Matrix-Chain Multiplication Using Greedy and Divide-Conquer approach

Raghav Lakhotia[#1], Sanjeev Kumar[#2], Rishabh Sood[#3], Harmeet Singh[#4], Javaid Nabi[#5]
[#1,#3,#4,#5]Samsung Research Institute, Bangalore, India
[#2]Amazon, India
[#2]Amazon India
[#3, #4, #5]Samsung Research Institute, Bangalore, India

***Abstract*— Matrix Chain Multiplication is a famous application of optimization problem and is used widely in signal processing and network industry for routing. The crux of the solution lies in minimizing the cost or minimizing the number of arithmetic operations required to multiply out the matrices. A top-down dynamic approach is a well-known solution for this problem which helps to determine the minimal cost required to perform the required multiplication of the matrices. The dynamic solution bears time complexity of the order (n^3).The authors in this paper present a greedy approach to find the optimal computation order of matrix chain multiplication. This approach provides the minimum cost required to compute the required result in a runtime of the order $(n \log n)$as compared to the dynamic approach runtime of (n^3). Although the end result i.e. the matrix obtained after the chain multiplication provided by theapproaches, the proposed approach and the dynamic approach is the same.**

Keywords: Matrix Chain Multiplication, Dynamic Approach, Greedy Approach.

I. INTRODUCTION

In few past decades, many algorithms have been proposed for matrix chain multiplication[4][10]. Some of them are acceptable and have been used in many applications or programs. Majority of these algorithms use dynamic approach which recursively evaluates the time required to compute the minimum cost needed to multiply a specific sequence and save it. This dynamic approach is a powerful tool for algorithms and is used in many other problems like knapsack problem, travelling salesman problem etc. In this paper an alternate approach has been described to calculate the minimum cost required to compute the chain multiplication using greedy approach. The proposed method is named as Greedy MCM.

The paper is organized in following sections. First a brief introduction is given about what is matrix chain multiplication problem and how to calculate the cost of multiplying the chain matrices. Second, the traditional approach using dynamic solution is discussed. The next three sections define the proposed algorithm G_MCM and explain the working using some examples.The paper ends with some tables and graphs displaying computational results of the proposed method and comparing those with dynamic approach.

II. OVERVIEW OF MATRIX CHAIN MULTIPLICATION PROBLEM

Consider the problem of evaluating the product of *n* matrices

$$M = M_1 * M_2 * \ldots\ldots\ldots\ldots * M_n$$

whereM_i is a matrix of the order mxn and $M_{(i+1)}$ is a matrix of order nxp. The product $N = M_i * M_{(i+1)}$ is a mxp matrix. This N can be computed in time O(mnp). For example, let there be 4 matrices named A, B, C, D of the order (2x3), (3x4), (4x5), (5x6) respectively. Now

$$M = A*B*C*D$$

Since matrix multiplication is associative, the order in which the above chain multiplication is evaluated does not affect the final result. The matrix can be multiplied in the following orders: ((AB)C)D, (AB)(CD), A((BC)D), (A(BC))D, A(B(CD)).

The problem is not actually to perform the multiplication, but to decide the order in which multiplications needs to be performed. Because this order in which the product of matrices is parenthesized affects the number of simple arithmetic operations needed to compute the product, or the efficiency [8]. Let us evaluate the number of arithmetic operations performed for all the above mentioned parenthesizations –

((AB)C)D = 2x3x4 + 2x4x5 + 2x5x6 = 124
(AB)(CD) = 2x3x4 + 4x5x6 + 2x4x6 = 192
A((BC)D) = 3x4x5 + 3x5x6 + 2x3x6 = 186
(A(BC))D = 3x4x5 + 2x3x5 + 2x5x6 = 150
A(B(CD)) = 4x5x6 + 3x4x6 + 2x3x6 = 228

Clearly the first method is more efficient in all. This gives a picture as the number of operations termed as scalar multiplications is affected by the order in which the product is computed. Thus, the number of scalar operations required depends on optimal parenthesis order. Now in order to determine the optimal parenthesis order, we can proceed in many ways. One is brute force method where we calculate the number of operations of all the possible parenthesis order and find the least amongst them. But it will be impractical and time consuming as the time complexity of such approach will be (2^n). The traditional approach i.e dynamic solution [1][2][3], reduces this time of (2^n)further to (n^3)which looks decent and of practical use.

III. DYNAMIC APPROACH

As described above, the cost of multiplying a chain of n matrices $M_1M_2....M_n$ depends on the order in which the n−1 multiplications are carried out. Here we will discuss in brief about the dynamic approach for the matrix chain multiplication problem as described in [1][7].

