Enhanced Matrix Chain Multiplication

B. Suvarna* and T. Maruthi Padmaja

Vignan's Foundation for Science Technology and Research, AP, India
E-mail: sv1720@gmail.com; padmaja.tu2002@gmail.com
**Corresponding Author*

Received 24 January 2018; Accepted 28 July 2018;
Publication 17 September 2018

Abstract

Let A_1, A_2,....An be the given sequence of n matrices, generally matrix chain multiplication algorithm is used to obtain its-product with minimum cost(lowest cost). However the matrix chain multiplication is a dynamic programming paradigm and takes $O(n^3)$ computational complexity. Here we present improved algorithm for matrix chain multiplication with minimum space and time complexities. The viability of this new algorithm is demonstrated using few examples and the performance is computationally verified. This algorithm does not take $O(n^3)$ if any two of the S values are not same and $O(n^3)$ when the two values of S are same in the worst case.

Keywords: Matrix multiplication, Matrix Chain Multiplication, Dynamic Programming, Optimal ordering of matrices.

1 Introduction

Matrix Chain Multiplication is one of the optimization problem which is widely used in graph algorithms, signal processing and network industry [1–4]. We can have several ways to multiply the given number of matrices because the matrix multiplication is associative. In other words, there is no matter how the matrices are parenthesize to perform the product. Because the result

Journal of Cyber Security and Mobility, Vol. 7_4, 409–420. *River Publishers*
doi: 10.13052/jcsm2245-1439.743

will be the same. For example, if we had four matrices A_1, A_2, A_3, A_4 with dimensions $A_1 = P_0 \times P_1$, $A_2 = P_1 \times P_2$, $A_3 = P_2 \times P_3$, $A_4 = P_3 \times P_4$ then we would have the parenthesize of the above arrays, along with multiplication number are listed below.

Ex: $A_1 = 2 \times 3$, $A_2 = 3 \times 4$, $A_3 = 4 \times 5$ and $A_4 = 5 \times 2$

$(A_1A_2)(A_3A_4) = P_0P_1P_2 + P_2P_3P_4 + P_0P_2P_4 = 80$ multiplications

$((A_1A_2)A_3)A_4 = P_0P_1P_2 + P_0P_2P_3 + P_0P_3P_4 = 84$ multiplications

$(A_1(A_2A_3))A_4 = P_0P_1P_3 + P_1P_2P_3 + P_0P_3P_4 = 110$ multiplications

$A_1((A_2A_3)A_4) = P_1P_2P_3 + P_1P_3P_4 + P_0P_1P_4 = 102$ multiplications

$A_1(A_2(A_3A_4)) = P_1P_2P_3 + P_1P_3P_4 + P_0P_1P_4 = 76$ multiplications

From the above observation, we can say that the last one is taking minimum number of multiplications. But comparison of each and every possible parenthesize like brute force approach would take exponential time depending on the number of matrices. This process is not suitable to implement where there is more number of matrices to multiply. A fastest possible solution to this matrix chain multiplication can be achieved by dividing the problem into a set of sub problems. By this the time complexity will be reduced by solving sub problems once and reusing the solutions as in the design technique of dynamic programming [5].

Example
Let n = 4, where n is the number of matrices has different dimensions $P_{i-1} \times P_i$

$A_1 = 3 \times 4$ $A_2 = 4 \times 6$ $A_3 = 6 \times 7$ $A_4 = 7 \times 2$ then find the sequences of matrices to multiply

P(0:n)=(3,4,6,7,2) is the single dimensional matrix

The values of M and S for the first row are calculated by using the Equations (1) and (2). The remaining values of M and S can be calculated by using the following logic.

```
START
        for w:=2 to n do
        for x:=1 to n-w+1 do
                set y=x+w-1;
                M[x,y]=infinity;
                for z=x to y-1 do
                        If M[x,z]+M[z+1,y]+p[x-1]*p[z]*p[y] is less than
                           M[x][y] then
                        M[x][y]= M[x,z]+M[z+1,y]+p[x-1]*p[z]*p[y];
                        S[x][y]=z;
                [End of for]
                set M[x,y]= M[x,z]+M[z+1,j]+P[x-1]*P[z]*P[y];
                set S[x,y]=z;
                [END OF FOR]
        [END OF FOR]
Write (M[1,n],S[1,n]);
END
```

