4/10/2017

Optimal Matrix Chain and Heuristics

CS404 Project, Spring Semester 2017

Jonathan Gregory

Table of Contents

Author’s Declaration

I understand and have adhered to the rules regarding student conduct. In particular, any and all material, including algorithms and programs, have been produced and written by myself. Any outside sources that I have consulted are free, publicly available, and have been appropriately cited. I understand that a violation of the code of conduct will result in a zero (0) for this assignment, and that the situation will be discussed and forwarded to the Academic Dean of the School for any follow up action. It could result in being expelled from the university.

X

Jonathan Gregory
4/10/2017

Introduction (5%)

This report is going to provide a critical analysis of the optimal way to solve a matrix chain. It will provide an analyses of different heuristics along with a random execution tree to solve for the most optimal performance solution. This report will then demonstrate the effectiveness of those methods.

There is a known dynamic programming approach to solve a matrix chain for its optimal cost that has a cost to execute of $(1/3)n^3$. The goal of this project is to find out if any of the heuristics that cost less than $(1/3)n^3$ can get us results close to the actual result.

We will test each of the six different heuristics with given input files and against random chains. We will test against the length of the chain, mean of the dimensions of the chain, and the variance of the dimensions.

Experimental Design (20%)

Present an in-depth and critical analysis of each heuristics, before implementing and collecting data.

Strategy A – Remove Largest Dimension First

One heuristic, Remove Largest Dimension First, is to find the largest value of the "inner" dimensions, $\{d_1, . . . d_{n-1}\}$ (thus exclude the "outer" dimensions d_0 and d_n), and multiply the two matrices with this shared largest dimension. Repeat this process, until done. The resulting cost is called $M_a[1; n]$, and the relative extra overage is

$$Y_a(n) = \frac{M_a[1,n] \cancel{- M_0[1,n]}}{M_0[1,n]}. \qquad Y_a(n) = \frac{M_a[1,n]}{M_0[1,n]}$$

- Why should the heuristic make sense?
    - Since the cost of each matrix chain is based on the number of times you must multiply each value against every other value in the matrix, removing the largest dimensions first will result in having to multiply more numbers against only the smaller numbers. You must multiply all the numbers against each other at some point; by multiplying the largest first, you are left with the smaller ones in the end to multiply. This is a fairly effective heuristic especially since it costs less than the optimal solution.
- Are there examples of specific values of n and $[d_0, d_1, … d_{n-1}, d_n]$ such that the heuristic does result in the optimal value?
    - While this heuristic will get you a value close to the actual minimum cost, only when matrix chain dimensions are equal to 3 does the heuristic result in the optimal value. Anything larger than that, it just gets close to the optimal value. For instance, the matrix chain {1,6,2} results in costing 12 which is the same as the optimal solution. For more examples of this heuristic working, including a working example where n equals 20, see the Experimental Implementation for this strategy.
- Are there examples of specific values of n and $[d_0, d_1, … d_{n-1}, d_n]$ such that the heuristic does not result in the optimal value?
    - For all values on n over 3, the heuristic does not always provide the optimal value when compared against using only a Matrix Chain Optimal Strategy. For instance, the matrix chain {3, 5, 17, 12, 9} results in costing 1695 versus the optimal solution of 1191.
- What is the cost of executing the heuristic idea?
    - The cost of implementing this heuristic is $\Omega(n)=n^2$, which is a significant improvement over the optimal solution of $\Omega(n)=n^3$.

Strategy B – Do Most Expensive Matrix Multiplication First

Another heuristic, Do Most Expensive Matrix Multiplication First, is to find the largest value of the possible matrix multiplications and multiply the two matrices with this largest computation. In other words, find i such that the product di−1 · di · di+1 is largest from i = 1 . . . n-1, and multiply matrices $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$. Repeat this process, until done. The resulting cost is called $M_b[1; n]$, and the relative extra overage is

$$\cancel{Y_b(n) = \frac{M_b[1,n] - \cancel{M_0[1,n]}}{M_0[1,n]}}. \qquad Y_b(n) = \frac{M_b[1,n]}{M_0[1,n]}$$

- <u>Why should the heuristic make sense?</u>
  - This heuristic could make sense in the way that you are getting the most expensive multiplications out of the way first. You will have cheaper multiplications later. However, this heuristic is ineffective in finding a close approximation for the total cost despite costing less than the optimal solution.
- <u>Are there examples of specific values of n and [d_0, d_1, … d_{n-1}, d_n] such that the heuristic does result in the optimal value?</u>
  - While this heuristic will get you a value close to the actual minimum cost, only when matrix chain dimensions are equal to 3 does the heuristic result in the optimal value. Anything larger than that, it just gets close to the optimal value. For instance, the matrix chain {1,6,2} results in costing 12 which is the same as the optimal solution. For more examples of this heuristic working including a working example where n equals 20, see the Experimental Implementation for this strategy.
- <u>Are there examples of specific values of n and [d_0, d_1, … d_{n-1}, d_n] such that the heuristic does not result in the optimal value?</u>
  - For all values on n over 3, the heuristic does not always provide the optimal value when compared against using only a Matrix Chain Optimal Strategy. For instance, the matrix chain {3, 5, 17, 12, 9} results in costing 2736 versus the optimal solution of 1191.
- <u>What is the cost of executing the heuristic idea?</u>
  - The cost of implementing this heuristic is $\Omega(n)=n^2$, which is a significant improvement over the optimal solution of $\Omega(n)=n^3$.

Strategy C – Remove Smallest Dimensions First

Another heuristic, Remove Smallest Dimension First, is to find the smallest value of the "inner" dimensions, $\{d_1 \cdot \ldots \cdot d_{n-1}\}$, and multiply the two matrices with this shared smallest dimension. Repeat this process until done. The resulting cost is called $M_c[1; n]$, and the relative extra overage is

$$\cancel{Y_c(n) = \frac{M_c[1,n] - M_0[1,n]}{M_0[1,n]}}. \qquad Y_c(n) = \frac{M_c[1,n]}{M_0[1,n]}$$

- Why should the heuristic make sense?
  - This heuristic does not make as much sense as the other heuristic presented. By removing the smallest dimensions first, you are leaving the larger dimensions to stack up at the end. Near the end of the heuristic, you end up multiplying only the largest of the possible dimensions against each other resulting in a very high cost. Though this is more efficient than the optimal solution, it is significantly less effective.
- Are there examples of specific values of n and $[d_0, d_1, \ldots d_{n-1}, d_n]$ such that the heuristic does result in the optimal value?
  - While this heuristic will get you a value close to the actual minimum cost, only when matrix chain dimensions are equal to 3 does the heuristic result in the optimal value. Anything larger than that, it just gets close to the optimal value. For instance, the matrix chain {1,6,2} results in costing 12 which is the same as the optimal solution. For more examples of this heuristic working including a working example where n equals 20, see the Experimental Implementation for this strategy.
- Are there examples of specific values of n and $[d_0, d_1, \ldots d_{n-1}, d_n]$ such that the heuristic does not result in the optimal value?
  - For all values on n over 3, the heuristic does not always provide the optimal value when compared against using only a Matrix Chain Optimal Strategy. For instance, the matrix chain {3, 5, 17, 12, 9} results in costing 2550 versus the optimal solution of 1191.
- What is the cost of executing the heuristic idea?
  - The cost of implementing this heuristic is $\Omega(n)=n^2$, which is a significant improvement over the optimal solution of $\Omega(n)=n^3$.