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 f(n) be the number of ways to fully parenthesize a product of n matrices. Suppose we want to perform the multiplication

$$(M_1M_2 ...M_k) \times (M_{k+1}M_{k+2} ...M_n).$$

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)[1].

In dynamic approach, the cost of multiplying two matrices together is done in a recursive way which can be represented as –

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

The algorithm for the same using dynamic approach can be found in [1]. Thus, time and space complexities of the algorithm are easy to calculate. For constant c > 0, the running time T(n) of algorithm is proportional to:

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

Similarly space complexity of the algorithm is:

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

IV. PROPOSED ALGORITHM USING GREEDY APPROACH

This section presents a solution for the problem to determine the minimum number of scalar multiplications performed for the matrix chain multiplication problem using greedy approach. To obtain optimal solution, we modify the greedy approach in combination with divide and conquer strategy and get an algorithm which can calculate the multiplication order in $(n \log n)$time which is very fast as compared to dynamic programming solution. In the proposed algorithm the main idea is to solve the problems in a top down fashion (divide and conquer).

ALGORITHM: G_MCM(x, y)

Description: G_MCM means Greedy approach for matrix chain multiplication(MCM)

Input: p = array containing the matrices dimensions.
x, y = starting and ending index.

Output: Parenthesized representation of the product of n matrices.

G_MCM(x, y)

```
(1)key=arb          //where p[arb] is infinity
(2)if x== (y-1)
(3)        Print "A"+char(y)
(4)else if x== (y-2)
(5)        Print "("+"A"+char(x+1)+"A"+char(y+1)+")"
(6)else
(7)        fori = x+1 down to y-1
(8)                do if p[key]>p[i]
(9)                        key=i
(10)       if p[x]!=p[y]
(11)               do if p[x] < p[key]
(12)                       key=x
(13)       if p[y]<p[key]
(14)               key=y
(15)       if key==x
(16)Print"("+G_MCM(x,y-1)+G_MCM(y-,y)+")"
(17)       else if key==y
(18)Print"("+G_MCM(x,x+1)+ G_MCM(x+1,y)+")"
(19)       else
(20)Print"("+G_MCM(x,key)+G_MCM(key,y)+")"
```

The arguments x and y denotes the two indices of the order array **p[0…..n].** The feature of the algorithm is to divide the array p into n sub-array, each containing at least 1 or at most 2 elements. This process of division is done in a greedy manner. At each step only least value is selected among all elements of in the array p[x, y], so that the multiplication cost is kept minimum at each step. This greedy approach ensures that the solution is optimal with least cost involved and the output is a fully parenthesized product of matrices. With every division, we put parenthesis around the matrices.

A. Complexity Analysis

In the above algorithm, we are dividing the problem into a number of sub problems recursively, each sub problem being of size n/b. This can be visualized as building a call tree with each node of the tree as an instance of one recursive call and its child nodes being instances of subsequent calls. In the above algorithm, each node would have a number of child nodes. Each node does an amount of work that corresponds to the size of the sub problem n passed to that instance of the recursive call and given by f(n). The total amount of work done by the entire tree is the sum of the work performed by all

the nodes in the tree. This recursive relation is presented by Master Theorem [5] –

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \text{ where } a > 1, b > 1 \quad (5)$$

The constants denotes –
n is the size of the problem.
a is the number of sub problems in the recursion.
n/b is the size of each sub problem.
f(n) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the sub problems.

There are following three cases –
1. If f(n) = $(n^c)$where c <$\log_b a$ then T(n) =$\Theta\left(n^{\log_b a}\right)$
2. If f(n)=$(n^c)$where c = $\log_b a$ then T(n) = $(n^c \log n)$
3. If f(n) = $(n^c)$where c >$\log_b a$ then T(n) = $(f(n))$

1) Best Case –

In best case, each recursive call will divide the array in two equal parts. The recurrence relation for this case can be given as –

$$T(n) = 2T(n/2) + f(n) \quad (6)$$

The above algorithm falls in case 2 as f(n) i.e. the cost of dividing the array into sub-arrays is done in linear fashion and hence c = 1. This implies $\log_b a$ is also 1. Hence the solution is $(n \log n)$.

2) Average Case –

In average case, each recursive call will divide the array into (N-I-1) and I parts where Iranges from 1 to N-2. We note that average number of comparisons to find the min keyis N-2.Thus we take the average of all the number of comparisons done over possiblecombinations of the input sequence. This can be estimated accurately by solving the recurrence relation

$$C(n) = n - 2 + 1/n \sum_{i=1}^{n-2} \{C(i) + C(n-i-1)\} \quad (7)$$

On solving the above recursive relation we get $C(n) \approx 1.39n \log_2 n$. Hence the average time complexity of the proposed G_MCM algorithm is $(n \log n)$.

