SCIBS (Subset Count Index Based Search) indexing algorithm to reduce the time complexity of search algorithms

Nishad Pm

Ph.D Scholar,
Department of Computer Science NGM College, Pollachi, India
nishadpalakka@yahoo.co.in

Dr. R.Manicka chezian

Associate professor,
Department of computer science NGM College Pollachi, Coimbatore, India
chezian_r@yahoo.co.in

Abstract: There are several algorithms like binary search, linear search, Interpolation search, Ternary search and, etc used for search. Search algorithms locate the position of an item in a sorted. But the time taken for the search is huge. The search algorithm initially set the first and last index for the search; this directly leads to the time complexity. This paper proposes a new prefix search indexing algorithm is called Subset Count Index Based Search Algorithm (SCIBS). This algorithm achieved the effective search with minimum time by restricting search only inside the subset instead of searching in entire dataset. This algorithm prefixes any search algorithm, reduces the time complexity by defining the proper subset for the search. The SCIBS algorithm reduces the comparison ratio. The algorithm is very effective on huge collection of data. The search time will reduce based on the length L. The experimental result shows 99.97 percentage improvements on linear search and 93.34% improvement on binary search if the Length equals two.

Keywords: Binary, linear, Interpolation, Ternary, algorithms, Subset, Count and Index

1. Introduction

Binary search are a fundamental data structure. There are two complexity metrics commonly associated with them: the maximum number of comparisons made during a search and the expected number of comparisons made during a search. Minimizing these metrics, separately or together, has been extensively studied. The difficulty of doing this depends on how much information is given. There are $2n+1$ possible outcomes of a search in a binary search on n keys: the search target could be less than the smallest key; for each key, the search target could equal that key; for each consecutive pair of keys, the search target could be strictly between them; or the search target could be greater than the largest key. For a fixed probability distribution P over these outcomes, the *cost* of a binary search is the expected number of comparisons made during a search. In this paper show how. SSCIBS algorithm reduces the worst case and average case of a binary search .A binary search with minimum cost is called *optimal*. There are algorithms for constructing optimal [5] or nearly optimal binary search [9,10,11], and algorithms for constructing optimal binary search whose heights are restricted [2,3,13]. The latter are useful because the maximum number of comparisons made during a search cannot be greater than the height plus 1. Thus, height-restriction can be used to produce binary search that perform well in both the worst case and the expected case. However, all of these algorithms require us to know the probability of each outcome. All known algorithms for constructing optimal binary search, whether height-restricted or not, use dynamic programming. Knuth's algorithm [5] runs in $O(n2)$ time, where n is the number of keys. Itai [3] and Wessner [12] independently improved Garey's algorithm [2] to run in $O(n2L)$ time, where L is theheight restriction. Their algorithm gives us an optimal binary search of height h and its cost, for each h with $[\log n] <= h <= L$. By log, we always mean log2. Mehlhorn's and Larmore's algorithms [9,10, 11] can be implemented to run in $o(n2)$ time, but the they produce are not always optimal. Evans and Kirkpatrick [1] presented an algorithm for constructing binary search that does not require the probabilities of the outcomes. Given one binary search on n keys, it constructs another binary search of height at most $\log n+1$ whose cost is at most $\log(n + 1)$ greater than T 's. This takes $O(n)$ time. If T performs well in the expected case, then T performs well in both the worst case and the expected case. They called this *restructuring T.* They also proved tight tradeoffs for the worst-case cost increase as a function of the height restriction and n.

2. SUBSET COUNT INDEX BASED SEARCH ALGORITHM

The proposed Subset Count Index Based Search Algorithm has several phases. The block diagram is of Subset Count Index Based Search Algorithm is given in figure-1.