Strategy D – Do Least Expensive Matrix Multiplication First

Another heuristic, Do Least Expensive Matrix Multiplication First, is to find the smallest value of the possible matrix multiplications, and multiply the two matrices with this smallest computation. In other words, find i such that the product $d_{i-1} \cdot d_i \cdot d_{i+1}$ is smallest from i = 1 . . . n - 1, and multiply matrices $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$. Repeat this process until done. The resulting cost is called $M_d[1; n]$, and the relative extra overage is

$$\cancel{Y_d(n) = \frac{M_d[1,n] - M_0[1,n]}{M_0[1,n]}.} \qquad Y_d(n) = \frac{M_d[1,n]}{M_0[1,n]}$$

- Why should the heuristic make sense?
  - This heuristic makes sense in that is looks for the absolute cheapest multiplication first. Each time will result in the next cheapest possible multiplication. However, since it does not look for the effect of the multiplication, one of the cheaper operations, it could result in a more expensive operation later that it might not have had before. This strategy does effectively reduce the number of multiplications needed and is more efficient, yet not as effective, as the optimal strategy.
- Are there examples of specific values of n and $[d_0, d_1, \ldots d_{n-1}, d_n]$ such that the heuristic does result in the optimal value?
  - While this heuristic will get you a value close to the actual minimum cost, only when matrix chain dimensions are equal to 3 does the heuristic result in the optimal value. Anything larger than that, it just gets close to the optimal value. For instance, the matrix chain {1,6,2} results in costing 12 which is the same as the optimal solution. For more examples of this heuristic working, including a working example where n equals 20, see the Experimental Implementation for this strategy.
- Are there examples of specific values of n and $[d_0, d_1, \ldots d_{n-1}, d_n]$ such that the heuristic does not result in the optimal value?
  - For all values on n over 3, the heuristic does not always provide the optimal value when compared against using only a Matrix Chain Optimal Strategy. For instance, the matrix chain {3, 5, 17, 2, 9, 10} results in costing 524 versus the optimal solution of 440.
- What is the cost of executing the heuristic idea?
  - The cost of implementing this heuristic is $\Omega(n)=n^2$, which is a significant improvement over the optimal solution of $\Omega(n)=n^3$.

Strategy E – Random Execution Tree

For this randomly generated execution tree, rather than executing it directly as the algorithm shows, you must change it to determine the cost of the randomized computation. Now this is only the first step. Rather than generating only one random execution tree, you can generate L of such execution trees, and use the execution tree that resulted in the lowest cost. For this option, you let L = 2n and report the minimal cost as $M_e[1; n]$, and the relative extra overage is

$$Y_e(n) = \frac{M_e[1, n]}{M_0[1, n]}$$

- Why should the heuristic make sense?
  - This heuristic makes sense in the sense that it does provide a valid execution tree. It creates 2n possible matrix trees and returns the lowest cost of those options. This is not necessarily any better than any of the other heuristics or the ignorant approach. This heuristic has a chance to provide the optimal cost value every time. However, it is just luck at that point. Out of all the possibilities, you are given 20 which could possibly include duplicates. This would be more efficient than just trying it with the ignorant approach.
- Are there examples of specific values of n and [d_0, d_1, … d_{n-1}, d_n] such that the heuristic does result in the optimal value?
  - One possibility of this working is when n=3 and the matrix chain is {1, 2, 3} which results in a cost of 6. For more examples of this heuristic working, including a working example where n equals 20, see the Experimental Implementation for this strategy.
- Are there examples of specific values of n and [d_0, d_1, … d_{-1}, d_n] such that the heuristic does not result in the optimal value?
  - One possibility of this not working is when n=4 and the matrix chain is {1, 2, 3, 4} which could result in 32 and not the optimal output of 18.
- What is the cost of executing the heuristic idea?
  - The cost of implementing this heuristic is $\Omega(n)=n$, which is a significant improvement over the optimal solution of $\Omega(n)=n^3$.

Strategy F – Ignorant Approach

Finally, there is the ignorant approach: $\mathbf{B}_1$ _ $\mathbf{B}_2$, multiply the result with $\mathbf{B}_3$, multiply the result with $\mathbf{B}_4$. and so on. The resulting cost is called M_f [1; n], and the relative extra overage is

$$\cancel{Y_f(n) = \frac{M_f[1,n] - M_0[1,n]}{M_0[1,n]}}. \qquad Y_f(n) = \frac{M_f[1,n]}{M_0[1,n]}$$

- Why should the heuristic make sense?
  - This heuristic only makes sense in the light that it is easy and simple to perform. It has a low cost of execution. You simply just multiply in order with no regard to the overall cost to perform this operation.
- Are there examples of specific values of n and [d_0, d_1, … d_{n-1}, d_n] such that the heuristic does result in the optimal value?
  - For all values of n 2 and under, it always provides the optimal value. When the matrix chain is {3, 5, 17, 2, 9, 10} it results in costing 38 which is the same as the optimal solution of 38.
- Are there examples of specific values of n and [d_0, d_1, … d_{n-1}, d_n] such that the heuristic does not result in the optimal value?
  - For all values on n over 3, the heuristic does not always provide the optimal value when compared against using only a Matrix Chain Optimal Strategy. For instance, the matrix chain {3, 5, 17, 2, 9, 10} results in costing 524 versus the optimal solution of 440.
- What is the cost of executing the heuristic idea?
  - The cost of executing this heuristic is linear which is Ω(n)=n.

Experimental Implementation (15%)

Coding Implementation

I decided to use C++ as the language to implement my program. I used it because of the robustness of the language and my familiarity with it. I could readily employee the use of double vectors and pointers to make my program more efficient. I could easily offer the user the option to load numbers from a file or to perform the Matrix Chain Heuristics Experiment. In my program when reading in numbers from a file, I had to assume that there are at least 3 items in the matrix chain and all separated by a comma followed by a space. The file must end in only the number and no extra return carriages. I was able to easily create many functions that allowed for robust efficient code. My program is robust for overflow assuming that the user follows the guide lines for the input file and has between 2 and 100 dimensions in each one.

Algorithm Implementation

To find the optimal matrix chain order, I used an algorithm found publicly online from geeksforgeeks.com from the Cormen book. The function required an array and an int of the array size, so to use the exact same function, I loaded the values of the matrix chain vector into an array. This is the only time I deviated from storing all the detentions in a vector. Because this particular function required an array, I had to make the assumption that no matrix chain will contain a length over 100.

I broke the problem down into 16 different functions as follows:

1. int main() – Prints out what the program can test for then goes to Selection() to get the user's input
2. void Selection() - Gets the users selection to load the chain from a file or perform the Matrix Chain Heuristic Experiment. Leads into either FileEntery() or MatrixChainHeuristicExperiment() based on the user's input.
3. void FileEntery() - Loads the numbers from a file after getting the input name from the user. It then prints to the terminal the numbers that it loaded along with the associated heuristics.
4. string removeSpaces(string) – Takes in a string and returns the same string without any spaces. This is needed to remove the spaces from the input file.
5. int MatrixChainOrder(int p[], int) – This function takes in an array of all the matrix chain dimensions and performs the optimal operation. It then returns the optimal cost. This particular function was based on a code segment from geeksforgeeks.com and is attributed as such in the code comments.
6. int RemoveLargestDimensionFirst(vector<int>) – This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.
7. int MostExpensiveMatrixMultiplicationFirst(vector<int>) - This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.

8. int RemoveSmallestDimensionFirst(vector<int>) - This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.
9. int LeastExpensiveMatrixMultiplicationFirst(vector<int>) - This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.
10. int RandomExecutionTree(vector<int>) - This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.
11. int BestRandomExecutionTree(vector<int>) - This function takes in the vector of matrix chain dimensions, computes the cost to perform this particular execution tree over 2n different trees and returns the lowest cost tree.
12. int IgnorantApproach(vector<int>) - This function takes in the vector of matrix chain dimensions and returns the total cost to perform this particular execution tree.
13. void MatrixChainHeuristicExperiment() – This function creates 30 different matrix chains for each value of n and then sends them each to getHeuristics() to get the individual heuristics for each set.
14. vector <int> BestAverageWorst(vector <vector <int> >, int) – This function looks for the best, worst, and average case for a specific heuristic and returns a vector containing just the best, worst, and average cases for each. It also returns the optimal value from using a matrix chain for the best, worst, and average case scenario. This secondary number is used to calculate Yb, Ya, and Yw by dividing the two sets of numbers.
15. void CreateChart(vector <vector <int> >) – This function takes in a vector of vectors containing the 30 trials from the MatrixChainHeuristicExperiment() and prints to the screen the best, worst and average case for each heuristics.
16. vector<int>getHeuristics(vector<int>) – This function takes in a matrix chain and loads all the different heuristics into a single vector and returns that vector.