3) Worst Case –

In worst case, each recursive call divides the array into only (N-1) and 1 part. The recurrence relation for this case can be given as –

$$T(n) = T(n-2) + T(1) + f(n)$$
$$= T(n-2) + f(n) \quad (8)$$

where f(n) is the cost of the work done outside the recursive calls, which includes the cost of finding the minimum key and dividing the problem. Using equation (5) of Master's Theorem, time complexity for worst case is (n^2).
The multiplication order of the matrices can be denoted by recursive approach presented as

$$G\_MCM(x,y) = \begin{cases} A(y), & x == (y-1) \\ A(x+1)A(y+1), & x == (y-2) \\ \begin{cases} G\_MCM(x, key) + \\ G\_MCM(key, y) \end{cases} \end{cases} \quad (9)$$

Where x, y denote the starting and ending index of the array P. The notation A(x) denotes the matrix number i.e. matrix 1 or matrix 2.

V. NUMERICAL RESULT AND ANALYSIS

Consider the chain of five matrices <A_1, A_2, A_3, A_4, A_5>of dimension 20x12, 12x17, 17x5, 5x23, 23x7 respectively.
Now going through G_ MCM, defined in equation (9), the required result can be obtained as per the below steps.

P = <P0, P1, P2, P3, P4, P5> = <20, 12, 17, 5, 23, 7>
Step 1: P[key] = ∞. Key set to 3, hence,
G_MCM (0, 5) = G_MCM (0, 3) + G_MCM (3, 5)
Step 2:G_MCM (0, 3) = G_MCM (0, 1) + G_MCM (1, 3)
Step 3:G_MCM (0, 1) = A1
Step 4:G_MCM (1, 3) = (A2xA3)
Step 5:G_MCM (3, 5) = (A4xA5)
Step 6:G_MCM (0, 5) = ((A1x(A2xA3))x(A4xA5))
On solving the same example using the dynamic approach, the solution would be $((A_1(A_2A_3))(A_4A_5))$. Hence the cost incurred by both the methods is same.

A. Special Case

It has been observed that the above proposed algorithm will work fine when the elements in the array **p** are all distinct and will give the optimal solution. But when the smallest element of array **p** will occur more than once then there are some chances that the above algorithm may fail to give the optimal solution. But it is guaranteed that the matrix chain multiplication cost of the ordering obtained by above algorithm would not exceed the optimal solution's cost by more than 25 percent.

Example –
Consider the chain of five matrices <A1, A2, A3, A4, A5> of dimension 3x4, 4x3, 3x6, 6x3, 3x7 respectively.
Now going through G_ MCM, defined in equation (1), the required result can be obtained as per the below steps.
P = <P0, P1, P2, P3, P4, P5> = <3, 4, 3, 6, 3, 7>
Here, P0, P2 and P4 have same values as 3. On applying the above given algorithm on this array we will get the order of matrix multiplication in the following steps –

Step 1: P[key] = ∞, key set to 2
Hence,G_MCM (0,5)=G_MCM (0,2)+G_MCM (2, 5)

Step 2:G_MCM (0,2) = (A1xA2)
Step 3:G_MCM (2, 5) = G_MCM (2, 4) + G_MCM (4, 5)
Step 4:G_MCM (2, 4) = (A3xA4)
Step 5:G_MCM (4, 5) = A5
Step 6:G_MCM (0, 5) = ((A1xA2)x((A3A4)xA5))
Multiplication Cost = 216

When we solve the same example using the dynamic approach the solution for the same set of matrices would be(((A$_1$A$_2$)(A$_3$A$_4$)) A$_5$)). Multiplication Cost =180
The above proposed algorithm found to be 20% more costly than the optimal solution using dynamic approach in cases when the dimensions of some of the matrices are same. This extra cost is due to the fact that the above algorithm did not chosen the correct least value (here 3) out of its three occurrences. This proposed algorithm can be modified for handling this kind of special cases in future.