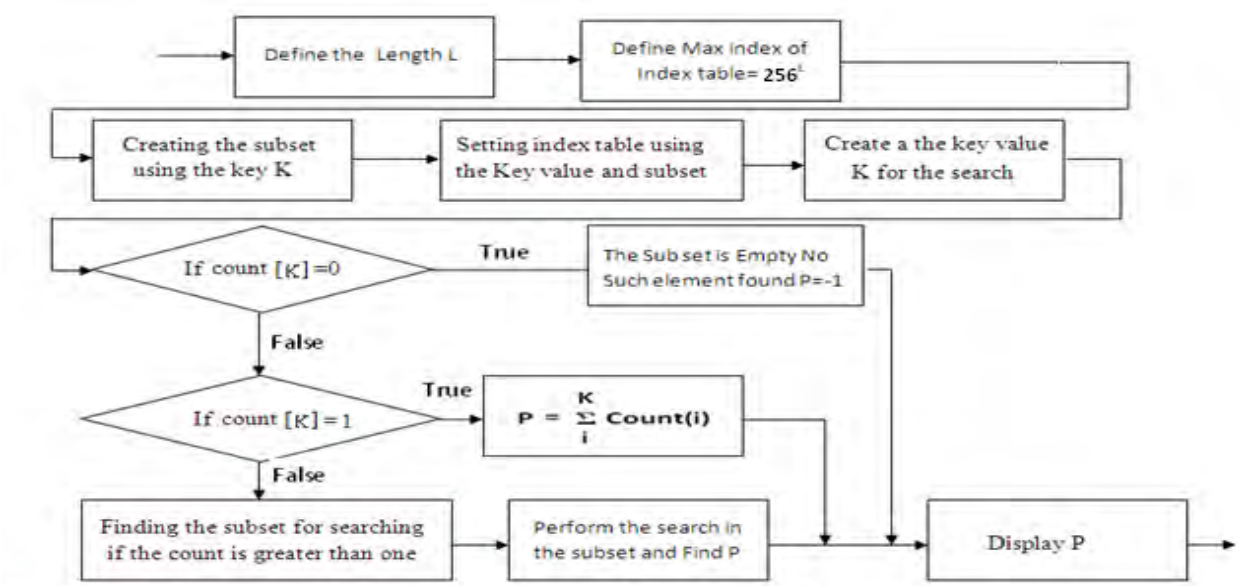


Figure -1 Block diagram of Subset Count Index Based Search Algorithm

2.1 DEFINE THE LENGTH L

The first phase of the algorithm is defining the length L, the Length L and number of subset is directly depended. If the L increases the subset increases, the L decreases the subset also decreases. By increasing the length L the element inside the subset can be reduced, so this directly leads to reduce the comparison ratio and result to the reduced time complexity. Based on the key length the index length will varies. Index value is used to index the subset for example if index equals to nine that indicates the tenth subset. Let us assume the key length is L by default the L equal to one so the max Index equals 256, by increasing the key length the index length also varies that is 256^L index table is shown in figure-2.

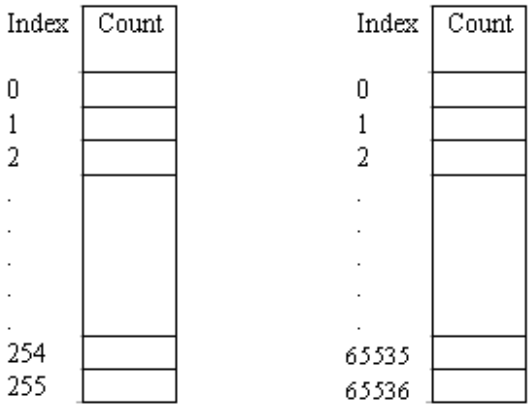


Figure -2 Index table if L=1 **Index table if L=2**

2.2 CREATING THE SUBSET USING THE KEY K

Each index indicates an individual subset and count indicate the number of members is available in the set. For example the S is a set that contains n number of members.

S= {111, 1122, 134, 6553, 6554, 884, ANANDU, ANANDI, ANANTHU ARAVIND, MIDHUN, SAMEER, SANGEETH, SANGEETHA, VISHAK, ZAHEER}

The formula (1) is applied on each element on the set S and the K value is calculated then the K^{th} Index count is incremented by one. For example if L equal to two then first two characters is fetched form each member of S, calculate the K value based on the formula (1) and constructed subset using the K value.

$$K = \sum_{i=1}^{L} ((\text{ASCII (char at(i))}-1)*(256^{(L-i)}))+1 \quad (1)$$

For the above set each member we creates a Key value K using the above formula

If L=3 then

For each member of S the key value K is created.

