12410
11.	nesprin-2 isoform 5 [Homo sapiens]	gi|118918407|ref|NP_878918.2|	6907	689.29	66.37	153.82	13824	12571

the results is done and discussed in the next section.

5. RESULT ANALYSIS

5.1 Quantitative Analysis

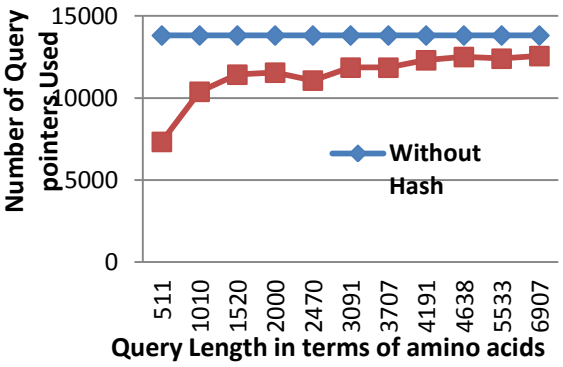
5.1.1 Space complexity

The number of query pointers initialized during structure construction is taken as measuring factor for space complexity. For a small set of query sequences, the results are tabulated in Table 1 given above.

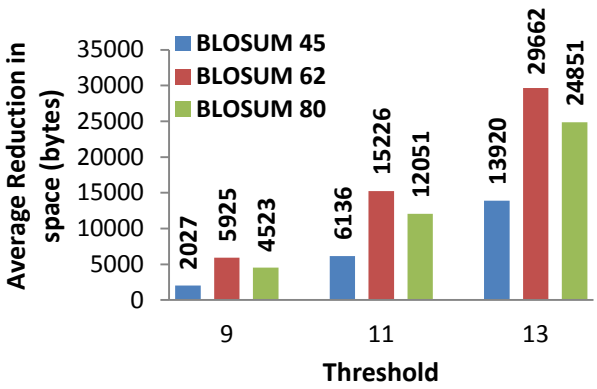
From the results we can see that, the number of query pointers initialized in the existing FSA DFA structure is fixed and is maximum possible. It is dependent on alphabet size 'a' (24 for proteins) and word size 'w' (3 for proteins), i.e., 24^3=13,824 whereas for the proposed structures it is made dependent on size of the query sequence and the number of neighborhood words, each query word has. It is found that the reduction in space is significant and varied based on threshold, BLOSUM and query length. For larger threshold, higher BLOSUM and shorter query, the reduction in space is more. Test results are tabulated in Table 2 and graphically represented in Figure 6 given below.

Table 2. Space efficiency of hashed FSA-DFA as a function of T and BLOSUM scoring matrix

Threshold (T)	**Average Reduction in Memory Space (bytes)**		
	BLOSUM 45	**BLOSUM 62**	**BLOSUM 80**
9	2027	5925	4523
11	6136	15226	12051
13	13920	29662	24851



(a)



(b)

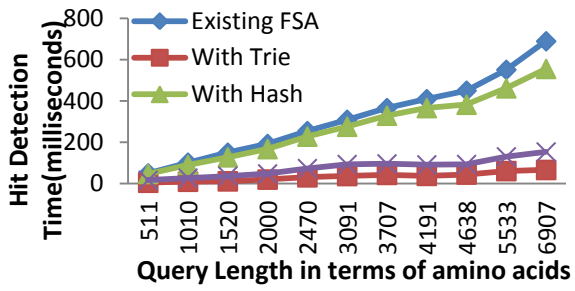
Figure 6: Graphical representation of (a) Hashed FSA-DFA performance (space) with BLOSUM 62 and T=11 in terms of query pointers (b) Average reduction in memory space in terms of bytes when Hashed FSA-DFA is used