B. Analysis of Implementation of G_MCM Algorithm

The results for implementation of G_MCM algorithm foroptimal solution to matrix multiplication problem are shown in Table I, II,III and IV. The number ofmatrices is 1-10 in table I, II, III and 1-18 in table IV. While the sequence of dimensions varies from 1-10, 1-25, 1-100 and 1-400 respectively in table I, II, III and IV. Online applet "Compare Dynamic Programming and Greedy Approach" for matrix chain multiplication problem has been implemented by the authors and can be used to test other results [6].Figures include the graph in two parts. Part 'a' of figure shows the comparison between multiplication cost and number of matrices for Dynamic programming (in blue) and Greedy Algorithm (in red).Part 'b' of figure compares the run time cost and the number of matrices for both the algorithm. In Figure 1, number of matrices and sequence of dimensions varies from 3-10 and 100-200 respectively. In Figure 2,number of matrices and sequence of dimensions variesfrom 3-20 and 200-400 respectively. Input includes numberof matrices and then the dimensions of each matrix. Thecolumn of matrix A must be equal to the row of matrix Bfor all the dimensions.
Analyzing tables I, II, III, IV and graphs, it is evidentthat the multiplication cost for both the algorithms is almost same. But there is considerable amount of runtime cost reduction in case of Greedy MCM algorithm as compared to dynamic MCM approach as the number of matrices and the dimension of matrices increases.

VI. COMPARISON STUDY

So far we have demonstrated an algorithm that computes the optimal cost for multiplying a chain of matrices. Table V summarizes the main results. It denotes that time complexity of proposed algorithm i.e. G_MCM is less compared to Dynamic programming technique. The same is true when we compare space complexities of both methods.

TABLE V- COMPLEXITIES FOR DIFFERENT APPROACHES

	Traditional Approach	**DP Approach**	**Proposed Method**
Time Complexity	(n^3)	(n^3)	(nlogn)
Space Complexity	(n^2)	(n^2)	(n)

VII. CONCLUSION

Matrix Chain Multiplication problem is not actually to perform the multiplications, but merely to decide the order to perform the multiplications. We have presented a different approach to determine an optimal multiplication order for a chain of matrix products. We have shown that, using this approach, an optimal order can be determined in time $(n \log n)$if the matrices are of different orders or can have approximately same solution as optimal solution otherwise. The above described technique is very efficient than existing approaches. It can be applied to any type of matrices successful. Finally, the proposed technique should be viewed more as a complement than as an alternative to existing methods, to be used in all those cases where the other techniques cannot be employed efficiently. The working of the proposed algorithm can be visualized and tested for different inputs through an applet [6] developed by the authors.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, *Introduction To Algorithms*, The MIT Press.

[2] Aho, A.V., Hopcroft, J.E., and Ullman, J.D*The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[3] Chin, F.Y. "*An O(n) algorithm for determining a near optimal computation order of matrix chain product."*Cemmun. ACM 21, 7 July1978, pp. 544-549.

[4] Nicola Santoro, "*Chain Multiplication of Matrices of Approximately or exactly the same size*.", February 1984 Volume 27 Number 2.

[5] Chee Yap, "*A Real Elementary Approach to the MasterRecurrence and Generalizations*.", 8th Annual Conference, TAMC 2011, Tokyo, Japan.

[6] Sanjeev Kumar, RaghavLakhotia, RishabhSood, HarmeetSingh– "*Compare Dynamic Programming and Greedy Matrix Chain Multiplication Algorithm*" , available at url - *http:// mcm-ssnk.rhcloud.com/compareAlgo.html*

[7] Muhammad Hafeez, Muhammad Younus, "*An Effective Solution for Matrix Parenthesization Problem through Parallelization*", International Journal of Computers, Issue 1, Vol. 1, 2007.

[8] T. C. Hu, M. T. Shirig, "*Computation of Matrix ChainProducts*", Stanford University, September 1981.

[9] Deimel, L.E., and Lampe T.A. "*An invariance theorem concerning optimal computation of matrix chain products*." Rep. TR79-14, North Carolina State Univ., Raleigh, NC, 1979.

[10]Sadashiva S. Godbole "An Efficient computation of matrix chain products". IEEE Trans. Comput. C-22, 9 Sept. 1973, 864-866.