Correct Implementation

Below you can find a worked out matrix chain of dimensions according to their respective heuristic algorithm proving the correctness of the implementation.

Correctness: Strategy A – Remove Largest Dimension First

For this algorithm, I looped around the entire matrix chain looking for the largest inner pair. Once it found the largest inner pair, it took the sub cost of multiplying it together and removed the multiplied pair. Below in figure 1, you can see the implementation of matrix chain using this algorithm with the input from the provided file: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 1

```
Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply largest Innter Dimention [22,29] & [29,26] = 16588

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [24,29] & [29,22] = 15312

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [26,29] & [29,27] = 20358

[18,6] [6,28] [28,28] [28,23] [23,26] [26,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [28,28] & [28,23] = 18032

[18,6] [6,28] [28,23] [23,26] [26,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [6,28] & [28,23] = 3864

[18,6] [6,23] [23,26] [26,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [25,27] & [27,26] = 17550

[18,6] [6,23] [23,26] [26,27] [27,23] [23,24] [24,25] [25,25] [25,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [26,27] & [27,23] = 16146

[18,6] [6,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,26] [26,24] [24,22] [22,26] [26,36]
Cost to Multiply largest Innter Dimention [22,26] & [26,36] = 20592

[18,6] [6,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,26] [26,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [25,26] & [26,24] = 15600

[18,6] [6,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [23,26] & [26,23] = 13754

[18,6] [6,23] [23,23] [23,24] [24,25] [25,25] [25,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [25,25] & [25,24] = 15000

[18,6] [6,23] [23,23] [23,24] [24,25] [25,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [24,25] & [25,24] = 14400

[18,6] [6,23] [23,23] [23,24] [24,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [24,24] & [24,22] = 12672

[18,6] [6,23] [23,23] [23,24] [24,22] [22,36]
Cost to Multiply largest Innter Dimention [23,24] & [24,22] = 12144

[18,6] [6,23] [23,23] [23,22] [22,36]
Cost to Multiply largest Innter Dimention [23,23] & [23,22] = 11638

[18,6] [6,23] [23,22] [22,36]
Cost to Multiply largest Innter Dimention [6,23] & [23,22] = 3036

[18,6] [6,22] [22,36]
Cost to Multiply largest Innter Dimention [6,22] & [22,36] = 4752

[18,6] [6,36]
Cost to Multiply largest Innter Dimention [18,6] & [6,36] = 3888

Total cost: 235326
```

Correctness: Strategy B – Do Most Expensive Matrix Multiplication First

For this algorithm, I looped around the entire matrix chain looking for the most expensive pair. Once it found the most expensive pair, it took the sub cost of multiplying it together and removed the multiplied pair. Below in figure 2, you can see the implementation of matrix chain using this algorithm with the input from the provided file: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 2

```
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply Most Expensive Pair [29,26] & [26,36] = 27144

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,36]
Cost to Multiply Most Expensive Pair [22,29] & [29,36] = 22968

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,36]
Cost to Multiply Most Expensive Pair [29,22] & [22,36] = 22968

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,36]
Cost to Multiply Most Expensive Pair [24,29] & [29,36] = 25056

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,36]
Cost to Multiply Most Expensive Pair [26,24] & [24,36] = 22464

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,36]
Cost to Multiply Most Expensive Pair [27,26] & [26,36] = 25272

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,36]
Cost to Multiply Most Expensive Pair [25,27] & [27,36] = 24300

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,36]
Cost to Multiply Most Expensive Pair [25,25] & [25,36] = 22500

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,36]
Cost to Multiply Most Expensive Pair [24,25] & [25,36] = 21600

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,36]
Cost to Multiply Most Expensive Pair [26,29] & [29,27] = 20358

[18,6] [6,28] [28,28] [28,23] [23,26] [26,27] [27,23] [23,24] [24,36]
Cost to Multiply Most Expensive Pair [23,24] & [24,36] = 19872

[18,6] [6,28] [28,28] [28,23] [23,26] [26,27] [27,23] [23,36]
Cost to Multiply Most Expensive Pair [27,23] & [23,36] = 22356

[18,6] [6,28] [28,28] [28,23] [23,26] [26,27] [27,36]
Cost to Multiply Most Expensive Pair [26,27] & [27,36] = 25272

[18,6] [6,28] [28,28] [28,23] [23,26] [26,36]
Cost to Multiply Most Expensive Pair [23,26] & [26,36] = 21528

[18,6] [6,28] [28,28] [28,23] [23,36]
Cost to Multiply Most Expensive Pair [28,23] & [23,36] = 23184

[18,6] [6,28] [28,28] [28,36]
Cost to Multiply Most Expensive Pair [28,28] & [28,36] = 28224

[18,6] [6,28] [28,36]
Cost to Multiply Most Expensive Pair [6,28] & [28,36] = 6048

[18,6] [6,36]
Cost to Multiply Most Expensive Pair [18,6] & [6,36] = 3888

Total cost: 385002
```

Correctness: Strategy C – Remove Smallest Dimensions First

For this algorithm, I looped around the entire matrix chain looking for the smallest inner pair. Once it found the smallest inner pair, it took the sub cost of multiplying it together and removed the multiplied pair. Below in figure 3, you can see the implementation of matrix chain using this algorithm with the input from the provided file: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 3

```
Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [18,6] & [6,28] = 3024

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [29,22] & [22,29] = 18502

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [27,23] & [23,24] = 14904

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [28,23] & [23,26] = 16744

[18,28] [28,28] [28,26] [26,29] [29,27] [27,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [26,24] & [24,29] = 18096

[18,28] [28,28] [28,26] [26,29] [29,27] [27,24] [24,25] [25,25] [25,27] [27,26] [26,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [27,24] & [24,25] = 16200

[18,28] [28,28] [28,26] [26,29] [29,27] [27,25] [25,25] [25,27] [27,26] [26,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [25,25] & [25,27] = 16875

[18,28] [28,28] [28,26] [26,29] [29,27] [27,25] [25,27] [27,26] [26,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [27,25] & [25,27] = 18225

[18,28] [28,28] [28,26] [26,29] [29,27] [27,27] [27,26] [26,29] [29,29] [29,26] [26,36]
Cost to Multiply the Smallest Innter dimension Pair [29,26] & [26,36] = 27144

[18,28] [28,28] [28,26] [26,29] [29,27] [27,27] [27,26] [26,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [27,26] & [26,29] = 20358

[18,28] [28,28] [28,26] [26,29] [29,27] [27,27] [27,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [28,26] & [26,29] = 21112

[18,28] [28,28] [28,29] [29,27] [27,27] [27,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [27,27] & [27,29] = 21141

[18,28] [28,28] [28,29] [29,27] [27,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [29,27] & [27,29] = 22707

[18,28] [28,28] [28,29] [29,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [28,28] & [28,29] = 22736

[18,28] [28,29] [29,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [18,28] & [28,29] = 14616

[18,29] [29,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [29,29] & [29,36] = 30276

[18,29] [29,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [29,29] & [29,36] = 30276

[18,29] [29,36]
Cost to Multiply the Smallest Innter dimension Pair [18,29] & [29,36] = 18792

Total cost: 351728
```

