Simplified Optimal Parenthesization Scheme for Matrix Chain Multiplication Problem using Bottom-up Practice in 2-Tree Structure

Article · November 2011

CITATION
1

READS
1,941

1 author:

Biswajit R Bhowmik
Indian Institute of Information Technology Bhagalpur
46 PUBLICATIONS **252** CITATIONS

SEE PROFILE

Simplified Optimal Parenthesization Scheme for Matrix Chain Multiplication Problem using Bottom-up Practice in 2-Tree Structure

Biswajit BHOWMIK
Department of Computer Science & Engineering, Indian Institute of Technology Guwahati,
Assam – 781 039, India
b.bhowmik@iitg.ernet.in

***Abstract*–Dynamic Programming is one of the sledgehammers of the algorithms craft in optimizations. The versatility of the dynamic programming method is really appreciated by exposure to a wide variety of applications. In this paper a modified algorithm is introduced to provide suitable procedure that ensures how to multiply a chain of matrices. The model Optimal Parenthesization Scheme using 2-Tree Generation (OPS2TG) acts as one relevancies of the proposed algorithm. A new approach for how to break the matrix chain and how to insert parenthesis in it is also designed in this model. The comparison study between the proposed model and the classical approach shows acceptance of the model suggested here.**

Keywords: **Scalar Multiplication, Optimal Parenthesization, Chain Matrix, Solution Matrix, Optimal Cost, Sequence of Decisions, Optimal Solution.**

I. INTRODUCTION

Dynamic Programming (DP) is one of the elegant algorithm design standards and is a powerful tool which yields classic algorithms for a variety of combinatorial optimization problems such as shortest path problems, traveling salesman problem, knapsack problem, etc including underlying *Matrix Chain Multiplication (MCM)* problem. The few past decades throughout many "fast" algorithms have been proposed for matrix multiplications [1]. Some of them are acceptable to some extent. Some of them even are in use till now. In this paper an alternative approach has been described extending a similar approach [2][3][4]. The proposed is named *Optimal Parenthesization Scheme using 2-Tree Generation (OPS2TG)* model.

The paper is organized in the following way. First, a general insight is provided into the traditional approach with a simple example. Second, a formal modification of the existing algorithm is given to show computation of cost of scalar multiplications for the chain of matrices conformable to multiplication in tabular form and enhances constructing solution table under certainty [2][5][6][7]. Third our re-devised algorithm is stated that provides us simple solution for optimal parenthesization of matrices in the chain. The paper ends with comparison study between traditional approach, dynamic programming approach [2][[7], and our proposed approach.

II. TRADITIONAL APPROACH

Let A_{mxn} and B_{nxp} be two matrices, then their product C = AB is an mxp matrix. This C can be computed in O(nmp) time, using very traditional elegant matrix multiplication scheme [7][8] that runs in $O(n^3)$ time. For more than two matrices say X, Y, Z conforming matrix multiplication condition, the purpose can be achieved in two ways as either ((XY)Z) or (X(YZ)) as matrix multiplication is associative. Similarly, for four matrices say A, B, C, and D, they may be multiplied either of the orders: (((AB)C)D), ((AB)(CD)), (A((BC)D)), ((A(BC))D), or (A(B(CD))) [7][9]. In all these cases however multiplication doesn't affect the final outcome. But the choice can affect the running time to compute it [6][10]. If the matrices are of different dimensions, the number of operations termed as scalar multiplications (which makes a huge difference in costs) is affected by the order in which the product is computed [10]. This computation for an optimal order which minimizes the total number of scalar multiplication operations depends fully on parenthesizing the chain of the matrices. Thus, it minimizes the number of scalar multiplications resulting minimum cost [11].