Key Set= {3158065, 3158066, 3158580, 3486773, 3486773, 3618612, 4214081, 4214081, 4214081, 4215105, 4999236, 5390413, 5390414, 5390414, 5589075, 5849160}

Using the K in key set several subset is created sub set is created

K_1 (3158065) = {111}

K_2 (3158066) = {1122}

K_3 (3158580) = {134}

K_4 (3486773) = {6553, 6554}

K_5 (3618612) = {884}

K_6 (4214081) = {ANANDU, ANANDI, ANANTHU}

K_7 (4215105) = {ARAVIND}

K_8 (4999236) = {MIDHUN}

K_9 (5390413) = {SAMEER}

K_{10} (5390414) = {SANGEETH,

SANGEETHA}

K_{11} (5589075) = {VISHAK}

K_{12} (5849160) = {ZAHEER}

Where each sub set of S is disjoint sets

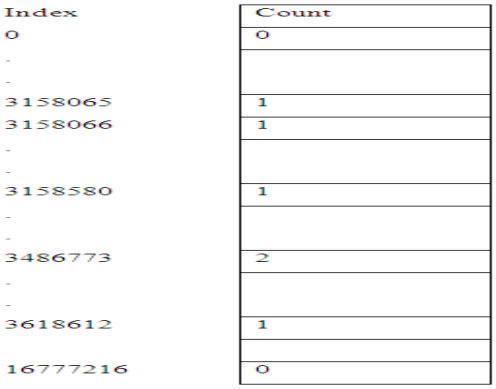
2.3 SETTING INDEX TABLE USING THE KEY VALUE AND SUBSET

In this phase evaluates the number of element in each subset and updated the K^{th} index. The max index of the index table equal to 256^L

If the value of L=3

So the Index Max = 16777216

Initially the subset K_1 is fetched and the K of K_1 is treated as index and $n(K_1)$ is treated as count in the index table this process will continue up to the last subset. The index table is shown in figure-3 each index in an index table is used to indicate the subset or the index value is the key value for the each subset.



Index	Count
0	0
.	
.	
3158065	1
3158066	1
.	
.	
3158580	1
.	
.	
3486773	2
.	
.	
3618612	1
16777216	0

Figure - 3 index tables for the Set S

$$\text{The n(S) equals} \quad \sum_{i=0}^{\text{Max index(index table)}} \text{Count(i)} \qquad (2)$$

2.4 CREATE A THE KEY VALUE K FOR THE SEARCH

Before performing the search operation the key value K is generated for that particular item to be searched in the set S the formula(1). For example the value is **"ANANDI"** then the first three character is fetched because L=3, to produce the key value
Then after applying the above formula
K= (((65-1)*65536) + ((78-1)*256) + ((65-1)*1)) +1
Then

K= 4214081

Using the Key value K from the index table the Location of the subset can easily find
Then check the count for the index using the key value K. if count =0 then the subset = {} or empty then it will return empty so the algorithm return position P= -1. Else if the count is one then it wills return the sum of count of index table up to the key value or K using the formula(3) given below.

$$P = \sum_{i}^{K} \text{Count(i)} \qquad (3)$$

2.5 FINDING THE SUBSET FOR SEARCHING IF THE COUNT IS GREATER THAN ONE

If the count is greater than one to performing the search the position of subset in S should be determine for that the Start and end of the subset in the set S is find using the equation **(**4) given below

$$\text{Start} = \sum_{i=0}^{K-1} \text{Count(i)} \qquad (4)$$

Using the above equation the beginning of subset in set S is determined. The next step is to find the termination of subset in the Subset in the set S for that the following equation is used

$$\text{End} = \sum_{i=0}^{K} \text{Count(i)} \qquad (5)$$

Using the above formulas the position of the subset is defined now the search is performed in-between those two values that is start and end. For this purpose any type of search algorithm can be used. The time ratio is reduced because of the search is performed only within the subset. The time ratio after Appling the algorithm on Linear and binary is shown in table -1 only the search is performed in the Subset denoted by the Key value K. E_ Binary(After applying the Subset Count Index Based Search Algorithm on binary search algorithm), E_ Linear (After applying the Subset Count Index Based Search Algorithm on Linear search algorithm),

For example if the key value K= 4214081 because the value to search is ANANDI and the L=3 then the start value = 6 and end value = 9. The 6 and 9 indicate the range of subset in the set S so the search is performed

only within that range so the time ratio is improved. So if we are using the binary search after the Subset Count Index Based Search Algorithm then the search performed only within the 6 to 9 that is shown in figure -4