Correctness: Strategy D – Do Least Expensive Matrix Multiplication First

For this algorithm, I looped around the entire matrix chain looking for the least expensive pair. Once it found the least expensive pair, it took the sub cost of multiplying it together and removed the multiplied pair. Below in figure 4, you can see the implementation of matrix chain using this algorithm with the input from the provided file: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 4

```
Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,6] & [6,28] = 3024

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [23,24] & [24,25] = 13800

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,28] & [28,28] = 14112

[18,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,28] & [28,23] = 11592

[18,23] [23,26] [26,29] [29,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,23] & [23,26] = 10764

[18,26] [26,29] [29,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,26] & [26,29] = 13572

[18,29] [29,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,29] & [29,27] = 14094

[18,27] [27,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,27] & [27,23] = 11178

[18,23] [23,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,23] & [23,25] = 10350

[18,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,25] & [25,25] = 11250

[18,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,25] & [25,27] = 12150

[18,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,27] & [27,26] = 12636

[18,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,26] & [26,24] = 11232

[18,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,24] & [24,29] = 12528

[18,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,29] & [29,22] = 11484

[18,22] [22,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,22] & [22,29] = 11484

[18,29] [29,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,29] & [29,26] = 13572

[18,26] [26,36]
Cost to Multiply the Least Expensive Matrix Multiplication Pair [18,26] & [26,36] = 16848

Total cost: 215670
```

Correctness: Strategy E – Random Execution Tree

For this algorithm, I split the feature up into two different functions. The first function is to create and find 2n dimension trees and return the smallest cost of those trees. The second function is to actually create that random execution tree. With the file that I used with 20 dimensions in it, my function created 40 random possible execution trees. Below in figure 5, you can see the implementation of the matrix chain using this algorithm with the input from the provided file for one of those execution trees: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 5

```
Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply using a Random Execution Tree [26,29] & [29,27] = 20358

[18,6] [6,28] [28,28] [28,23] [23,26] [26,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply using a Random Execution Tree [26,27] & [27,23] = 16146

[18,6] [6,28] [28,28] [28,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to Multiply using a Random Execution Tree [22,29] & [29,26] = 16588

[18,6] [6,28] [28,28] [28,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply using a Random Execution Tree [27,26] & [26,24] = 16848

[18,6] [6,28] [28,28] [28,23] [23,26] [26,23] [23,24] [24,25] [25,25] [25,27] [27,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply using a Random Execution Tree [28,23] & [23,26] = 16744

[18,6] [6,28] [28,28] [28,26] [26,23] [23,24] [24,25] [25,25] [25,27] [27,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply using a Random Execution Tree [25,25] & [25,27] = 16875

[18,6] [6,28] [28,28] [28,26] [26,23] [23,24] [24,25] [25,27] [27,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply using a Random Execution Tree [24,25] & [25,27] = 16200

[18,6] [6,28] [28,28] [28,26] [26,23] [23,24] [24,27] [27,24] [24,29] [29,22] [22,26] [26,36]
Cost to Multiply using a Random Execution Tree [22,26] & [26,36] = 20592

[18,6] [6,28] [28,28] [28,26] [26,23] [23,24] [24,27] [27,24] [24,29] [29,22] [22,36]
Cost to Multiply using a Random Execution Tree [28,28] & [28,26] = 20384

[18,6] [6,28] [28,26] [26,23] [23,24] [24,27] [27,24] [24,29] [29,22] [22,36]
Cost to Multiply using a Random Execution Tree [28,26] & [26,23] = 16744

[18,6] [6,28] [28,23] [23,24] [24,27] [27,24] [24,29] [29,22] [22,36]
Cost to Multiply using a Random Execution Tree [6,28] & [28,23] = 3864

[18,6] [6,23] [23,24] [24,27] [27,24] [24,29] [29,22] [22,36]
Cost to Multiply using a Random Execution Tree [24,29] & [29,22] = 15312

[18,6] [6,23] [23,24] [24,27] [27,24] [24,22] [22,36]
Cost to Multiply using a Random Execution Tree [6,23] & [23,24] = 3312

[18,6] [6,24] [24,27] [27,24] [24,22] [22,36]
Cost to Multiply using a Random Execution Tree [24,27] & [27,24] = 15552

[18,6] [6,24] [24,24] [24,22] [22,36]
Cost to Multiply using a Random Execution Tree [6,24] & [24,24] = 3456

[18,6] [6,24] [24,22] [22,36]
Cost to Multiply using a Random Execution Tree [24,22] & [22,36] = 19008

[18,6] [6,24] [24,36]
Cost to Multiply using a Random Execution Tree [6,24] & [24,36] = 5184

[18,6] [6,36]
Cost to Multiply using a Random Execution Tree [18,6] & [6,36] = 3888

Total cost: 247055
```

Correctness: Strategy F – Ignorant Approach

For this algorithm, I simply multiplied the first two matrices and then removed the second dimension. I added the cost of each multiplication to the total cost. Below in figure 6, you can see the implementation of the matrix chain using this algorithm with the input from the provided file: CS404SP17MatrixChainHeuristicsInput1.txt

Figure 6

```
Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: CS404SP17MatrixChainHeuristicsInput1.txt
Found CS404SP17MatrixChainHeuristicsInput1.txt
Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

[18,6] [6,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,6] & [6,28] = 3024

[18,28] [28,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,28] & [28,28] = 14112

[18,28] [28,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,28] & [28,23] = 11592

[18,23] [23,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,23] & [23,26] = 10764

[18,26] [26,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,26] & [26,29] = 13572

[18,29] [29,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,29] & [29,27] = 14094

[18,27] [27,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,27] & [27,23] = 11178

[18,23] [23,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,23] & [23,24] = 9936

[18,24] [24,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,24] & [24,25] = 10800

[18,25] [25,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,25] & [25,25] = 11250

[18,25] [25,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,25] & [25,27] = 12150

[18,27] [27,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,27] & [27,26] = 12636

[18,26] [26,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,26] & [26,24] = 11232

[18,24] [24,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,24] & [24,29] = 12528

[18,29] [29,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,29] & [29,22] = 11484

[18,22] [22,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,22] & [22,29] = 11484

[18,29] [29,26] [26,36]
Cost to multiply using the Ignorant Approach [18,29] & [29,26] = 13572

[18,26] [26,36]
Cost to multiply using the Ignorant Approach [18,26] & [26,36] = 16848

Total cost: 212256
```

Correctness: Manual Input

For the option to allow manual input, I created several functions that allow the user to choose between two options. The user is given the option to select either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment. If the user selects the manual input option, it asks the user for the file name of the text file containing the matrix chain dimensions. I had to assume that there are between 2 and 100 numbers inside, all separated by a comma and a space with no spaces, comma, or return carriage at the end of the file. The function then opens and verifies the file exist. It then imports the contents to a vector. From there, the vector is sent to another function that gets all the heuristics from the appropriate functions, loads them all into an array, and returns it back to the original function which then prints it out to the screen for the user.

Data Collection and Interpretation of Results (30%)

Heuristics from Experiment 1 (n=10, 15, 20, and 25)

Figure 7 depicts the output from the heuristics experiment when the upper bound is 17 and the lower bound is 7. The values tested for n are 10, 15, 20, and 25. The values for Yb is found by taking the best of the 30 trials with the given heuristic, divided by the optimal cost as given by a matrix chain for that specific trial. The values for Ya is found by taking the average of the 30 trials with the given heuristic, divided by the average of the optimal cost as given by a matrix chain for all 30 trials. The values for Yw is found by taking the worst of the 30 trials with the given heuristic, divided by the optimal cost as given by a matrix chain for that specific trial.