Table 1 Calculation of M and S values for the given example

M_{11}=0 S_{11}=0	M_{22}=0 S_{22}=0	M_{33}=0 S_{33}=0	M_{44}=0 S_{44}=0
M_{12}=72 S_{12}=1	M_{23}=168 S_{23}=2	M_{34}=84 S_{34}=3	
M_{13}=198 S_{13}=2	M_{24}=132 S_{24}=2		
M_{14}=156 S_{14}=2			

Where M[x, y] is the minimum number of multiplications required by each pair and S[x, y] be the value of contains the minimum z which is used for where to place parenthesis to split the product A_x, A_{x+1},.... A_y for $1 \leq x \leq y \leq n$.

The above example shows how the equation can be used to determine M's and S's. Now let us examine the time requirement of the above procedure to construct M's and S's to calculate M(x,y) for y-x = 1,2,. . . n in that order when y-x = w there are n-w+1 M(x,y)'s to compute. The total time for all M(x,y) with y-x = m is therefore O(nm-m^2). And the total time to compute M(x,y) and S(x,y) is $\sum_{1 \leq m \leq n}(nw - w^2) = O(n^3)$ [6].

However we can do better than this for finding optimal z (used for parenthesize) by limiting search from the range of S(x,y-1) to S(x+1,y) proposed by D.E Knuth instead of repeating the z from x to y-1 which is nearly equal to n.

2 Existing Algorithm

Existing algorithm for matrix chain multiplication is presenting here.

```
Algorithm MCM(P) BEGIN
        //n is number of matrices //let m[1..n, 1..n] and s[1..n, 2..n]
              be new tables.
          for x: =1 to n do
          SET M[x,x]: =S[x,x]:=0;
          SET M[x][x+1]=M[x][x]+M[x+1][x+1]+P[x-1]*P[x]*P[x+1];
          SET S[x][x+1]=x;
          [END OF FOR]
          for w:= 3 to n do
                  for x:=1 to n-w+1 do
                     set y=x+w-1;
                     set z=find(x,y);
                     set M[x,y]=M[x,z]M[z+1,j]+P[x-1]*P[z]*P[y];
                     set S[x,y]=z;
                     [END OF FOR]
                  [END OF FOR]
Write (m[1,n],s[1,n]);
END

Algorithm find_of_z(x,y) BEGIN
        SET min: ∞;
        If S[x,y-1] is less than S[x+1,y] then
              for z:=S[x,y-1] to S[x+1,y] do
              if ((M[x,z]+M[z+1,y]+P[x-1]*P[z]*P[y])<min)
                    min=M[x,z]+M[z+1,y]+P[x-1]*P[z]*P[y];
                    1=z;
              [END IF]
              Else
                    for z=x toy-1 do
                    if((M[x,z]+M[z+1,y]+P[x-1]*P[z]*P[y]])<min)
                       min=M[x,z]+M[z+1,y]+P[x-1]*P[z]*P[y];
```

```
                    1=z;
                END OF IF
            END OF ELSE
        return 1;
END
```

In the existing algorithm given for finding a sequence and finding the less number of multiplications takes $O(n^3)$ because the algorithm is composed with 3 loops and each loop takes at most n-1 times [7]. For that we proposed an algorithm for finding the sequence of matrices to multiply with the less time.