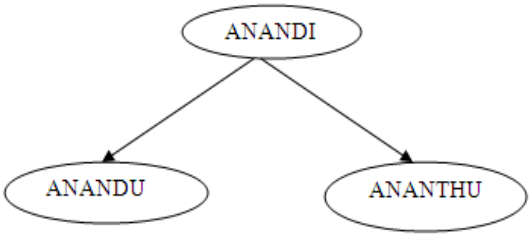
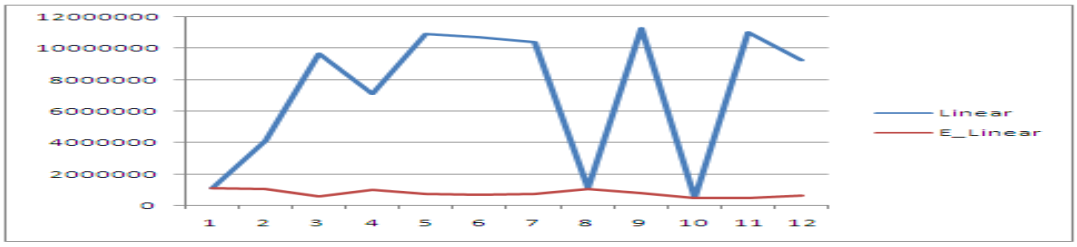


Figure 4 Binary tree after finding the Key value K

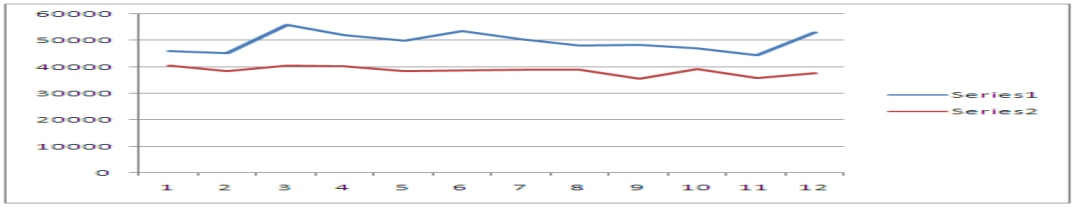
Table 1 time ratio in nanoseconds if the L=1

S no	values	Linear	E_Linear	Binary	E_Binary
1	12345	1126680	**1108241**	46095	**40508**
2	22222	4177346	**1075276**	45257	**38273**
3	54321	9702630	**596724**	55873	**40508**
4	3210	7122972	**1019683**	51962	**40307**
5	87652	10933792	**729422**	50006	**38273**
6	9999	10746896	**708749**	53638	**38552**
7	77777	10412496	**738920**	50286	**38831**
8	12134	1109917	**1048737**	48051	**38832**
9	98765	11296128	**781384**	48331	**35480**
10	10203	501181	**471847**	46933	**39112**
11	91827	11032967	**493917**	44419	**35758**
12	45981	9249779	**651479**	53079	**37714**
Average		7,284,398.67	**785,364.92**	49,494.17	**38,512.33**

The time improvement graph is shown in Graph-1 and graph -2 for linear search and Binary search after using the Subset Count Index Based Search Algorithm.

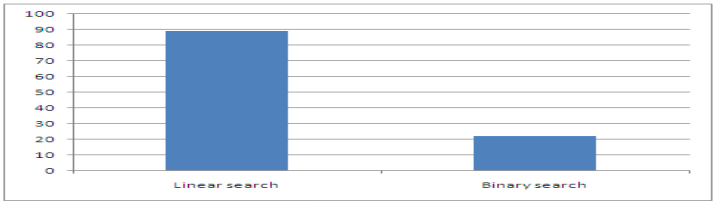


Graph –1 time ratio graph for Linear search after and before Enhancements in nanoseconds



Graph –2 time ratio graph for Binary search after and before Enhancements in nanoseconds

Graph -3 represents the improvement of average time ratio on binary and linear search algorithms if the L=1. 89.218347 percentage improvement on linear search, 22.18813665 percentage on binary search algorithm if L=1 by incrementing the length of L we can again improve the time ratio using this algorithm. The improvement of time ratio is shown in table-2 if the L=2. And also the time improvement graph is shown in Graph-4 and graph -5 for linear search and Binary search after using the Subset Count Index Based Search Algorithm if the L=2.