5.1.2 Time complexity

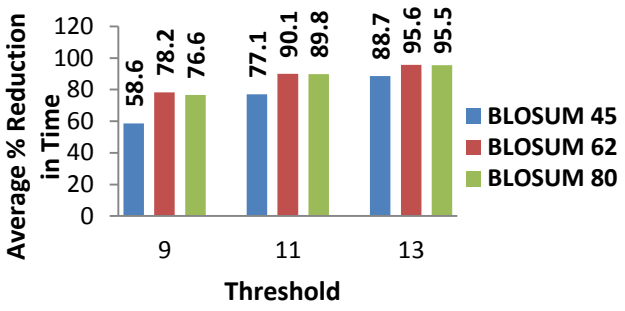
Time complexity of modified structure with the currently being used structure is compared as follows. Performance counter has been set separately to measure the structure construction time for the given query and time for processing the database sequence for finding hits. The least time unit that can be measured by the performance counter is millisecond. For better accuracy, processing of database sequences has been set to run 10 times. It is observed that there is slight increase in processing database sequence time due to hashing function used to map the database sequence word into corresponding query position during the processing stage. But there is drastic improvement in structure initialization time due to modified structure and usage of trie to find the neighborhood words. Percentage reduction in overall time for finding hits using modified structure is calculated for a set of query sequences. The results are graphically represented in Figure 7 given below.

5.2 Qualitative Analysis

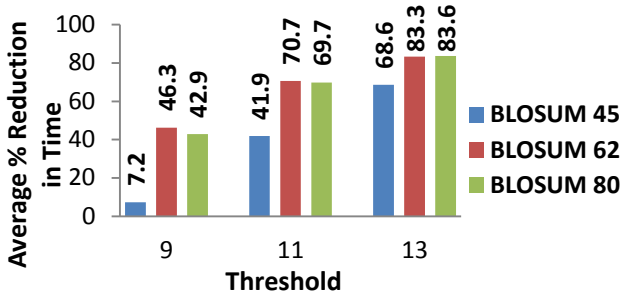
BLAST performs similarity search which can be improved either by increasing sensitivity or selectivity. Sensitivity of BLAST is defined as ability to recognize distantly related data base sequences to that of a query sequence. Selectivity is defined as ability to reject unrelated database sequences to



(a)



(b)



(c)

Figure 7: Graphical representation of (a) performance (time) of FSA-DFA and Hashed FSA-DFA with and without Trie using BLOSUM 62 and T=11 (b) Average percentage reduction in hit detection time of FSA-DFA using Trie (c) Average percentage reduction in hit detection time of Hashed FSA-DFA using Trie.

that of a query sequence. The number of alignments being considered in the second stage of the BLAST is dependent on the number of hits generated in the first stage of the BLAST. For example, for a query of length 110 amino acids, the number of hits recorded for both existing structures and proposed structures is 984, when run on the database sequence of length 48748 amino acids. In the alignment stage, the scores of all 984 alignments are computed and best scored alignments will be taken into account for final output of BLAST. The experimental results tabulated in Table 3 given below reveals that the number of hits recorded for FSA-DFA and modified structures are similar. Hence we can say that, space and time complexity of the algorithm is reduced without effecting sensitivity and selectivity.

Table 3: Number of hits recorded when each query sequence is searched against the database using FSA-DFA and hashed FSA-DFA with and without trie for BLOSUM 62 scoring matrix and T=11