Figure 7

Optimal Cost Heuristics	n = 10 Mo[1,10] Yb		Ya		Yw
Largest Dimension First	(5807) 1.04857	-	(10173) 1.11448	-	(19788) 1.06182
Most Expensive First	(7336) 1.32467	-	(15004) 1.64373	-	(25881) 1.38876
Smallest Dimension First	(8324) 1.50307	-	(16978) 1.85999	-	(29396) 1.57738
Least Expensive First	(6597) 1.19122	-	(10872) 1.19106	-	(19644) 1.05409
Best of Several Random	(6002) 1.08378	-	(10125) 1.10922	-	(19500) 1.04636
One After the Other	(6804) 1.06931	-	(12711) 1.39253	-	(21330) 1.55206

Optimal Cost Heuristics	n = 15 Mo[1,15] Yb		Ya		Yw
Largest Dimension First	(10920) 1.19084	-	(16509) 1.17351	-	(26718) 1.16464
Most Expensive First	(15363) 1.44294	-	(25535) 1.81511	-	(35883) 1.56414
Smallest Dimension First	(13168) 1.43599	-	(29141) 2.07144	-	(42749) 2.47748
Least Expensive First	(9384) 1.02334	-	(18005) 1.27985	-	(34451) 1.50172
Best of Several Random	(10659) 1.16238	-	(17244) 1.22576	-	(29261) 1.27549
One After the Other	(12558) 1.06659	-	(22451) 1.59589	-	(35568) 1.86572

Optimal Cost Heuristics	n = 20 Mo[1,20] Yb		Ya		Yw
Largest Dimension First	(16064) 1.13551	-	(23768) 1.20766	-	(34279) 1.25804
Most Expensive First	(27852) 1.77198	-	(37714) 1.91626	-	(52930) 1.95227
Smallest Dimension First	(30890) 1.76676	-	(45082) 2.29064	-	(61789) 2.26765
Least Expensive First	(17510) 1.09492	-	(27941) 1.41969	-	(38439) 1.79983
Best of Several Random	(16180) 1.14371	-	(25635) 1.30253	-	(36295) 1.33202
One After the Other	(16359) 1.02295	-	(34537) 1.75484	-	(51425) 1.94497

Optimal Cost Heuristics	n = 25 Mo[1,25] Yb		Ya		Yw
Largest Dimension First	(16902) 1.00992	-	(27631) 1.19838	-	(37849) 1.40661
Most Expensive First	(25303) 1.51189	-	(44210) 1.91742	-	(55879) 1.95415
Smallest Dimension First	(39501) 2.36024	-	(54271) 2.35378	-	(65166) 2.56277
Least Expensive First	(19793) 1.18266	-	(31469) 1.36483	-	(46772) 1.7399
Best of Several Random	(19475) 1.16366	-	(30438) 1.32012	-	(42393) 1.48253
One After the Other	(22200) 1.22227	-	(41865) 1.81572	-	(58395) 2.04214

