Improved Hit Detection Algorithm for FSA Protein BLAST

M. Anuradha , K. Suman Nelson, and P. V. G. D. Prasad Reddy

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. In this work a modified data structure is used for the hit detection process. Codes have been developed separately for use of existing and modified data structures and tested on sample database. It has been found that the use of new data structure results in up to 81% reduction in run time memory requirement and about 20% reduction in hit detection time without affecting sensitivity and selectivity of the algorithm.

***Index Terms*—BLAST, hit detection, neighborhood words, DFA, prefix word table, query pointers, space complexity, time complexity, blosum matrix.**

I. 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) [2], [3], WU-BLAST [4], Paracel BLAST [5] and fast search algorithm (FSA)-BLAST[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 [8]) any improvement to the BLAST algorithm that can reduce runtime space and time without effecting sensitivity and selectivity [9] would be very much desirable. 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], [10-16]. This paper proposes modified data structure that reduces the runtime space during the hit detection stage of the FSA protein BLAST algorithm.

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 stage and the other is using semi-gapped and restricted insertion alignments during alignment stage of the algorithm [6,7]. This paper proposes modified data structure that reduces the runtime space as well as time complexity of the hit detection stage of the 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 because we are initializing only the necessary query pointers.

This paper is organized as follows. The hit detection process in protein BLAST and construction of existing FSA-DFA structure for finding hits are briefly described in section 2. The modified structure is discussed and compared with existing one, in section3. In Section4, implementation details and testing of code are described. Result analysis is described in Section5. Conclusions are given in section6 and scope for further work is discussed in section7.

II. Background

A. Hit Detection Process

In this stage, BLAST compares query sequence with each sequence of the database using Wilbur and Lipmann algorithm [17]. 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.

Manuscript received December 31, 2011; revised February 13, 2012.

M. Anuradha is with the Department Of Computer Science and Systems Engineering, Andhra University, Visakhapatnam – 530003, Andhrapradesh, India (phone: 91-044-22435134; mobile: 91-9444412833; anuradha_malempati@yahoo.com).

K. Suman Nelson is with Google India, Hyderabad, Andhrapradesh, India (e-mail: suman.nelson@gmail.com).

P. V. G. D. Prasad Reddy is with the Department Of Computer Science and Systems Engineering, Andhra university, Visakhapatnam - 530 003, Andhrapradesh, India (e-mail: prof.prasadreddy@gmail.com).

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) [18-20], 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 [21]. 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.

B. 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).

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	NWS
1	BAB	BAC, BBB, BCB, CAB
2	ABB	ABC, ACB,BBB, CBB
3	BBC	BBA, BBB

(c)

Fig. 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 (NWS) along with their query positions (QP) for the given query.

FSA-DFA shown in Figure.2 for an input given in Figure.1 is constructed as follows.

1) $a^{(w-2)}$ 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.
2) 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.

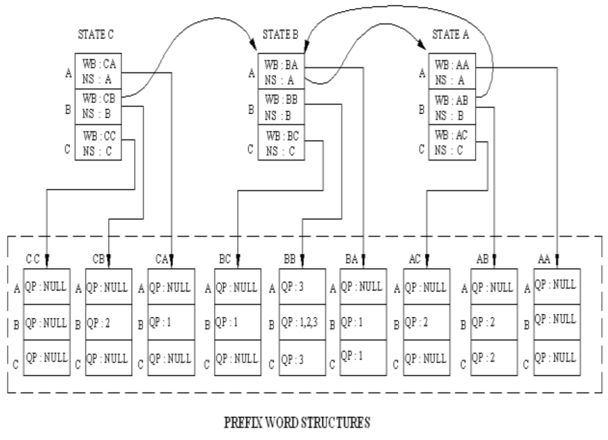

Fig. 2. Existing FSA-DFA constructed for the input given in Fig. 1(a) and (b)

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.

III. Modified Structure

The limitation of currently being used FSA-DFA is, whatever may be the query length, size of the look-up 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 '$w_1w_2w_3$'. For each query word, w_1 is the current state, w2 is the next state and w_3 is the transition that maps query word $w_1w_2w_3$ from current state into corresponding query position in prefix word hash table w_1w_2. 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.

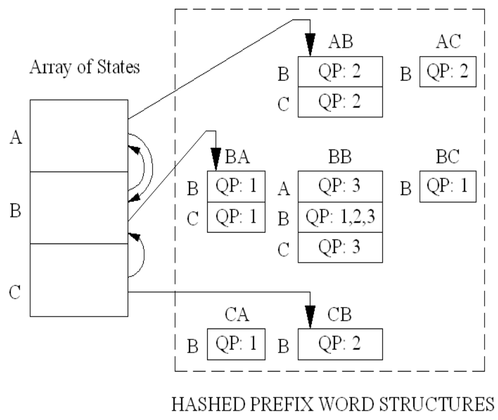

Fig. 3. Modified FSA-DFA constructed for the input given in Figure.1 (a)and(b)

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.