Database sequence	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7	Query 8	Query 9	Query 10	Query 11
1.	99	230	257	466	835	995	1031	960	984	1299	1454
2.	26	485	714	881	2277	2268	241	2217	2012	2993	2920
3.	364	654	959	1338	1461	1949	2551	2853	3003	3326	4223
4.	481	990	1309	1977	2957	3522	4140	4135	3941	5630	5477
5.	1237	1138	1418	2247	2947	3840	5002	4861	4591	6182	6194
6.	819	1262	1632	2496	3672	4695	6109	5999	5048	7604	6806
7.	962	1814	2388	401	4728	6136	7806	7223	7421	9072	10597
8.	1047	1932	2807	4625	4829	6865	7662	8191	8696	10868	13166
9.	1138	3152	2970	465	5928	7258	9066	8846	9584	11996	13475
10.	1187	2488	3216	5333	8158	9762	11183	10238	10311	14800	14455
11.	1184	2226	3394	4891	6886	7978	9447	9950	10325	13440	16842
12.	1336	2735	3749	5922	6903	8628	9919	10003	12077	14232	18940
13.	1446	2875	4142	5781	6522	8785	10771	11694	12788	14831	18760
14.	1418	2970	5874	6252	7613	9654	11424	11457	13436	15532	20642
15.	1867	3474	4872	8938	9373	12716	13397	13634	15662	19093	24937
16.	1952	3809	5288	8504	10204	12449	15017	15020	16494	20876	24898
17.	2247	4658	6252	14760	12981	15711	19217	17616	19693	28200	30829
18.	2947	5928	7613	12981	45804	36870	36106	28078	22690	49048	33906
19.	3840	7258	9654	15711	36870	48317	39363	33411	30611	53490	41796
20.	3579	7208	9609	14742	16003	22216	25958	27616	34976	36971	47964
21.	5002	9066	11424	19217	36106	39363	59007	43725	34445	62938	46425
22.	4861	8846	11457	17616	28078	33411	43725	48883	36491	56969	47771
23.	4591	9584	13436	19693	22690	30611	34445	36491	48748	50307	66640
24.	6182	11996	15532	28200	49048	53490	62938	56969	50307	115243	69181
25.	6194	13475	20642	30829	33906	41796	46425	47771	66640	69181	127841

6. CONCLUSIONS

An alternative data structure for FSA DFA and a trie structure in place of BLOSUM62 matrix, during the structure construction stage have been successfully implemented and tested. It is concluded that:

• Use of the new data structure resulted in significant reduction in runtime space. For smaller query lengths (110 aa) the reduction in run time space was found to be up to 78.7%, for very large query length (6907 aa) it is about 9.1% using a threshold of 11 and BLOSUM 62 scoring matrix.

• Use of trie in place of BLOSUM62 reduced the hit detection time very significantly. Percentage reduction in hit detection time was found to be directly proportional to the query length, i.e., as the query length increases the percentage reduction in hit detection time also increases.

• Use of trie also improves the accuracy of BLAST output. In the existing implementation, the number of neighborhood words to each query word to be taken into account is limited to ~50 on the first come first serve basis (if it exceeds50), whereas trie structure consists of the best 50 words (since trie is constructed from pre-computed lists of neighborhood words arranged in descending order according to word score when compared with T).

• The number of hits detected remained the same for both FSA DFA and hashed FSA DFA, with and without trie for a specific BLOSUM matrix and threshold. It indicates that the use of hashed FSA DFA and trie did not alter the sensitivity and selectivity of the BLAST tool.

• The data structures proposed in this work can be of use for stand-alone BLAST as well as web based one.

The above said improvements can be incorporated into FSA BLAST and the overall reduction in run time space utilization as well as time can be tested on real time databases.

This work focuses only on improving run time space and reducing hit detection time of the structure construction stage. While scanning the database, the number of hits is known when a specific query is compared with each sequence of the database. This information can be used to filter some of the database sequences that have negligible number of hits from being used in alignment stage.

Pre-computed word matrices require more memory than the BLOSUM matrices. This can be overcome by creating a database of tries for all BLOSUM matrices and giving it as one of the input to the BLAST tool.

7. ACKNOWEDGEMENTS

This research work is done independently and is part of Ph.D work. It is not supported by any funding body.

8. REFERENCES

[1] Pertsemlidis, A. and John, W. Fondon III. 2001. Tutorial - Having BLAST with bioinformatics (and avoiding BLASTphemy), Genome Biology 2(10).

[2] Altschul, S. F. Gish, W. Miller, W. Myers, E. W. and D.J. Lipman, D. J. 1990. Basic local alignment search tool. Journal of Molecular Biology, 215(3):403– 410.

[3] Altschul, S. F. Madden, T. L. Schaffer, A. A. Zhang, J. Zhang, Miller, Z. W. and D.J. Lipman, D. J. 1997. Gapped BLAST and PSI–BLAST: A new generation of protein database search programs. Nucleic Acids Research, 25(17):3389–3402.

[4] WU-BLAST [http://blast.wustl.edu/]

[5] Boysen, C. and Marc, A. Rieffel. 2004. Enhancing BLAST Performance by using paracel filtering package, Paracel Technology.