Now, a sequence of n matrices requires n-1 multiplications. The cost of multiplying this chain obviously depends on the order in which the n-1 multiplications are carried out. The rearrangement which results out the minimum number of scalar multiplications can be found in many ways. In the sequence there are n-1 places where each parenthesization defines a set of n-1 matrix multiplications. The sequence is segmented with the outermost pair of parentheses as just after the 1st matrix, just after the 2nd matrix… and just after the $(n-1)^{th}$ matrix. In this way we are able to pick the best parenthesized matrix order [2][4][7][12]. A question may arise in mind about how many parenthesization are possible? Eqn. (1) gives the answer.

In general, the number of orderings is equal to the number of ways to place parentheses to multiply the *n* matrices in every possible way. Let us suppose *P(n)* be the number of to fully parenthesize a product of n matrices. Suppose we want to perform the multiplication *(MiM2…Mk)(Mk+iMk+2…Mn).* Then, there are *P(k)* ways to parenthesize the first *k* matrices. For each one of the *P(k)* ways, there are *P(n-k)*

ways to parenthesize the remaining *n-k* matrices, thus for a total of *P(k)P(n - k)* ways. Since *k* can be assumed any value between 1 and *n* – 1, the overall number of ways to parenthesize the *n* matrices is given by the summation:

$$P(n) = \begin{cases} 1, n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k), n \geq 2 \end{cases} = \Omega\left(4^n / n^{3/2}\right) \quad (1)$$

(Using Catalan numbers and Stirlings' formula)[2][4].

It will be heuristics to find the optimal solution that computes all the rearrangements. Both the number of solutions and the brute-force method of exhaustive searching as expected will become a poor tactic to determine optimal parenthesization in the matrix chain [2][13].

III. PROPOSED MODEL

Dynamic programming [14] is one of the suitable standards and powerful technique for optimal matrix parenthesization, which is discussed in detail in the proposed OPS2TG model. The results and their analysis reveal considerable amount of time reduction compared with the existing algorithms [2][9]. Efficiency for optimal solution can be achieved through:

(a) Recursive definition for cost of subproblems computation.
(b) Computing Optimal cost of scalar multiplication using bottom-up.
(c) Generating Tree structure for the chain.
(d) The optimal parenthesization using bottom-up.

A. Formulating the Problem

DP approach is basically a potent algorithmic stage wise search method of optimization problems whose solutions may be viewed as the result of a sequence of decisions i.e. solutions that recursively solve sub-problems which overlap [7][15]. Naturally the number of scalar multiplications is computed recursively in terms of the optimal solutions to subproblems. The partial solution is stored in a table m namely "cost multiplication" table. The result is then fed to derive optimal cost of solution.

Let us consider M1M2…….MiMi+1…… Mn a given chain of n matrices and Mij, 1≤i≤ j≤n, is the expected resultant matrix. Its cost can be obtained as: Suppose m[i][j] is the minimum number of scalar multiplications needed to find Mij, and P is a list that contains dimension values of the matrices in chain. Here Pi−1xPi represents dimension of Mi matrix. Now, m[i][j] can recursively be defined at the same time like below:

(a) If i = j, m[i][j] = 0. The problem becomes trivial. No need of scalar multiplication.
(b) If i < j, we assume that the optimal parenthesization splits MiMi+1…..Mj between Mk and Mk+1, i ≤ k < j.

Then m[i][j] would be a minimum cost of computation of the sub problems Mik and Mk+1j + The cost of multiplying these two matrices together. i.e.

m[i][j]= m[i][k] + m[k + 1][j] + Pi−1PkPj .

Thus, m[i][j] could be represented by the recurrence as:

$$m[i][j] = \begin{cases} 0, & i = j \\ \min_{i \leq k < j} \begin{cases} m[i][k] + \\ m[k+1][j] + \\ P_{i-1}P_kP_j \end{cases} & if \quad i < j .. \end{cases} \quad (2)$$

B. Computation of Scalar Multiplications

The value at m[1][n] is our desired optimal number of scalar multiplications. But the computational procedure of m[1][n] for M1M2……Mn using m[i][j], takes $O(n^2)$ time which is not better than brute force method of checking each way of parenthesizing the product [1][2]. For improvement, instead of computing m[i][j] recursively we construct this table m shown in Table II using modified algorithm OSM_MCM() that goes after bottom up approach at stages in computation. At the same time another table S[1…n][1…n] called "solution matrix" is constructed. The S[i][j] keeps the position of parenthesization (inserting parenthesis in chain) k in the chain. This table S shown in Table III is used for finding optimal solution.