IV. Testing

This section presents implementation details and testing of both existing and modified structures used for finding hits, in the hit detection stage of the FSA protein BLAST program. The algorithms using the existing and modified FSA-DFA structures described in section2 and 3 are implemented in C^{++} language, in visual studio 2010 environment.

To test the code, first the input and output files are created. The input file consists of twenty protein sequences, randomly extracted from the non-redundant protein database of NCBI and Blosum62 scoring matrix. Here, the set of 20 protein sequences extracted are considered as database and each one of the 20, taken one at a time as query to be searched against database. Threshold value needed to find neighborhood words for each query word is hard coded (T=11). 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 and modified FSA-DFA with BLOSUM 62 scoring matrix. Time taken to construct the structure and processing the database sequence has been measured separately using system clock functions. Quantitative and qualitative analysis on the results is done and discussed in the next section.

V. Analysis of Results

A. Quantitative Analysis

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 Table1 given below. From the results we can see that, the number of query pointers initialized by the existing structures 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 where as for the modified 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 percentage reduction in space varied from 18.8 to 81.6%, depending on the query length. Shorter is the query more is the percentage reduction in space.

TABLE I: NUMBER OF QUERY POINTERS USED AND HITS DETECTED IN EXISTING FSA-DFA STRUCTURE AND MODIFIED STRUCTURE.

S.No	**Query length (in aa)**	**Number of query pointers used**		**Number of Hits Recorded**		**% Reduction in run-time space**
		Existing	Modified	Existing	Modified	
1.	110	13824	2548	984	984	81.6
2.	252	13824	2860	2012	2012	79.3
3.	348	13824	4515	3003	3003	67.3
4.	450	13824	6851	3941	3941	50.4
5.	511	13824	6648	4591	4591	51.9
6.	609	13824	7282	5048	5048	47.3
7.	772	13824	8571	7421	7421	38
8.	1520	13824	10176	13436	13436	26.4
9.	2470	13824	9922	22690	22690	28.2
10.	4638	13824	11222	48748	48748	18.8

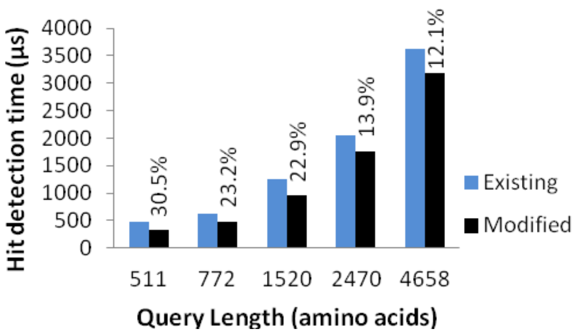

Fig. 4. Reduction of hit detection time (existing Vs modified data structures).

Time complexity of modified structure with the currently being used structure is compared as follows. System clock 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 system clock function is millisecond. Since the processing of database sequence for finding hits runs in microseconds, it has been set to run 1000 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. Since the reduction in structure construction time is more significant (on an average reduced to 23.4%) than the slight increase in processing time, the overall time for finding hits is reduced by 20% on an average. Average percentage reduction in overall time for finding hits using modified structure is calculated for a small set of query sequences. The results are graphically represented in Figure.4 given above.

B. *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 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 structure and modified structure 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.1 reveals that the number of hits recorded for existing FSA-DFA and modified FSA-DFA are similar. Hence we can say that, space complexity of the algorithm is reduced without effecting sensitivity and selectivity.

VI. CONCLUSIONS

- The Alternative data structure for FSA-BLAST has been successfully implemented and tested.
- Use of the new data structure resulted in reducing the runtime memory requirement significantly. For smaller query lengths (110 aa) the reduction was found to be up to 81%, for very large query length (4638 aa) it is about 19%.
- The number of hit detections remained the same for all query sizes indicating that the use of new structures did not alter the sensitivity and selectivity of the BLAST tool.
- The usage of modified data structures also resulted in reduction of total time taken by the algorithm to detect hits. The average percentage reduction in hit detection time is around 20% and showed little correlation to query length.
- While the new algorithm reduces the space and time requirement in the structure construction stage, some time advantage is lost due to the usage of prefix word hash tables in place of prefix word table look-ups. Overall, there is significant advantage in terms of run-time memory and limited benefit in terms of hit detection time.
- In the currently being used structure, state transitions are implemented as pointers to next states. But in the modified FSA-DFA structure these pointers are not needed because states are implemented as a linear array.

It resulted in reducing the run time space by 24x4 bytes per state object, which in turn drastically reduced the overall memory footprint of FSA-DFA structure.

VII. Scope of the Work

The structure is to be further modified so that significant gains can be made in case of space and time complexities simultaneously.

While processing the database, the number of hits per database sequence is known when a specific query is compared with each sequence of the database. This information can be used to filter the database sequences that have negligible number of hits from being used in alignment stage. Implementing such filtering mechanism could further reduce the overall BLAST search time.