Figure 8

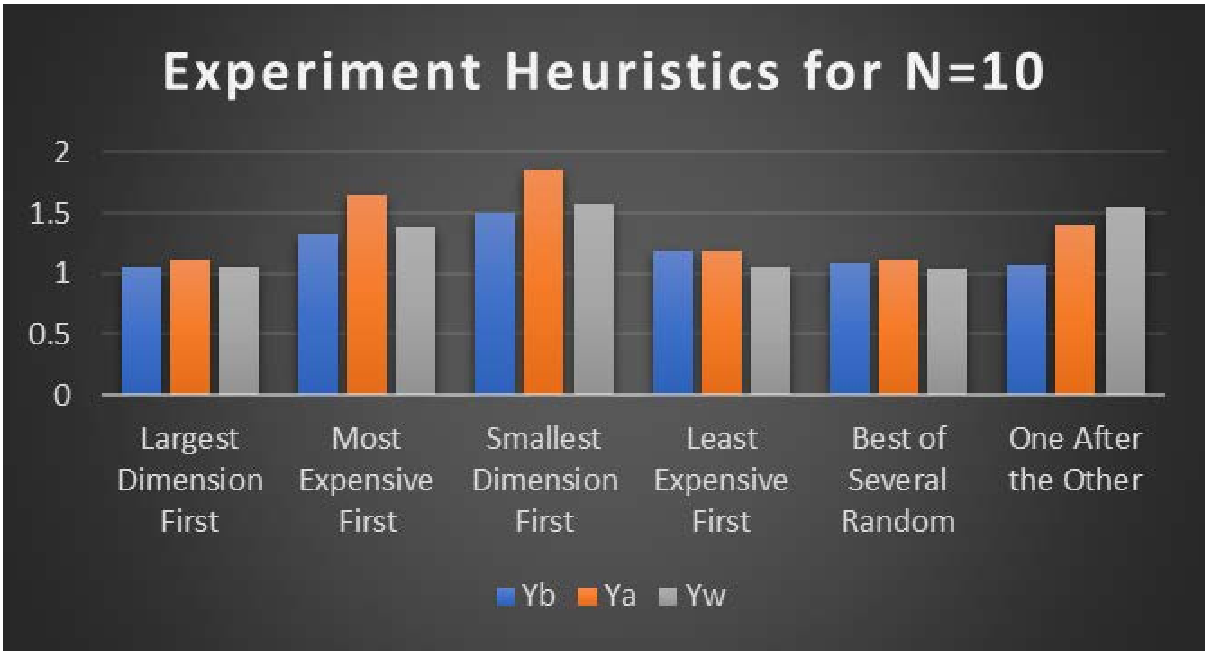


Figure 9

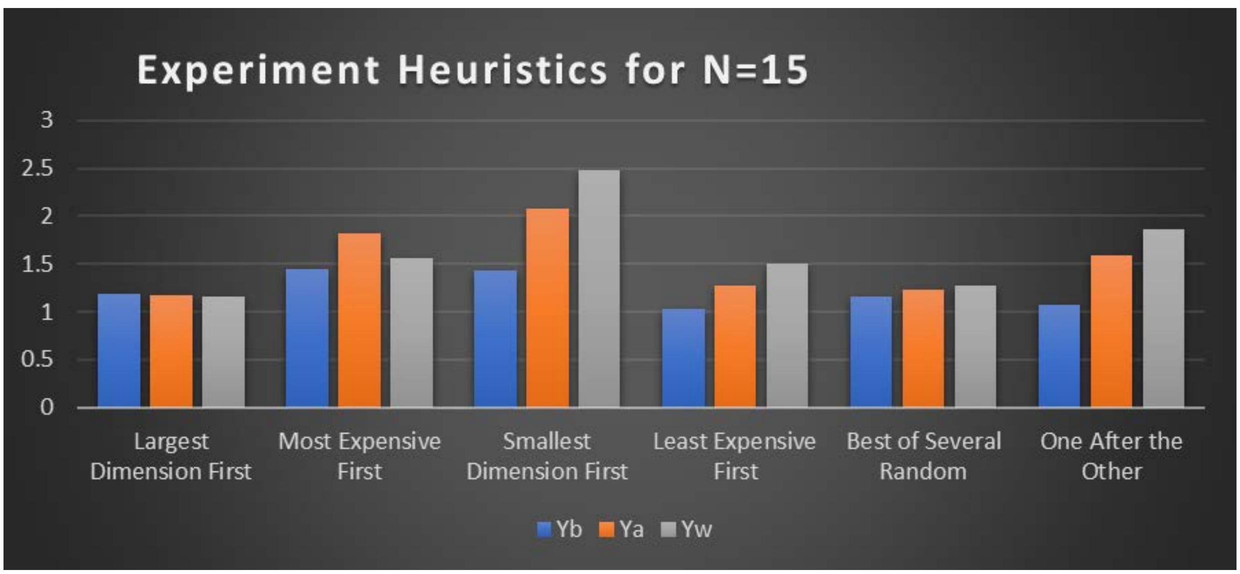


Figure 10

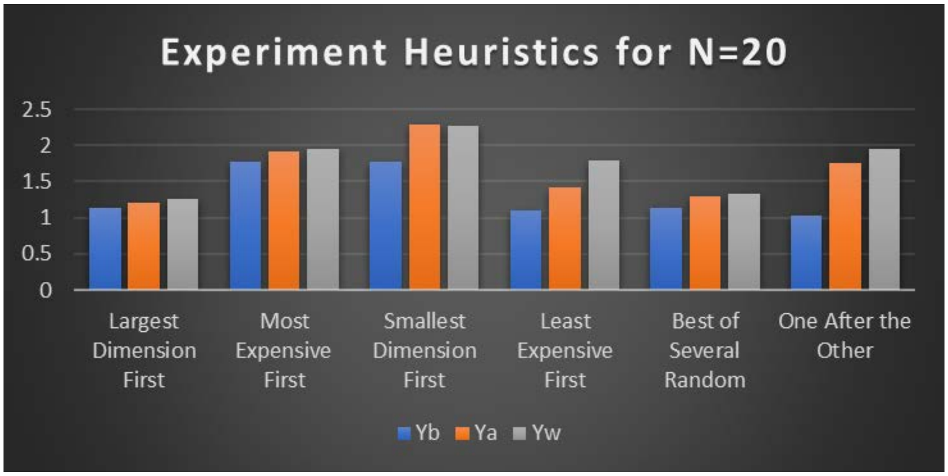
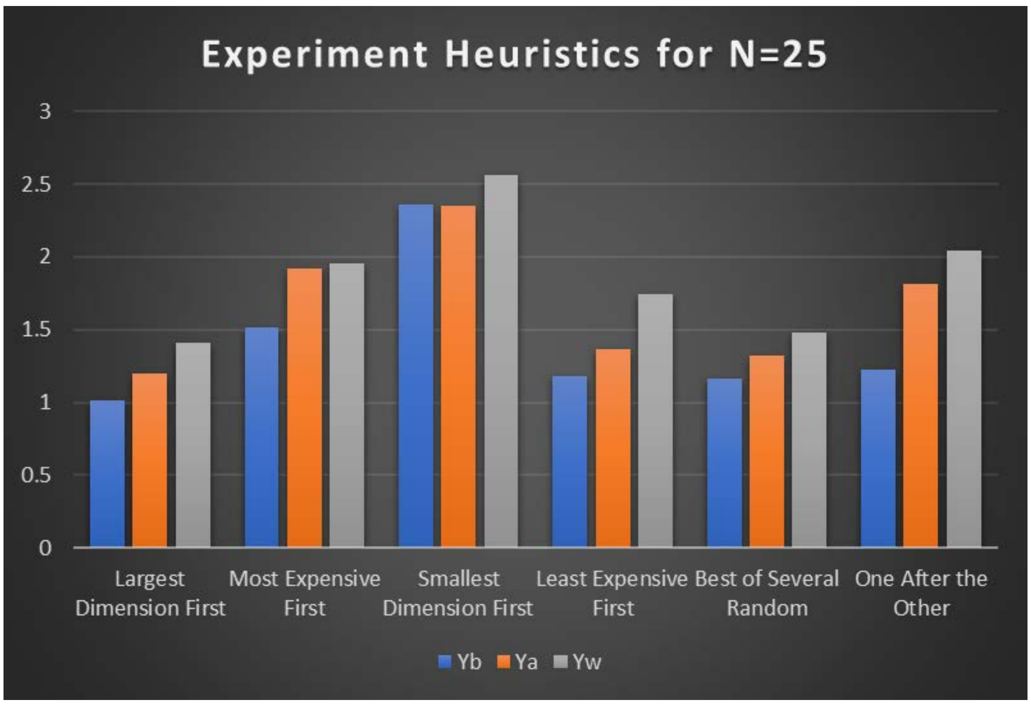
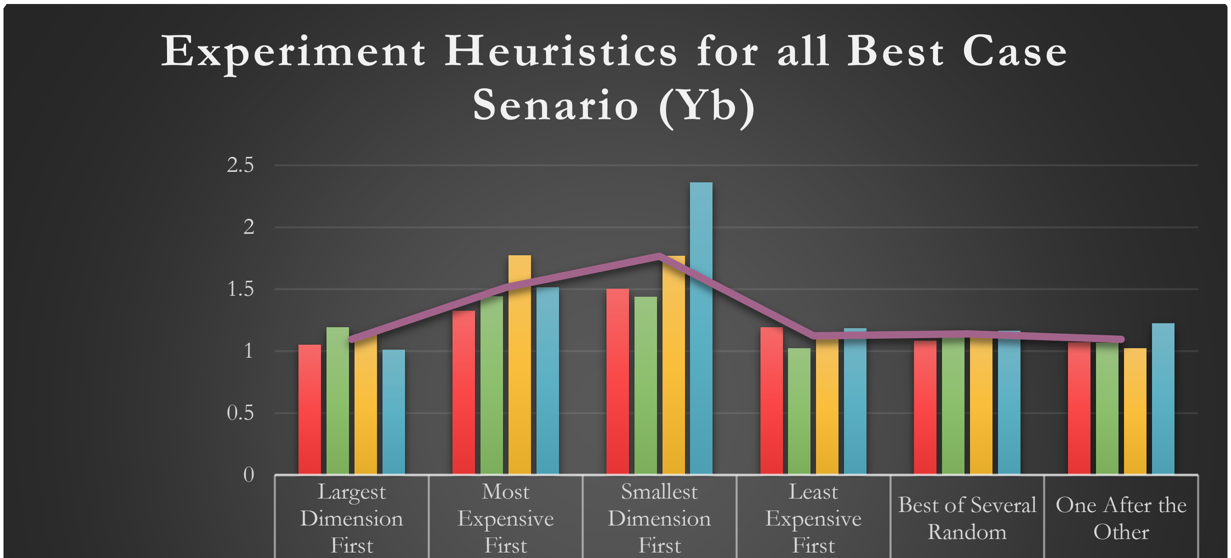


Figure 11



Figures 8, 9, 10 and 11 portray the variance of the best, average, and worst case costs for each of the heuristics when n is between 10 and 25.

Figure 12



	Largest Dimension First	Most Expensive First	Smallest Dimension First	Least Expensive First	Best of Several Random	One After the Other
n=10	1.04857	1.32467	1.50307	1.19122	1.08378	1.06931
n=15	1.19084	1.44294	1.43599	1.02334	1.16238	1.06659
n=20	1.13551	1.77198	1.76676	1.09492	1.14371	1.02295
n=25	1.00992	1.51189	2.36024	1.18266	1.16366	1.22227
average of 10-25	1.09621	1.51287	1.766515	1.123035	1.1383825	1.09528

In figure 12, we can clearly see the best case (Yb) from each of the heuristics. We are able to find the averages of the results taken in all four matrix chain dimension scenarios. We can conclude that in this particular case, the Largest Dimensions First scenario can reliably give us the closest result to the optimal solution at 1.096 on average. Surprisingly, in this particular experiment, performing the multiplications using the One After The Other technique happens to be the most accurate at 1.095. This is just by pure chance and would probably not yield the same results if it were to be run again. We can also conclude that the heuristic that goes after finding the smallest dimension first results in the worst accuracy at 1.766.