3 Proposed Algorithm

Time complexity is reduced by limiting the search for z from S(x, y-1) to S(x+1,y). However the formulas for computing the rows of the table are described below.
The values for M's and S's of first row can be computed by the formula

$$\text{Set } M[x, x] \text{ to } 0, \text{ where } x \text{ is from } 1 \text{ to } n \tag{1}$$

$$\text{Set } S[x, x] \text{ to } 0, \text{ where } x \text{ is from } 1\text{to } n \tag{2}$$

And for to compute the values for M's and S's of second row, the formulas are given below.

$$M[x][x+1]=M[x][x]+M[x+1][x+1]+P[x-1]*p[x]*p[x+1] \text{ where } y\text{-}x=1,2,\ldots n \tag{3}$$

S[x][x+1]=x; where y-x = 1,2,. . . n
And for to compute the remaining rows below formula can be used
M[x, y]= M[x, z]+M[z+1,y]+P[x-1]*P[z]*P[y];
S[x, y]=z;
Following are the complete algorithms for finding the M's and S values

```
Algorithm Matrix_Chain_Multiplication (P) BEGIN

    //n is number of matrices //let m[1..n,1..n] and s[1..n,2..n] be new tables.
    for x: =1 to n do
    SET   M[x,x]:=S[x,x]:=0;
    SET   M[x][x+1]=M[x][x]+M[x+1][x+1]+P[x-1]*P[x]*P[x+1];
```

```
    SET   S[x][x+1]=x;
    [END OF FOR]
    for w:= 3 to n do
        for x:=1 to n-w+1 do
            set y=x+w-1;
            set z=find_of_z(x,y);
            set M[x,y]= M[x,z]+M[z+1,y]+P[x-1]*P[z]*P[y];
            set S[x,y]=z;
            [END OF FOR]
        [END OF FOR]
Write (M[1,n],S[1,n]);
END

Algorithm find_ of_ z(x, y) BEGIN

        SET min:= ∞;
        If S[x,y-1] is less than S[x+1,y] then
          for z:=S[x,y-1] to S[x+1,y] do
          if(( M[x, z]+M[z+1,y]+P[x-1]*P[z]*P[y])<min)
              min= M[x, z]+M[z+1,y]+P[x-1]*P[z]*P[y];
              1=z;
          [END IF]
          Else
              for z=x toy-1 do
              if((M[x, z]+M[z+1,y]+P[x-1]*P[z]*P[y]])<min)
                  min=M[x, z]+M[z+1,y]+P[x-1]*P[z]*P[y];
                  1=z;
              END OF IF
          END OF ELSE
        return 1;
END
```

From the above algorithm the find_ of_ z function is used to calculate the minimum value for z, The z value is getting by repeating the loop from S[x,y-1] to S[x+1,y] instead of repeating the z from x to y-1 which is taking at most n-1 times.

Hence the above find_ of_ z function is again modified "if S[x,y-1] is less than S[x+1,y] then the z will repeat S[x,y-1] to S[x+1,y] else it will repeat from x to y-1.

This type of z which repeats from x to y-1 may or may not be occurred for all the type of inputs. This can be observed in Tables 2 and 3.

4 Results

This section demonstrates the proposed approach by considering example. The above proposed algorithm is implemented and tested for the following input. The experimental result values of M and S are given in the form of table.

Example 1: n=5,(P_0,P_1,P_2,P_3,P_4,P_5)=(3,4,6,2,3,7)

Table 2 Results for M values

M values				
0	0	0	0	0
72	48	36	42	
72	72	126		
90	146			
153				

Table 3 Results for S values

S values				
0	0	0	0	0
1	2	3	4	
1	3	3		
3	3			
2				

Output sequence is (((A1(A2A3))A4)A5)

Example 2: n=4,(P_0,P_1,P_2,P_3,P_4)=(3,4,6,7,2)

Table 4 Results for M values

M values			
0	0	0	0
72	168	84	
198	132		
156			

Table 5 Results for S values

S values			
0	0	0	0
1	2	3	
2	2		
2			

Output sequence is (A1(A2(A3A4)))

Example 3: n=4,(P_0,P_1,P_2,P_3,P_4)=(2,3,6,4,5)

Table 6 Results for M values

M values			
0	0	0	0
36	72	120	
84	132		
124			

Table 7 Results for S values

S values			
0	0	0	0
1	2	3	
2	3		
2			

Sequence is (((A1A2) A3) A4)

From the above Table 3 the values of S_{14} and S_{25} and also $S_{13,}$ S_{24} in the Table 4 which are marked as red have same cut so at this case only it is running three nested loops; otherwise it does not run three nested loops. However the corresponding S values are marked with red color (in the tables) at which the algorithm is running three nested loops. Otherwise it will not run three nested loops. From this observation we can claim that this Matrix_Chain_Multiplication algorithm will not take $O(n^3)$ time.