***M Cost refers to Multiplication Cost**
***R Cost refers to Runtime Cost**

TABLE I - IMPLEMENTATION OF GREEDY MCM PARENTHESIZATION ALGORITHM, NO. OF MATRICES:1-10, SEQUENCE OF DIMENSIONS:1-10

No. of Matrix	Sequence of Dimensions	Parameters Using Greedy Approach			Parameters Using Dynamic Approach			% Difference (b-a)/b*100
		M Cost*	Parenthesization	R Cost* (a)	M Cost*	Parenthesization	R Cost* (b)	
3	9,8,3,3	297	((A0A1)A2)	4106	288	(A0(A1A2))	9235	55.5
4	2,10,9,6,7	372	(((A0A1)A2)A3)	2155	372	(((A0A1)A2)A3)	4785	54.9
5	9,7,6,2,1,7	180	((A0(A1(A2A3)))A4)	2246	180	((A0(A1(A2A3)))A4)	3205	29.9
6	4,4,6,10,7,6,10	1024	(((((A0A1)A2)A3)A4)A5)	1924	1024	((A0(((A1A2)A3)A4))A5)	3778	49.0
7	5,6,3,5,8,5,8,7	723	((A0A1)((((A2A3)A4)A5)A6))	1523	723	((A0A1)((((A2A3)A4)A5)A6))	4011	62.0
8	4,6,5,9,9,7,2,1,6	281	((A0(A1(A2(A3(A4(A5A6))))))A7)	2462	281	((A0(A1(A2(A3(A4(A5A6))))))A7)	4808	48.7
9	1,5,7,5,8,1,3,4,4,10	189	(((((((A0A1)A2)A3)A4)A5)A6)A7)A8)	2072	186	(((((A0(A1(A2(A3A4))))A5)A6)A7)A8)	7219	71.2

TABLE II - IMPLEMENTATION OF GREEDY MCM PARENTHESIZATION ALGORITHM, NO. OF MATRICES:1-10, SEQUENCE OF DIMENSIONS:1-25

No. of Matrix	Sequence of Dimensions	Parameters Using Greedy Approach			Parameters Using Dynamic Approach			% Difference (b-a)/b*100
		M Cost*	Parenthesization	R Cost* (a)	M Cost*	Parenthesization	R Cost* (b)	
3	16,13,4,9	1408	((A0A1)A2)	4029	1408	((A0A1)A2)	11227	64.1
4	23,21,2,24,21	2940	((A0A1)(A2A3))	1066	2940	((A0A1)(A2A3))	3856	72.3
5	6,12,14,2,12,14	984	((A0(A1A2))(A3A4))	1509	984	((A0(A1A2))(A3A4))	3408	55.7
6	22,16,23,2,3,6,9	1980	((A0(A1A2))((A3A4)A5))	1624	1980	((A0(A1A2))((A3A4)A5))	3224	49.6
7	12,10,12,10,7,5,18,15	4400	((A0(A1(A2(A3A4))))(A5A6))	1940	4400	((A0(A1(A2(A3A4))))(A5A6))	4208	53.8
8	1,8,21,7,20,18,1,8,1	849	(((((((A0A1)A2)A3)A4)A5)A6)A7)	2252	831	((((A0A1)A2)(A3(A4A5)))(A6A7))	4543	50.4
9	2,21,9,25,16,2,14,6,23,24	3296	(((((((A0A1)A2)A3)A4)A5)A6)A7)A8)	3015	3236	(((((A0A1)(A2(A3A4)))(A5A6))A7)A8)	8993	66.4

TABLE III - IMPLEMENTATION OF GREEDY MCM PARENTHESIZATION ALGORITHM, NO. OF MATRICES:1-10, SEQUENCE OF DIMENSIONS:1-100

No. of Matrix	Sequence of Dimensions	Parameters Using Greedy Approach			Parameters Using Dynamic Approach			% Difference (b-a)/b*100
		M Cost*	Parenthesization	R Cost* (a)	M Cost*	Parenthesization	R Cost* (b)	
3	92,34,59,62	318308	(A0(A1A2))	3146	318308	(A0(A1A2))	8068	61.0
4	42,88,100,66,32	611072	(A0(A1(A2A3)))	1634	611072	(A0(A1(A2A3)))	3110	47.4
5	13,22,35,42,77,91	162253	((((A0A1)A2)A3)A4)	2534	162253	((((A0A1)A2)A3)A4)	5587	54.6
6	40,21,72,25,2,45,1	4292	(A0(A1(A2(A3(A4A5)))))	2863	4292	(A0(A1(A2(A3(A4A5)))))	6311	54.6
7	15,24,13,82,62,41, 18,22	122850	((A0A1)((((A2A3)A4)A5)A6))	2856	122850	((A0A1)((((A2A3)A4)A5)A6))	6613	56.8
8	78,81,12,4,71,66,9 7,86,5	110160	((A0(A1A2))((((A3A4)A5)A6)A7))	2691	110160	((A0(A1A2))((((A3A4)A5)A6)A7))	7276	63.0
9	65,66,56,27,44,47, 40,88,65,70	817398	((A0(A1A2))(((((A3A4)A5)A6)A7)A8))	3106	817398	((A0(A1A2))(((((A3A4)A5)A6)A7) A8))	8471	63.3