[6] Cameron, M.. Williams, H. E. and Cannane, A. 2004. Improved gapped alignment in BLAST. IEEE Transactions on Computational Biology and Bioinformatics, 1(3):116-129.

[7] Cameron, M.. Williams, H. E. and Cannane, A. 2006. A deterministic finite automaton for faster protein hit detection in BLAST, Journal of Computational Biology 13(4), 965–978.

[8] McGinnis, S. and T.L. Madden, T. L. 2004. BLAST: at the core of a powerful and diverse set of sequence analysis tools. Nucleic Acids Research, 32:W20–W25.

[9] Chen, X. L. 2004. personal communication.

[10] Shapaer, E. G. Robinson, M. Yee, D. Candlin, J. D. Mines, R. and Hunkapiller, T. 1996. Sensitivity and selectivity in protein similarity searches: A Comparison of Smith-Waterman in Hardware to BLAST and FASTA. Genomics,38, 179-191.

[11] Gotoh, O. 1982. An improved algorithm for matching biological sequences. Journal of Molecular Biology, 162(3):705–708.

[12] Jian, Y. McGinnis, S. and Madden, T. L. 2006. BLAST: improvements for better sequence analysis, W6–W9 Nucleic Acids Research, Vol. 34, Web Server issue doi:10.1093/nar/gkl164

[13] Noé,* L. and Kucherov, G. 2004. Improved hit criteria for DNA local alignment, BMC Bioinformatics, 5:149 doi:10.1186/1471-2105-5-149.

[14] Delaney, S. Butler, G. Lam, C. and Thiel, L. Three Improvements to the BLASTP Search of Genome Databases, 0-7695-0686-0/00 $10.00 _ 2000 IEEE

[15] Cameron, M. and Williams, H. E. 2007. Comparing Compressed Sequences for Faster Nucleotide BLAST Searches, IEEE Transactions.

[16] Afratis, P. Galanakis, C. Sotiriades, E. Mplemenos, G. G. Chrysos, G. Papaefstathiou, I. and Pnevmatikatos, D. Design and Implementation of a Database Filter for BLAST Acceleration, 978-3-9810801-5-5/DATE09 © 2009 EDAA.

[17] Guo, X. Wang, H. Vijay, D. Design of a FPGA-Based Parallel Architecture for BLAST Algorithm with Multi-hits Detection, 2011 Eighth International Conference on Information Technology: New Generations, 978-0-7695-4367-3/11 $26.00 © 2011 IEEE, DOI 10.1109/ITNG.2011.122.

[18] Boratyn, G. M. Schaffer, A. A. Agarwala, R. Altschul, S. F. Lipman, D. J. and Madden, T. L. 2012. Domain enhanced lookup time accelerated BLAST, Biol Direct Apr 17;7(1):12.

[19] Anuradha, M. Suman Nelson, K. and Prasad Reddy, P. V. G. D. March 2012. Improved hit detection algorithm for FSA protein BLAST. International Journal of Bioscience, Biochemistry and Bioinformatics, Vol. 2, No. 2:61-65.

[20] Wilbur, W. J. and Lipman, D. J. 1983. Rapid similarity searches of nucleic acid and protein data banks. Proceedings of the National Academy of Sciences USA, 80(3):726–730.

[21] Karlin, S. and Altschul, S. F.1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proceedings of the National Academy of Sciences USA, 87(6):2264–2268.

[22] Altschul, S. F. Boguski, M. Gish, W. and Wootton, J. 1994. Issues in searching molecular sequence databases. Nature Genetics, 6:119–129.

[23] Altschul, S. F. and Gish, W. Local alignment statistics. Methods in Enzymology, 266:460–480, 1996.

[24] Altschul, S. F. Bundschuh, R. Olsen, R.. and Hwa, T. 2001. The estimation of statistical parameters for local alignment score distributions. Nucleic Acids Research, 29(2):351–361.

[25] Henikoff, S. and Henikoff, J. 1992. Amino acid substitution matrices from protein blocks. Proceedings of the National Academy of Sciences USA, 89(22):10915–10919.