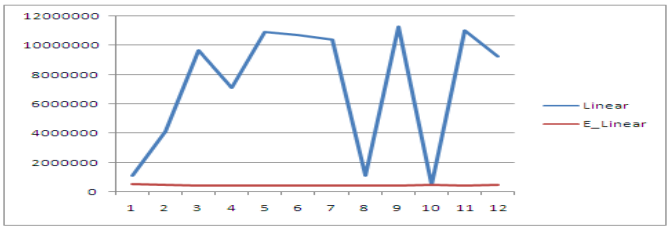
Linear search	Binary search
89.21853467	22.18813665

Graph -3 time ratio improvement ratio after applying the Subset Count Index Based Search Algorithm on linear and binary search

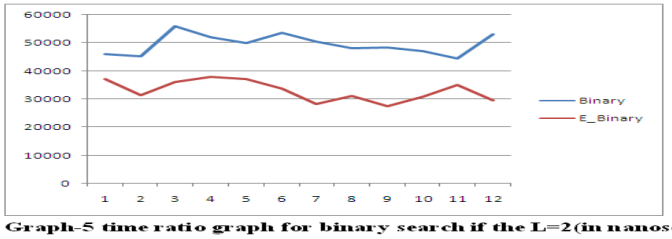
Table -2 Time ratio in nanoseconds if L=2

S NO	values	Linear	E_Linear	Binary	E_Binary
1	12345	1126680	**550350**	46095	**37155**
2	22222	4177346	**514032**	45257	**31289**
3	54321	9702630	**452851**	55873	**36038**
4	3210	7122972	**460953**	51962	**37993**
5	87652	10933792	**472406**	50006	**37155**
6	9999	10746896	**460673**	53638	**33803**
7	77777	10412496	**475200**	50286	**28215**
8	12134	1109917	**476597**	48051	**31009**
9	98765	11296128	**469892**	48331	**27377**
10	10203	501181	**499225**	46933	**30730**
11	91827	11032967	**469613**	44419	**35200**
12	45981	9249779	**482743**	53079	**29612**
		14221065.33	**482044.5833**	49494.2	**32964.66667**

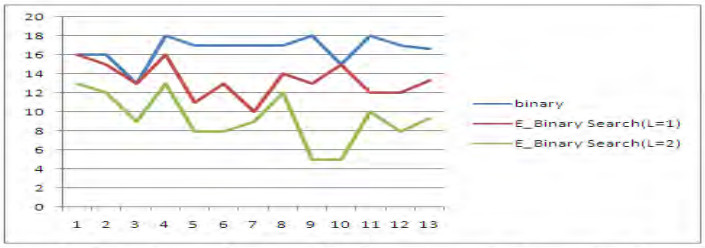
The time comparison graph is shown in graph -4 and graph -5 for linear search and binary search respectivly. The graph -4 and graph-5 is showing the improvement of time ratio after incrementing the length because by incrementing the length the L correspondingly the number of subset also increased by defining the key value K so by increasing the subset that leads to the reduction on number of elements in the subset then the searching is very easy on that occasion with limited number of symbols. By increasing the length L as 2 the percentage of improvement on linear search is 93.38250684 and for binary search is 33.39686495. While comparing L=1 and L=2 the difference ratio is 11.20873 for binary search and 4.16416 percentage more than for linear search.



Graph-4 time ratio graph for linear search if the L=2 (in nanoseconds)



Graph-5 time ratio graph for binary search if the L=2 (in nanoseconds)



Graph -7 Lookup value graph for binary search after and before concatenating with Subset Count Index Based Search Algorithm

3 MERITS OF THE SUBSET COUNT INDEX BASED SEARCH ALGORITHM

Any other algorithm like binary search ternary search and linear search etc., used for search then the search is done in the whole set S but using the Subset Count Index Based Search Algorithm as a first phase then search done only inside the subset and search is very easy because the key value K help us to find the range of the value or the subset of the value then the search operation is performed only within that subset, so such operation leads to increase the ability of the search in short time.