TABLE IV - IMPLEMENTATION OF GREEDY MCM PARENTHESIZATION ALGORITHM, NO. OF MATRICES:1-18, SEQUENCE OF DIMENSIONS:1-400

No. of Matrix	Sequence of Dimensions	Parameters Using Greedy Approach			Parameters Using Dynamic Approach			% Difference (b-a)/b*100
		M Cost*	Parenthesization	R Cost* (a)	M Cost*	Parenthesization	R Cost* (b)	
3	259,290,67,366	11383568	((A0A1)A2)	4274	11383568	((A0A1)A2)	9060	52.8
6	84,47,145,20,312,1 90,141	2173540	((A0(A1A2))((A3A4)A5))	2300	2173540	((A0(A1A2))((A3A4)A5))	5209	55.8
9	388,308,294,27,33 1,289,23,17,259,15 1	7257878	((A0(A1(A2(A3(A4(A5A6))))))(A7 A8))	3833	7257878	((A0(A1(A2(A3(A4(A5A6))))))(A7A8))	9912	61.3
12	278,303,238,308,2 48,379,189,296,20 9,364,231,366,400	182233422	((A0(A1(A2(A3(A4A5)))))(((((A6A 7)A8)A9)A10)A11))	4114	178661469	((A0(A1(A2(A3(A4A5)))))((((A 6A7)(A8A9))A10)A11))	13838	70.2
15	96,121,339,238,21 3,66,379,176,5,231 ,41,96,33,125,82,4 1	1640175	((A0(A1(A2(A3(A4(A5(A6A7)))))))((((((A8A9)A10)A11)A12)A13)A1 4))	4999	1640175	((A0(A1(A2(A3(A4(A5(A6A7)))))))((((((A8A9)A10)A11)A12) A13)A14))	20631	75.7
18	310,94,219,303,39 1,291,360,351,217, 235,275,240,90,23 4,273,41,46,59,64	39774756	((A0(A1(A2(A3(A4(A5(A6(A7(A8(A9(A10(A11(A12(A13A14))))))))))))))((A15A16)A17))	7104	39774756	((A0(A1(A2(A3(A4(A5(A6(A7(A8(A9(A10(A11(A12(A13A14)))))))))))))((A15A16)A17))	27547	74.2

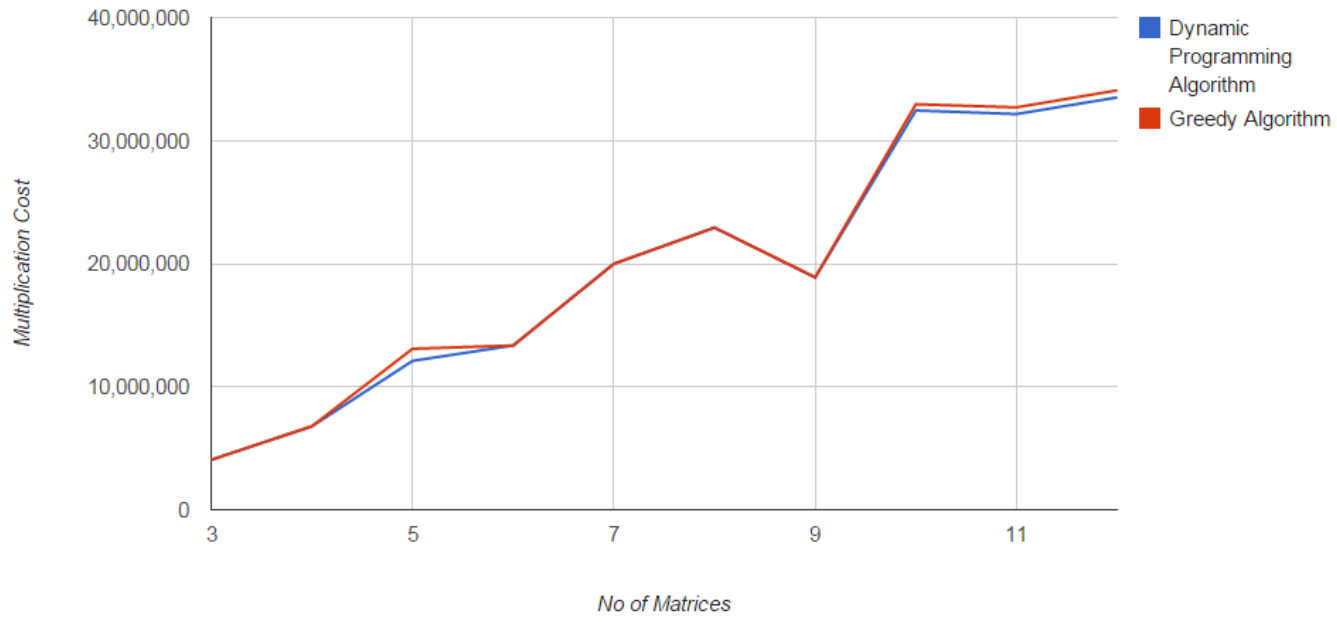

Fig -1a Multiplication Cost v/s Number of matrices comparison between Greedy MCM and Dynamic MCM. Matrix Count: 3-10, Sequence of dimensions: 100-200.