Algorithm: OSM_MCM(C)

Description: OSM_MCM means Optimal Scalar Multiplication for MCM.

Input: C = matrix chain
n = number of matrices.
P = i/p sequence represents matrices dimension.

Output: Tables m and S.

```
(1)   Repeat for i = 1 to n     // fill diagonal of m
(2)     m[i][i]  = 0
(3)   Initialize x = 2
(4)   Repeat for  i = 1 to n – 1  // fill m[i][j] when i < j
(5)     Initialize j = x
(6)     Repeat while  j ≤ n
(7)       m[i][j] = ∞
(8)     x = x + 1
(9)   Initialize  x = 2
(10)  Repeat while x ≤ n through line 14
(11)    Initialize
(12)    i = 1
(13)    j = x
(14)    Repeat while (i ≤ n) through line 19
(15)    If j ≤ n
(16)      Repeat for k = i to j – 1 //check all possible splits
(17)       m[i][j]=min(m[i][j],m[i][k]+m[k+1][j]+Pi-PkPj)
(18)       S[i][ j] = k
(19)      Initialize j = j + 1
(20)  Output "Cost of multiplication" = m[1][n]
(21)  Exit.
```

Complexity Analysis: Every entry in the table m is 0 along

the main diagonal and unknown value (∞) in the rest of cells above this diagonal shown in Table I(a). It is seen that only the upper triangular matrix [16] is sufficient to find cost of scalar multiplication. The modified structure of m is shown in Table I(b). The m in Table I is constructed for n = 5. Thus lines 1 to 8 halve both time and space complexities to construct the table m. And, it is $\Theta(n^2)$. Now lines 10 and 14 iterate at most n-1 and n times respectively whereas looping varies in line 16. Although the order is cubic but it would be $\Theta(n^3)$. Thus, finding these time and space complexities of Algorithm OSM_MCM is straightforward. For some constant c > 0, the running time T(n) of the algorithm is proportional to:

$$T(n)=\sum_{x=2}^{n}\sum_{i=1}^{n}\sum_{k=1}^{n-x}\frac{cn^3-cn}{10}=\Theta(n^3) \qquad \ldots\ldots (3)$$

TABLE I: INITIAL CONFIGURATION OF M.

i \ j	1	2	3	4	5
1	0	∞	∞	∞	∞
2		0	∞	∞	∞
3			0	∞	∞
4				0	∞
5					0

(a)

j

1	2	3	4	5	
0	∞	∞	∞	∞	1
	0	∞	∞	∞	2
		0	∞	∞	3
			0	∞	4
				0	5

i

(b)

Similarly space complexity for S can be put up as well.

$$\sum_{i=1}^{n}\sum_{j=1}^{n-i}\frac{cn^2-cn}{5}=\Theta(n^2) \qquad (4)$$

C. Construction of Tree Structure

The algorithm that directly shows the next step after the computation of most favorable scalar multiplications how to construct an optimal solution with the help of the table S[1..n][1..n] is described in this section. The algorithm generates a 2-tree [17] as the output when executes.

Algorithm: TT_MCM(S, 1, n) // TT_MCM means 2-Tree generation for MCM

Input: S = nxn matrix obtained in the previous algorithm.

Output: Generated 2-Tree.

Description: the algorithm generates a 2-tree based on the solution matrix S. All the internal nodes including root are represented by (S, i, j). The leaves are the matrices of the chain. Thus the leaf (S, i, i) represents a matrix Mi.