Length of Chain

Figure 13

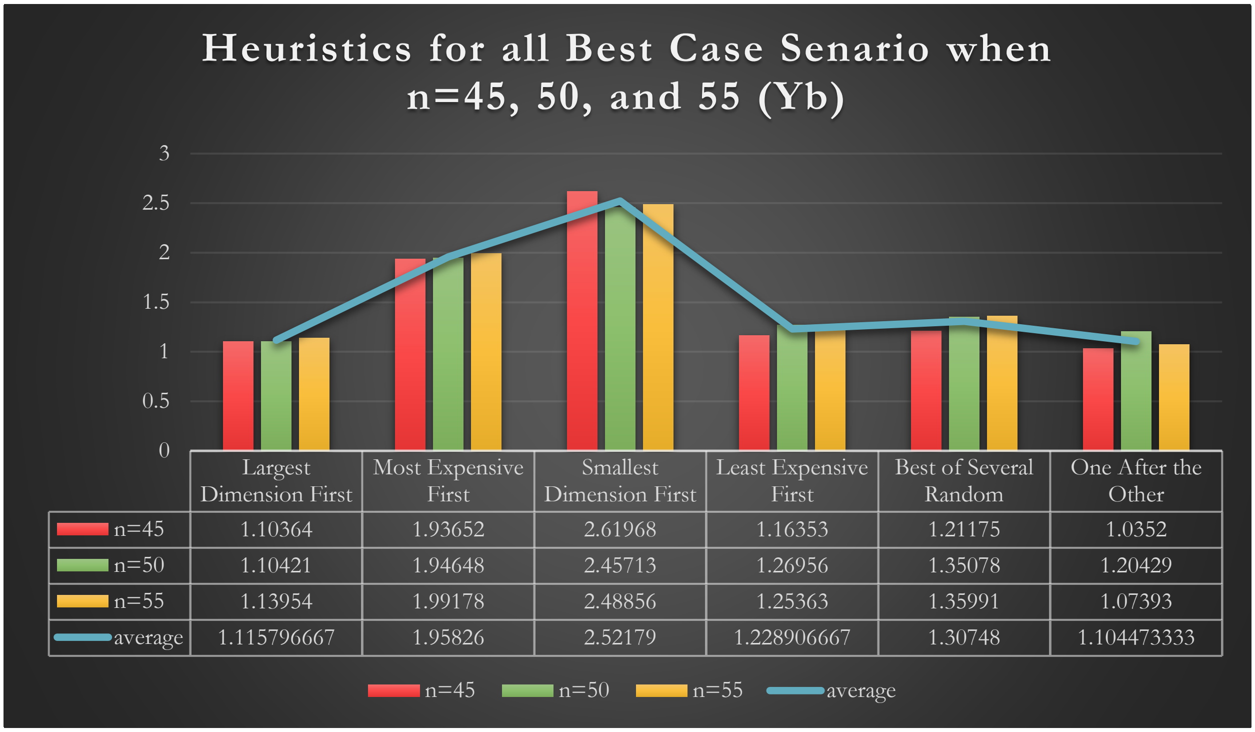
Optimal Cost / Heuristics	n = 3, Mo[1,3] Yb	Ya	Yw
Largest Dimension First	(588) 1	(1996) 1	(4624) 1
Most Expensive First	(588) 1	(1996) 1	(4624) 1
Smallest Dimension First	(588) 1	(1996) 1	(4624) 1
Least Expensive First	(588) 1	(1996) 1	(4624) 1
Best of Several Random	(588) 1	(1996) 1	(4624) 1
One After the Other	(588) 1	(1996) 1	(4624) 1

Optimal Cost / Heuristics	n = 4, Mo[1,4] Yb	Ya	Yw
Largest Dimension First	(1386) 1	(3400) 1.01827	(6409) 1.0173
Most Expensive First	(1638) 1.01111	(4123) 1.2348	(7072) 1.33939
Smallest Dimension First	(1620) 1	(4115) 1.2324	(6647) 1.50385
Least Expensive First	(1344) 1	(3479) 1.04193	(6630) 1.35417
Best of Several Random	(1344) 1	(3339) 1	(6300) 1
One After the Other	(1344) 1	(3650) 1.09314	(7072) 1.33939

As illustrated in figure 13, we can see that the accuracy of the heuristic is effected by the length of the chain. We can see that for lower values such as 3 and 4, the algorithms are just as efficient as the optimal solution as all the cases for this particular set for their respective best cases. I ran this particular experiment many times and found the results to be nearly the same every time. Rarely, I would find that one or two of the heuristics best cases would vary slightly.

Mean of Dimensions

Figure 14



	Largest Dimension First	Most Expensive First	Smallest Dimension First	Least Expensive First	Best of Several Random	One After the Other
n=45	1.10364	1.93652	2.61968	1.16353	1.21175	1.0352
n=50	1.10421	1.94648	2.45713	1.26956	1.35078	1.20429
n=55	1.13954	1.99178	2.48856	1.25363	1.35991	1.07393
average	1.115796667	1.95826	2.52179	1.228906667	1.30748	1.104473333

In figure 14, we can see the results when we change the value of n to vary between 45 and 55. In figure 15, we can verify that the results are very similar to the results found in figure 12 where n is between 10 and 25. This goes to show that while the results are not exactly proportional, they are relatively close regardless of the means of the dimensions.

Variance of Dimensions

Figure 15

Optimal Cost / Heuristics	n = 7, Mo[1,7] Yb	Ya	Yw
Largest Dimension First	(4256) 1	(6778) 1.11719	(13750) 1.4133
Most Expensive First	(6129) 1.22287	(9635) 1.5881	(15878) 1.63203
Smallest Dimension First	(6243) 1.52977	(10212) 1.6832	(13585) 1.55773
Least Expensive First	(4326) 1.06003	(7017) 1.15658	(11978) 1.36985
Best of Several Random	(4256) 1	(6399) 1.05472	(11343) 1.1659
One After the Other	(5160) 1	(8781) 1.44734	(15878) 1.63203

Optimal Cost / Heuristics	n = 8, Mo[1,8] Yb	Ya	Yw
Largest Dimension First	(4047) 1.0926	(7889) 1.0838	(12180) 1.03396
Most Expensive First	(6230) 1.68197	(12024) 1.65188	(18292) 1.5528
Smallest Dimension First	(6937) 1.87284	(13794) 1.89504	(20084) 1.93562
Least Expensive First	(4501) 1.21517	(9137) 1.25525	(17760) 1.71164
Best of Several Random	(3998) 1.07937	(7992) 1.09795	(13624) 1.15654
One After the Other	(4011) 1.08288	(11733) 1.6119	(18343) 2.20575

Optimal Cost / Heuristics	n = 9, Mo[1,9] Yb	Ya	Yw
Largest Dimension First	(4734) 1.04342	(10158) 1.16384	(16488) 1.04885
Most Expensive First	(6123) 1.34957	(13940) 1.59716	(22589) 1.43696
Smallest Dimension First	(7003) 1.54353	(15245) 1.74668	(25002) 1.59046
Least Expensive First	(4631) 1.02072	(10686) 1.22434	(19930) 1.26781
Best of Several Random	(4948) 1.09059	(9814) 1.12443	(16666) 1.06018
One After the Other	(5670) 1.24972	(12358) 1.4159	(19357) 1.23136

Optimal Cost / Heuristics	n = 10, Mo[1,10] Yb	Ya	Yw
Largest Dimension First	(6104) 1	(10660) 1.15543	(17136) 1.26512
Most Expensive First	(9454) 1.53474	(15232) 1.65099	(22356) 1.8029
Smallest Dimension First	(8976) 1.45714	(17459) 1.89237	(25764) 2.07774
Least Expensive First	(6206) 1.13414	(11365) 1.23184	(17734) 1.64082
Best of Several Random	(5584) 1.02047	(10579) 1.14665	(15946) 1.17726
One After the Other	(6503) 1.05568	(13723) 1.48743	(21296) 1.74787