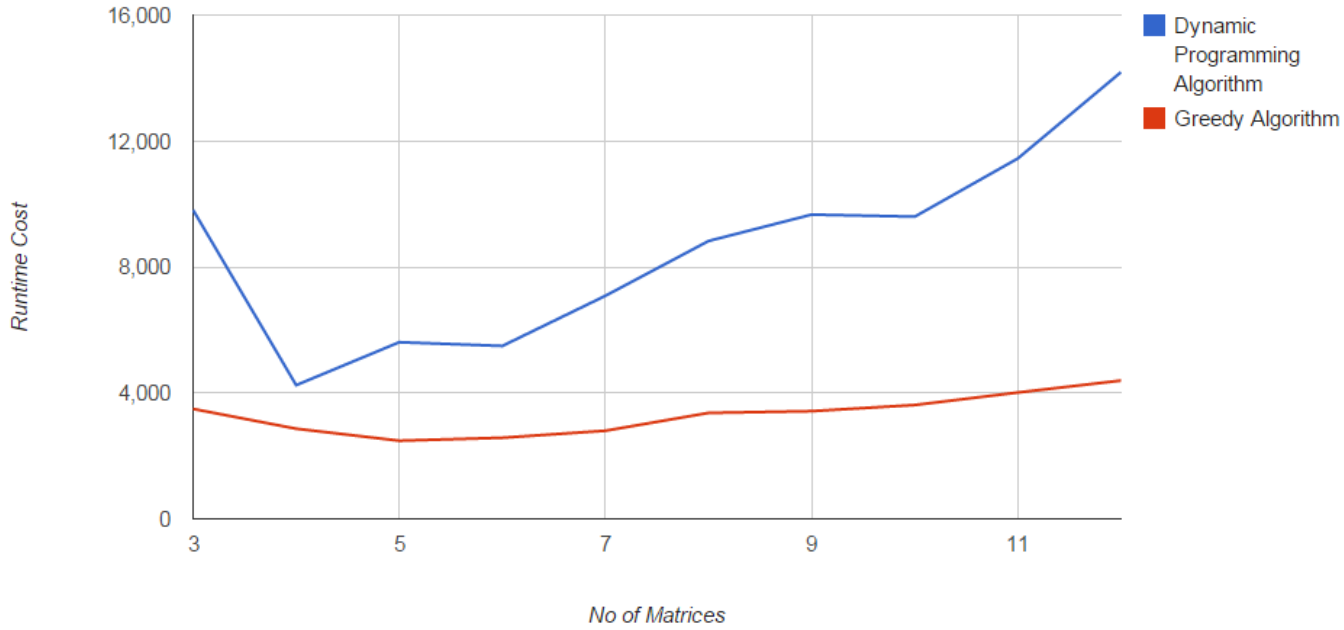

Fig -1bRuntime Cost v/s Number of matrices comparison between Greedy MCM and Dynamic MCM. Matrix Count: 3-10, Sequence of dimensions: 100-200.