(1) Initialize
(2) i = 1
(3) j = n
(4) Construct root node of the 2-tree
(5) R = (S, i, j) // R = current root node of the tree
(6) k = S[i][j]
(7) Recursively construct children nodes of R in the 2-tree at tentative k.
(8) Lchild = TT_MCM(S, i, k) // Left child node
(9) Rchild = TT_MCM(S, k+1, j) // Right child node
(10) Exit.

Complexity Analysis: The algorithm generates a 2-tree through recursion. Complexity of any binary tree construction will be applicable undoubtedly. Thus the running time will be Ω(nlogn). The generic view of a 2-tree shown in Fig. 1 is generated by the algorithm TT_MCM(). The node (S, i, j) represents multiplication of the chain MixMi+1x……Mj

D. Optimal Parenthesization

This section describes how a parenthesis is inserted in the chain i.e. a chain of matrices is parenthesized. The algorithm given below inserts parenthesis in the 2-tree generated in previous algorithm in bottom-up fashion.

Algorithm: OP_MC(T) // OP_MC means Optimal Parenthesization on Matrix Chain

Input: The generated 2-tree
Output: Parenthesized matrix chain
(1) Repeat for i = 1 to n
(2) Replace the leaf (S, i, i) by Mi
(3) Repeat for x = last level h to root level // bottom up fashion.
(4) Parenthesize all leaves say A, B at x with same parent as (AB) from left to right.
(5) h = h -1
(6) Take (AB) as new leaf at level h in place of parent node of A, and B.
(7) Exit.

Complexity Analysis: It is clearly observed that complexity of this algorithm is primarily due to parenthesizing the contemporary leaves climbing up the tree. The parenthesization starts at lowest level h and ends at root level. Parenthesizing these leaves at every level requires constant time. Thus the running time of the algorithm is O(h) = O(logn).

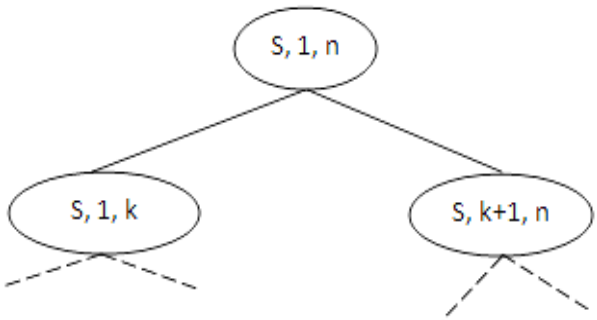


Fig. 1: Typical 2-Tree for Optimal Parenthesization

IV. Numerical Results

Suppose, the chain of 5 matrices <M1, M2, M3, M4, M5> be ABCDE. Their dimensions are 1x5, 5x4, 4x3, 3x2, and 2x1 respectively. Going through the algorithm OSM_MCM() the optimal scalar multiplications can be put in the table m shown in Table II using initial arrangement of m in Table I(b). Subsequent solution table S is constructed as shown in Table III. The value at m[1][7] cell in Table II incurs cost of computing the chain. As a sample, few partial solutions are derived below.

P = <P0, P1, P2, P3, P4, P5> = <1, 5, 4, 3, 2, 1>.
Exploiting equation (2),

m[1][1] = m[2][2] = …m[5][5] = 0.

m[1][2] = m[1][1]+m[2][2]+P0P1P2 = 0+0+1x5x4 = 20
S[1][2] = k = 1

Similarly,
m[2][3] = P1P2P3 =5x4x3 = 60
S[2][3] = k = 2

m[3][4] = P2P3P4 =4x3x2 = 24
S[3][4] = k = 3

m[4][5] = P3P4P5 =3x2x1 = 6
S[4][5] = k = 4

$$m[1][5] = \min \begin{cases} m[1][1] + m[2][3] + P_0P_1P_3 \\ m[1][2] + m[3][3] + P_0P_2P_3 \end{cases}$$
$$= \min \begin{cases} 0 + 60 + 1 \times 5 \times 3 \\ 20 + 0 + 1 \times 4 \times 3 \end{cases}$$
$$= 32$$
S[1][3] = k = 2