References

[1] A. Pertsemlidis and J. W. Fondon III, "Tutorial - Having BLAST with bioinformatics (and avoiding BLASTphemy)," *Genome Biology*, vol. 2, no. 10, 2001.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403– 410, 1990.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI–BLAST: A new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

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

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

[6] M. Cameron, H. E. Williams, and A. Cannane, "Improved gapped alignment in BLAST," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 1, no. 3, pp. 116-129, 2004.

[7] M. Cameron, H. E. Williams, and A. Cannane, "A deterministic finite automaton for faster protein hit detection in BLAST," *Journal of Computational Biology* vol. 13, no. 4, pp. 965–978, 2006.

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

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

[10] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology* vol. 162, no. 3, pp. 705–708, 1982.

[11] J. Ye, S. McGinnis, and T. L. Madden, "BLAST: improvements for better sequence analysis," *W6–W9 Nucleic Acids Research*, vol. 34, Web Server issue doi:10.1093/nar/gkl164

[12] L. No é and G. Kucherov, "Improved hit criteria for DNA local alignment," *BMC Bioinformatics* 2004, 5:149 doi: 10.1186/1471-2105-5-149.

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

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

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

[16] X. Guo, H. Wang, and V. Devabhaktuni, "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 © 2011 IEEE*, DOI 10.1109/ITNG.2011.122.

[17] W. J. Wilbur and D. J. Lipman, "Rapid similarity searches of nucleic acid and protein data banks," in *Proceedings of the National Academy of Sciences USA*, vol. 80, no. 3, pp.726–730, 1983.

[18] S. Altschul, M. Boguski, W. Gish, and J. Wootton, "Issues in searching molecular sequence databases," *Nature Genetics*, vol. 6, pp. 119–129, 1994.

[19] S. F. Altschul and W. Gish, "Local alignment statistics," *Methods in Enzymology*, vol. 266, pp. 460–480, 1996.

[20] S. F. Altschul, R. Bundschuh, R. Olsen, and T. Hwa, "The estimation of statistical parameters for local alignment score distributions." *Nucleic Acids Research* vol. 29, no. 2, pp. 351–361, 2001.

[21] S. Henikoff and J. Henikoff, "Amino acid substitution matrices from protein blocks," in *Proceedings of the National Academy of Sciences USA*, vol. 89, no. 22, pp. 10915–10919, 1992.

Anuradha. Malempati was born in Vuyyur, Krishna District, Andhra Pradesh, India on 1st of July, 1966. She obtained her B.E. degree in Electronics and Communication Engineering from Andhra University, Visakhapatnam, India in the year 1992 and M.E. in Computer Science and Engineering from Madras University, Chennai, India in the year 2000.

She worked as an assistant professor about 7 years in Panimalar Engineering college and Anand Institute of Higher Technology affiliated to Anna University. Currently she is a research scholar in the Department of Computer Science and Systems Engineering at Andhra University. She is working on improving the performance of BLAST algorithm under the supervision of Prof. Prasad Reddy, P.V.G.D. Her research interests include Sequence Analysis, data structures and algorithms.

Suman N. Kancherla was born in Andhra Pradesh, India on the 22nd of April, 1976. He completed his Bachelor of Technology in computer science and engineering from Indian Institute of Technology, Madras, India in 1997.

He worked as a Senior Software Engineer at Amadasoft India Pvt Ltd, as a Computer Scientist in Adobe Systems India, as Software Design Engineer at Microsoft IDC, as Principal Engineer at Xiotech Corporation and currently working as Software Engineer at Google India, Hyderabad (India) campus. His current interests include natural language processing, machine learning, cloud computing and algorithm optimization.

Prof. Prasad Reddy, P.V.G.D, was born in Rajahmundry, East Godavari district, Andhra Pradesh, India on 14th July, 1961. He obtained his B.Tech, in Mechanical Engineering, from Andhra University in the year 1985 and M.Tech, in Computer Science and Technology, in 1987**,** and Ph.D, in Computer Engineering in the year 1993**.**

He joined Andhra University in the year 1987 and gained 24 years of Teaching and Research experience which includes more than 10 years of Administrative experience. As a Professor of Computer Science, his specialization includes Enterprise wide Computing, ERP, XML based object models and scalable web applications. His research areas are Soft Computing, Software Architectures, Knowledge Discovery from Databases, Image Processing, Number theory and Cryptosystems. He has so far successfully guided 14 Ph.D's, 18 M.Phil's and 259 M. Tech Projects in the field of Computer Engineering , and has to his credit, more than 118 Research papers which includes 52 publications in the referred international Journals of repute, 3 Patents and he is currently working as Rector of Andhra University, Visakhapatnam, AP, India..

Prof. Prasad Reddy is a Fellow, Institution of Engineers, INDIA, Member, International Association of Engineers, Member, Indian Science Congress. He has been awarded the BEST Teacher Award by the Government of Andhra Pradesh for the year 2011.