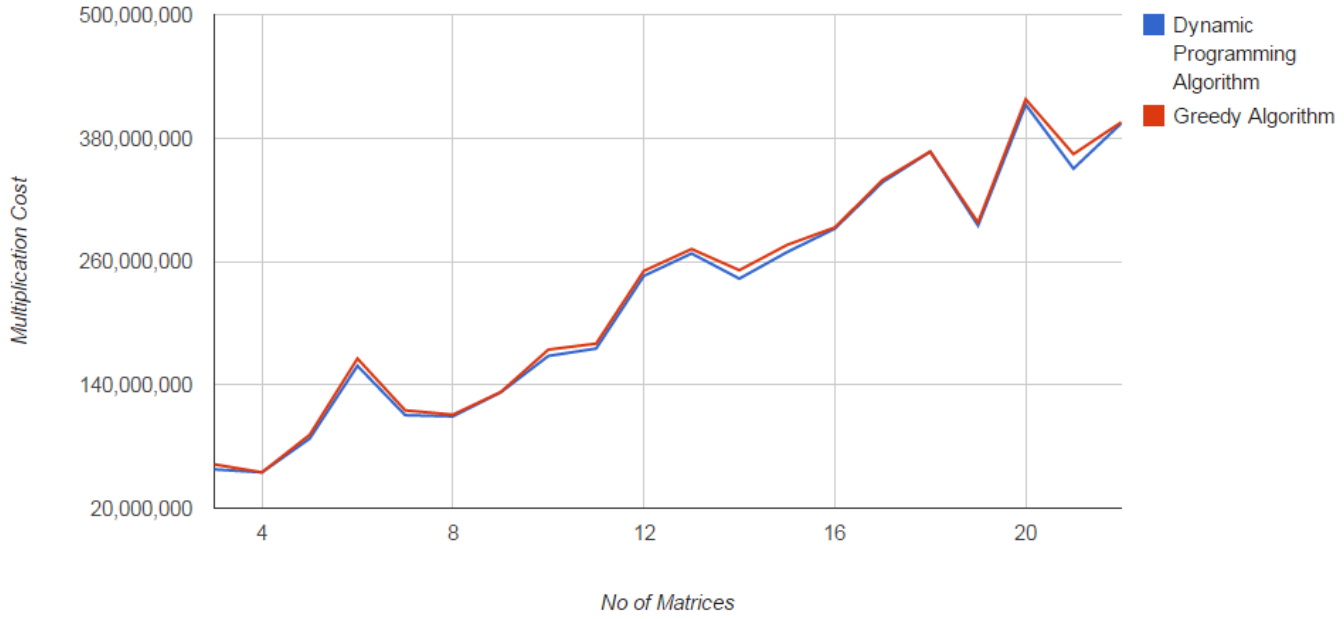

Fig -2a Multiplication Cost v/s Number of matrices comparison between Greedy MCM and Dynamic MCM. Matrix Count: 3-20, Sequence of dimensions: 200-400.

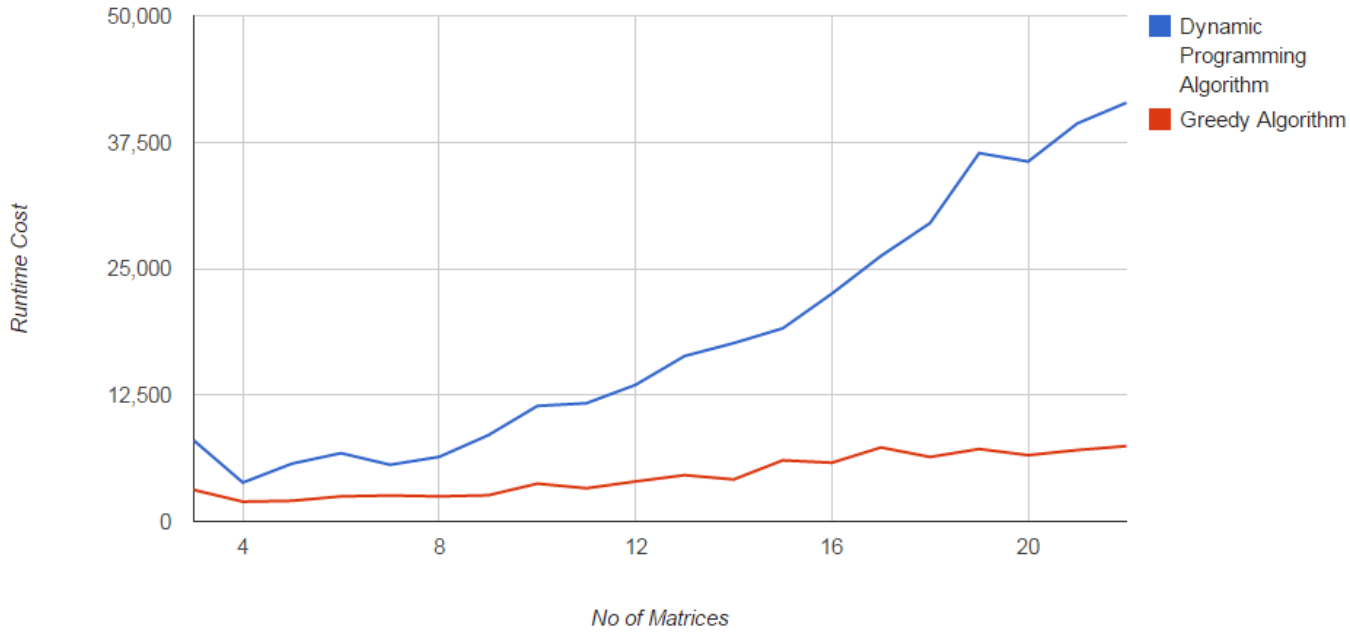

Fig -2b Runtime Cost v/s Number of matrices comparison between Greedy MCM and Dynamic MCM. Matrix Count: 3-20, Sequence of dimensions: 200-400.