By observing the above tables we can say that the algorithm for finding minimum split i.e. z will take only from S[x, y-1] and S[x+1, y] instead of repeating for n-1 times.

This algorithm does not take $O(n^3)$ if any two of the S values are not same and $O(n^3)$ when the two values of S are same in the worst case.

5 Conclusion

From the above discussion we can say that the proposed matrix chain multiplication algorithm using Dynamic Programming in the best case and average case takes $O(n^2)$ time complexity which is less when it is compared with existing matrix chain multiplication which takes $O(n^3)$. Hence the time complexity is reduced with the space requirement of $O(n^2)$.

References

[1] Shyamala, K., Raj Kiran, K., and Rajeshwari, D. (2017). Design and implementation of GPU-based matrix chain multiplication using C++ AMP. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, (pp. 1–6). IEEE.

[2] Mabrouk, B. B., Hasni, H. and Mahjoub, Z. (2014). Performance evaluation of a parallel dynamic programming algorithm for solving the matrix chain product problem. In *2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA)*, (pp. 109–116). IEEE.

[3] Cáceres, E. N., Mongelli, H., Loureiro, L., Nishibe, C. and Song, S.W. (2009). A parallel chain matrix product algorithm on the InteGrade grid. In *International Conference on High Performance Computing, Grid and e-Science in Asia Pacific Region* (pp. 304–311).

[4] Lakhotia, R., Kumar, S., Sood, R., Singh, H. and Nabi, J. (2015). Matrix-Chain Multiplication Using Greedy and Divide-Conquer approach. *International Journal of Computer Trends and Technology (IJCTT)*, 23(2):65–72. Doi: 10.14445/22312803/IJCTT-V23P115.

[5] Godbole, S. S. (1973). On efficient computation of matrix chain products. *IEEE Transactions on Computers*, 100(9):864–866.

[6] Horowitz, E., Sahni, S. and Rajasekaran, S. (2006). *Fundamentals of Computer Algorithms*, 2nd Edition, Galgotia publications.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). *Introduction to algorithms.* Third Edition, MIT press.

[8] Nishida, K., Ito, Y. and Nakano, K. (2011). Accelerating the dynamic programming for the matrix chain product on the GPU. *In 2011 Second International Conference on Networking and Computing* (pp. 320–326). IEEE.

[9] Parvaresh, F. and Vardy, A. (2004). Polynomial matrix-chain interpolation in Sudan-type Reed-Solomon decoders. *International Symposium on Information Theory, 2004. ISIT 2004. IEEE. Proceedings.* Doi: 10.1109/ISIT.2004.1365424.

[10] Lee, H., Kim, J., Hong, S. J. and Lee, S. (2003). Processor allocation and task scheduling of matrix chain products on parallel systems. *IEEE Transactions on Parallel & Distributed Systems*, (4):394–407.

[11] Myung, J. and Lee, S. G. (2012). Matrix chain multiplication via multi-way join algorithms in MapReduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication* (p. 53). ACM.

Biographies

B. Suvarna is pursuing Ph.D in VFSTR deemed to be University. Prior to that she has received M.Tech degree from vignan's Engineering college affiliated to JNTU Kakinada and B.Tech degree from VR Siddhartha Engineering college affiliated to ANU. Currently she is working as an Asst Professor, CSE Department, VFSTR deemed to be University, AP, INDIA. Her research interests include Data Mining, Machine Learning and analysis of algorithms.

T. Maruthi Padmaja, received Ph.D degree from the University of Hyderabad, Hyderabad. During that time she was a research scholar at IDRBT, Hyderabad. Prior to that she has received M.Tech degree from the Tezpur University, Assam. Currently, she is working as an Associate Professor, CSE Department, VFSTR University, AP. Her research interests include Data Mining, Machine Learning and Pattern Recognition.