Proceeding in the same way,
m[2][4] = 64
S[2][4] = 2

m[3][5] = 18
S[3][5] = 3

m[1][4] = 38
S[1][4] = 3

m[2][5] = 38
S[2][5] = 2

$$m[1][5] = \min \begin{cases} m[1][1] + m[2][5] + P_0P_1P_5 \\ m[1][2] + m[3][5] + P_0P_2P_5 \\ m[1][3] + m[4][5] + P_0P_3P_5 \\ m[1][4] + m[5][5] + P_0P_4P_5 \end{cases}$$
$$= 40$$
S[1][5] = 4

Thus, the tables' m and S can be represented as in Table II and Table III respectively.

Taking advantage of S, the TT_MCM() procedure recursively builds the preferred 2-tree similar to fig. 1 and it is shown in fig. 2.

Undergoing OP_MC() algorithm on the above tree in Fig. 2, the chain can be parenthesized as close to Fig. 4 in consequence. All the leaves are first replaced by the matrices shown in Fig. 3, and Fig. 4(d) gives an idea about optimal parenthesized chain.

TABLE II: Matrix M For Scalar Multiplications.

i \ j	1	2	3	4	5
1	0	20	32	38	40
2		0	60	64	38
3			0	24	18
4				0	6
5					0

TABLE III: Solution Matrix S for 2-Tree Generation.

i \ j	2	3	4	5
1	2	3	4	1
2	2	2	2	2
3		3	3	3
4			4	4

V. Comparision Study

So far we have demonstrated an algorithm that computes the optimal cost for multiplying a chain of matrices. It is not hard to modify the algorithm so that it will also compute the actual order of matrix multiplications [4]. The Table IV summarizes the main result.

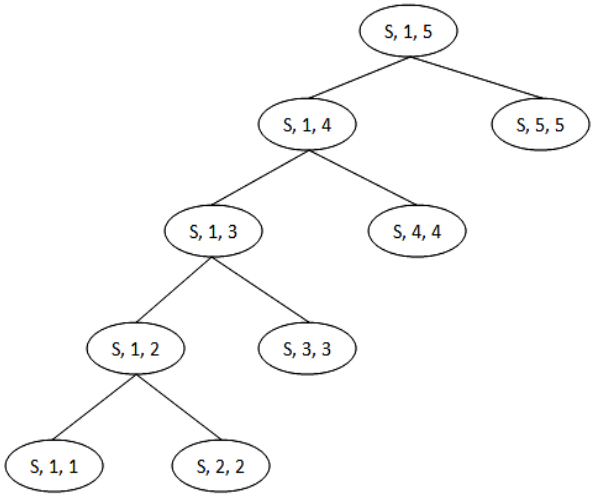


Fig. 2: 2-Tree for Optimal Parenthesization.