Table -3 Lookup values for binary search and linear search after and before implementing Subset Count Index Based Search Algorithm

Values	Binary	Linear	E_Binary Search (L=1)	E_Linear search (L=1)	E_Binary Search (L=2)	E_Linear search (L=2)
12345	16	25797	16	25797	13	3797
22222	16	135432	15	24666	12	2466
54321	13	334246	13	4755	9	355
3210	18	245077	16	23311	13	1111
87652	17	370910	11	8419	8	719
9999	17	384481	13	10990	8	1090
77777	17	360047	10	8556	9	856
12134	17	23475	14	23475	12	1475
98765	18	383134	13	9643	5	843
10203	15	2234	15	2234	5	2234
91827	18	375502	12	2011	10	911
45981	17	325072	12	6581	8	1081
	16.5833333	247117.25	13.33333333	12536.5	9.333333333	1411.5

Updation of new value to the set S then finding the insertion position is difficult using the other algorithms but using the Subset Count Index Based Search Algorithm easily find the key value K and using K Locating the subset and only check insertion point within that Subset on instead of searching the whole set S. if we are using the index sequential search. Then after incretion of new element to the set S then all index range should be updated after the position of insertion. But using this algorithm only the count of the particular subset alone updated after insertion or deletion.

By increasing the length L the number of subset of S increases, so by increasing the subset the ability of searching can increase because by increasing the sub set the number of elements in the subset or range is decreased, then search operation can only performed with few collection of values, hence the time ratio is increased.

4 CONCLUTION AND FUTURE ENHANCEMENT

This paper proposed a new indexing algorithm called Subset Count Index Based Search. This algorithm reduces the time complexity of any search algorithm to the maximum by restricting the search only inside the subset using the key value k. If the length L increases the number of elements in each subset reduces that leads to reduce the comparison ratio. So this algorithm gives maximum accuracy in time. The experimental result shows 99.97 percentage improvements on linear search and 93.34% improvement on binary search if the Length equals two.

The proposed work can be further enhanced and expanded for the authentication of search techniques to obtain optimum accuracy in time.

References

[1] W.S. Evans, D.G. Kirkpatrick, Restructuring ordered binary trees, J. Algorithms 50 (2004) 168–193.
[2] M.R. Garey, Optimal binary search trees with restricted maximal depth, SIAM J. Comput. 3 (1974) 101–110.
[3] A. Itai, Optimal alphabetic trees, SIAM J. Comput. 5 (1976) 9–18.
[4] J.L.W.V. Jensen, Sur les fonctions convexes et les inégalités entre les valeurs moyennes, Acta Math. 30 (1906) 175–193.
[5] D.E. Knuth, Optimum binary search trees, Acta Inform. 1 (1971) 79–110.
[6] D.E. Knuth, 2nd ed., The Art of Computer Programming, vol. 3, Addison-Wesley, Reading, MA, 1998.
[7] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Statist. 22 (1951) 79–86.
[8] L.L. Larmore, A subquadratic algorithm for constructing approximately optimal binary search trees, J. Algorithms 8(1987) 579–591.
[9] K. Mehlhorn, Nearly optimal binary search trees, Acta Inform. 5 (1975) 287–295.
[10] K. Mehlhorn, A best possible bound for the weighted path length of binary search trees, SIAM J. Comput. 6 (1977) 235– 239.
[11] C.E. Shannon, A mathematical theory of communication, Bell System Techn. J. 27 (1948) 379–423, 623–656.
[12] R.L. Wessner, Optimal alphabetic search trees with restricted maximal height, Inform. Process. Lett. 4 (1976) 90–94.

Biography

Nishad PM M.Sc., M.Phil. Seven months Worked as a project trainee in Wipro in 2005, five years experience in teaching, one and half year in JNASC and Three and half year in MES Mampad College. He has published eight papers national level/international conference and journals. He has presented three seminars at national Level. Now he is pursuing Ph.D Computer Science in Dr. Mahalingam center for research and development at NGM College Pollachi.

Dr. R.Manicka chezian received his M.Sc., degree in Applied Science from P.S.G College of Technology, Coimbatore, India in 1987. He completed his M.S. degree in Software Systems from Birla Institute of Technology and Science, Pilani, Rajasthan, India and Ph D degree in Computer Science from School of Computer Science and Engineering, Bharathiar University, Coimbatore, India. He served as a Faculty of Maths and Computer Applications at P.S.G College of Technology, Coimbatore from 1987 to 1989. Presently, he has been working as an Associate Professor of Computer Science in N G M College (Autonomous), Pollachi under Bharathiar University, Coimbatore, India since 1989. He has published thirty papers in international/national journal and conferences: He is a recipient of many awards like Desha Mithra Award and Best Paper Award. His research focuses on Network Databases, Data Mining, Distributed Computing, Data Compression, Mobile Computing, Real Time Systems and Bio-Informatics.