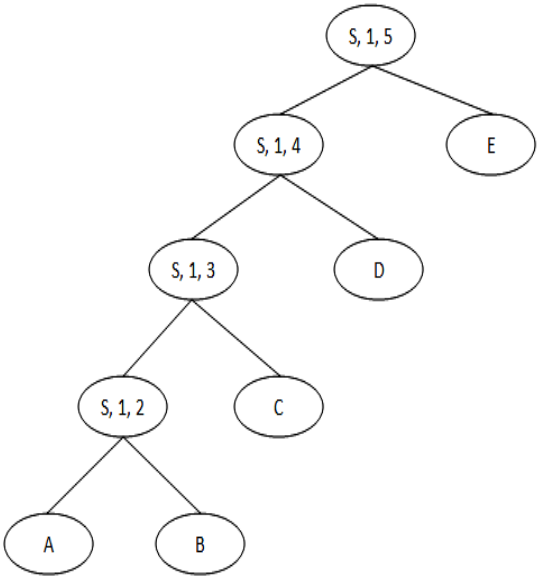
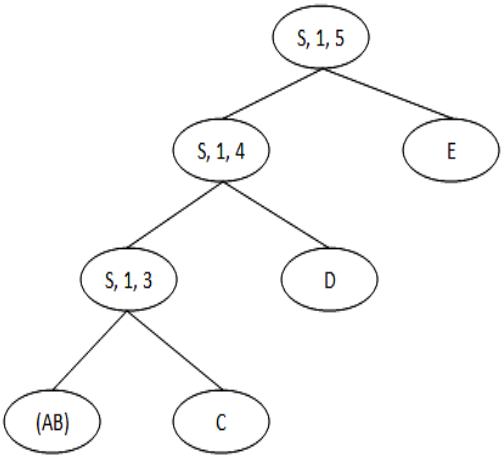
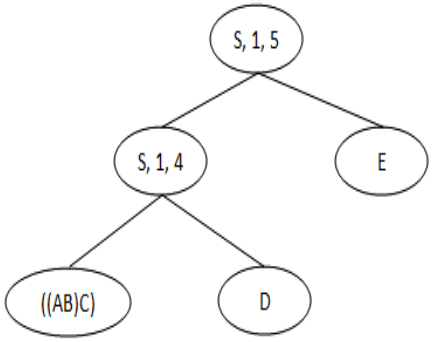


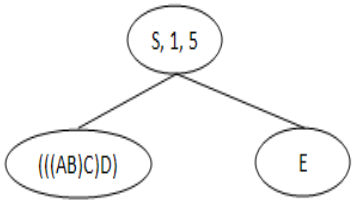
Fig. 3: Replacement of Leaves Nodes of 2-tree by Matrices.



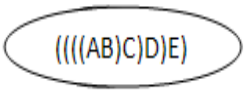
(a)



(b)



(c)



(d)

Fig. 4: Insertion of Parenthesis in the Chain

TABLE IV: COMPARING COMPLEXITIES FOR DIFFERENT APPROACHES.

	Traditional Approach	***Classical DP Approach***	***Proposed Model***
Time Complexity	$O(n^3)$	$\Theta(n^3)$	$\Theta(n^3)$
Space Complexity	$O(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

Although complexities are seemed to be alike still the proposed approach is better one at least in four respects. First, a complexity represents asymptotic behavior of the proposed algorithm. Upon computation exact numerical complexity value would be lower. Second the approach conveys us well again towards simple understanding of the problem. Third the table m can be worked out in easy way. Finally the model suggested has introduced 2-tree that makes a sense how contemporary two matrices can be parenthesized in a chain. Thus in general, proposed model shows simple method to solve a chain of matrices. Surprisingly, this problem could be solved in *O(nlogn)* time by Hu –Shing's suggested model [4].

VI. CONCLUSION

Matrix Chain Multiplication problem involves the question how the optimal sequence for performing a series of operations can be determined. This general class of problem

is not only important in compiler design for code optimization and in databases for query optimization but also in task scheduling and allied applications. The model in fact does not carry out any matrix multiplication rather it shows order of sequence how to multiply. The main contribution to this work is the formalization of the proposed algorithm which significantly enhances performances improvement reducing handful arithmetic operations to the number of matrices and the sequence of dimensions. At the same time the practice might turn out to be very straightforward for the mathematic enthusiasts.

ACKNOWLEDGEMENT

The author would like to convey his heartily gratitude to his beloved Madhurima Mondal, Amei Rai, and their friends, B.Tech, 3rd Year Students, Department of Computer Science & Engineering, Bengal College of Engineering and Technology, Durgapur for providing notes, existing solution illustrated in Section IV, valuable suggestions, graceful inspiration towards completion of this draft in time.

REFERENCES

[1] Nai-Kuan Tsao, "Error Complexity Analysis of Algorithms for Matrix Multiplication and Matrix Chain Product", IEEE Transactions on Computers, Vol. C-30, No. 10, October 1981.
[2] Thomas H. Coremen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms", PHI, 2nd Edn, 2008.
[3] Ellis Horowitz, Sartaj Sahani, Sanguthevar Rajasekaran, "Computer Algorithms", University Press, 2nd Edn, 2008.
[4] M.H. Alsuwaiyel , "Algorithms Design Techniques and Analysis", PHEI, 2002.
[5] David B. Wagner, "Dynamic Programming", The Mathematica Journal, Miller Freeman Publications, 1995.
[6] Phillip G. Bradfordy, Gregory J. E. Rawlinsz, Gregory E. Shannonx, "Efficient Matrix Chain Ordering in Polylog Time", Journal of Computer, SIAM, Vol. 27, No. 2, Pp. 466 - 490, April 1998.
[7] Heejo Lee, Jong Kim, Sung Je Hong, Sunggu Lee, "Processor Allocation And Task Scheduling Of Matrix Chain Products On Parallel Systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 4, April 2003.
[8] Marco Bodrato, "A Strassen-Like Matrix Multiplication Suited for Squaring And Highest Power Computation", CIVV, 2008.
[9] Muhammad Hafeez, Muhammad Younus, "An Effective Solution for Matrix Parenthesization Problem through Parallelization", International Journal of Computers, Issue 1, Vol. 1, 2007.
[10] T. C. Hu, M. T. Shirig, "Computation of Matrix Chain Products", Stanford University, September 1981.
[11] Cary Cherng, Richard E. Ladner, "Cache Efficient Simple Dynamic Programming", ICAA, France, 2005.
[12] Biswajit Bhowmik, "Design and Analysis of Algorithms", S. K. Kataria and Sons, 1st Edition, 2011.
[13] Ting Wang, "Algorithms for Parallel and Sequential Matrix-Chain Product Problem", Ohio University, November, 1997.
[14] Biswajit Bhowmik, "Dynamic Programming – Its Principles, Applications, Strengths, and Limitations", International Journal of Engineering Science and Technology, Vol. 2(9), 2010, Pages: 4822–4826.
[15] Yanhong A. Liu, Scott D. Stoller, "Dynamic Programming via Static Incrementalization", ESOP, 1999.
[16] A. R. Vasistha, "Matrices", Krishna Publications.
[17] D. Samanta, "Classic Data Structures", PHI, 18th printing, 2010.

Biswajit Bhowmik is currently doing research work as a Research Scholar in the Department of Computer Science & Engineering of Indian Institute of Technology Guwahati, India. Before assuming his present designation he was very renowned faculty member with Bengal College of Engineering and Technology, Durgapur, India as Assistant Professor in the Department of Computer Science & Engineering last seven years. He is a member of different professional bodies such as IEEE, IACSIT, IAENG, IAS, IAOE, ISOC, PASS, UACEE etc. He is also a member of some leading professional societies such as IEEE Computer Society, IEEE Communications Societies, IAENG Society of Computer Science, IAENG Society of Wireless Networks, IAENG Society of Software Engineering, and IAENG Society of Artificial Intelligence. He is reviewer of several international journals such as ETASR, IJCSIC, IJCSIS, JACSM, IJoAT, JWMC etc in the area of computer science. He has authored a book titled *Design and Analysis of Algorithms*. He has many publications in international journals including international conference proceedings on the subjects ranging from Algorithms Analysis, Graph Theory, Compiler Design, and Mobile Computing. In addition his area of interests includes Data Structures & Algorithms, Software Engineering, Computational Geometry, and Green Computing. He has guided several projects at under graduate level. This paper is based on Dynamic Programming approach as easy solution tool towards optimal solution of Chain Matrix Multiplication Problem and is for all the readers who feel difficulty in understanding classical solution procedure of the problem.