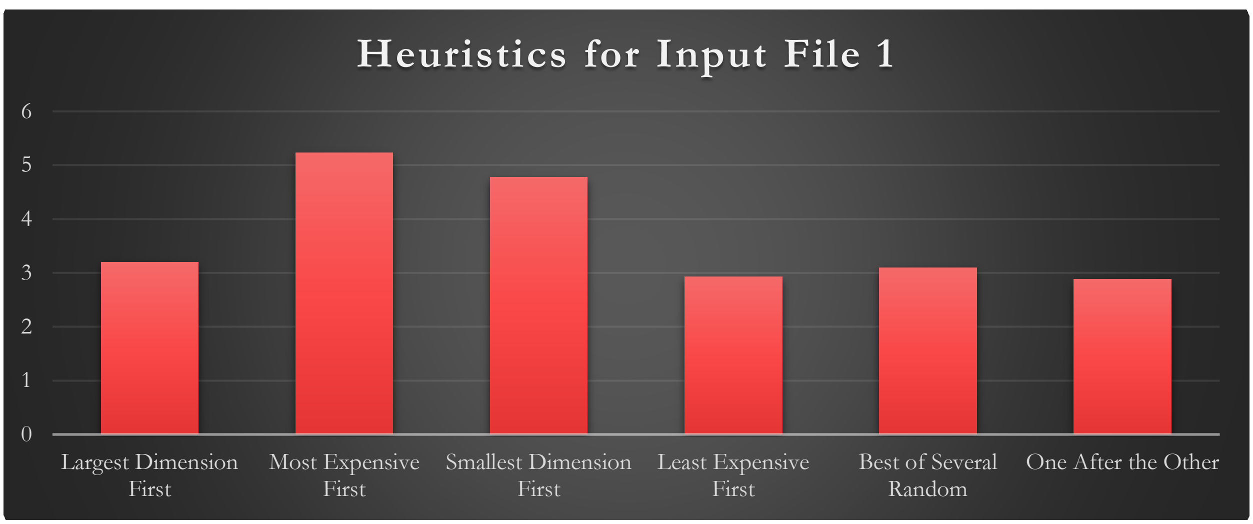
Figure 16 shows that the heuristics do vary in proportion when the variance of dimensions is altered.

Heuristics from CS404SP17MatrixChainHeuristicsInput1.txt

Figure 16

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput1.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput1.txt

Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

----------------------------------------------------------
| Heuristics                     |  Cost                   |
----------------------------------------------------------
| Largest Dimension First        | (235326)      3.19597  |
| Most Expensive First           | (385002)      5.22873  |
| Smallest Dimension First       | (351728)      4.77684  |
| Least Expensive First          | (215670)      2.92903  |
| Best of Several Random         | (228246)      3.09982  |
| One After the Other            | (212256)      2.88266  |
| Chain Multiplication           | (73632)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Figure 17



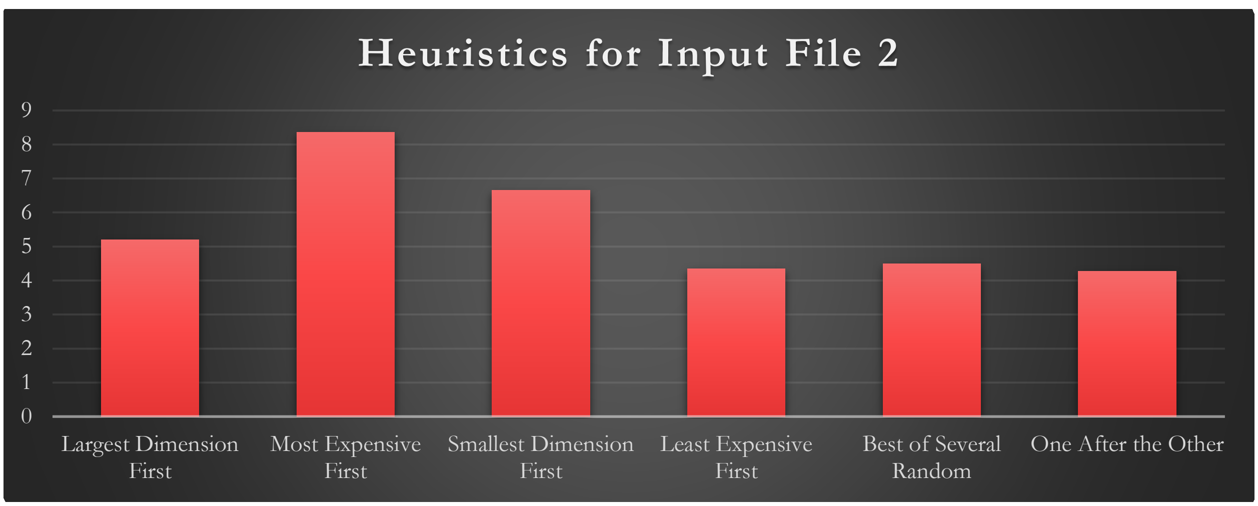
Figures 17 and 18 show the output of input file 1 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput2.txt

Figure 18

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput2.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput2.txt

Matrix Chain Loaded: 18 4 23 23 25 29 24 24 22 23 26 25 26 27 23 24 28 29 23 38

----------------------------------------------------------
| Heuristics                      |  Cost                  |
----------------------------------------------------------
| Largest Dimension First         | (241289)      5.19258  |
| Most Expensive First            | (388474)      8.36003  |
| Smallest Dimension First        | (309019)      6.65015  |
| Least Expensive First           | (201918)      4.34531  |
| Best of Several Random          | (208884)      4.49522  |
| One After the Other             | (198450)      4.27068  |
| Chain Multiplication            | (46468)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Figure 19



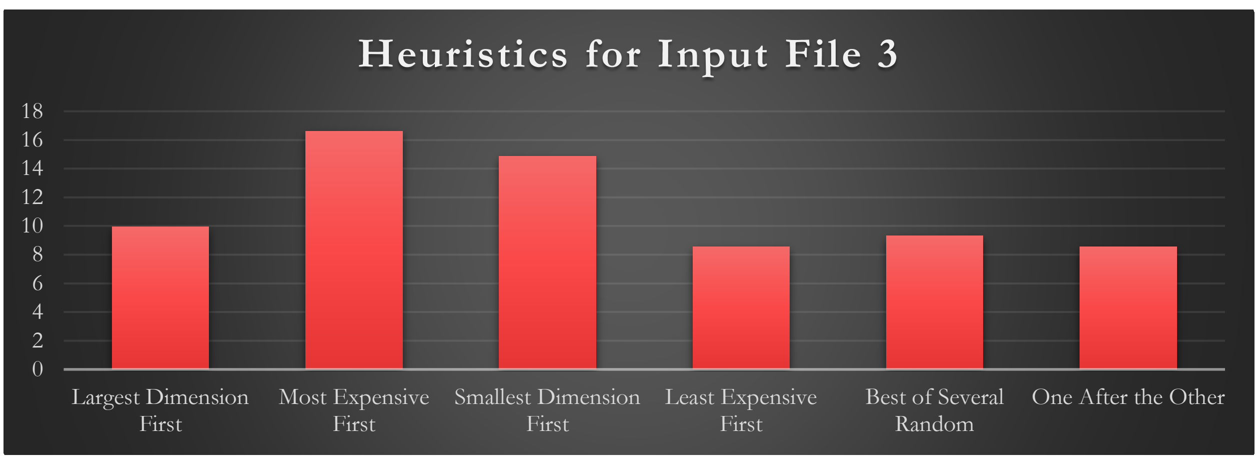
Figures 19 and 20 show the output of input file 2 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput3.txt

Figure 20

```
Optimal Matrix Chain and Heuristics

-------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
-------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput3.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput3.txt

Matrix Chain Loaded: 18 2 26 24 29 22 27 26 27 28 25 28 24 27 26 28 28 24 26 38

---------------------------------------------------------
| Heuristics                      |  Cost                    |
---------------------------------------------------------
| Largest Dimension First         | (250626)      9.93444  |
| Most Expensive First            | (419140)      16.6141  |
| Smallest Dimension First        | (374892)      14.8602  |
| Least Expensive First           | (215676)      8.54907  |
| Best of Several Random          | (235150)      9.32099  |
| One After the Other             | (215676)      8.54907  |
| Chain Multiplication            | (25228)       1        |
---------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: _
```

Figure 21



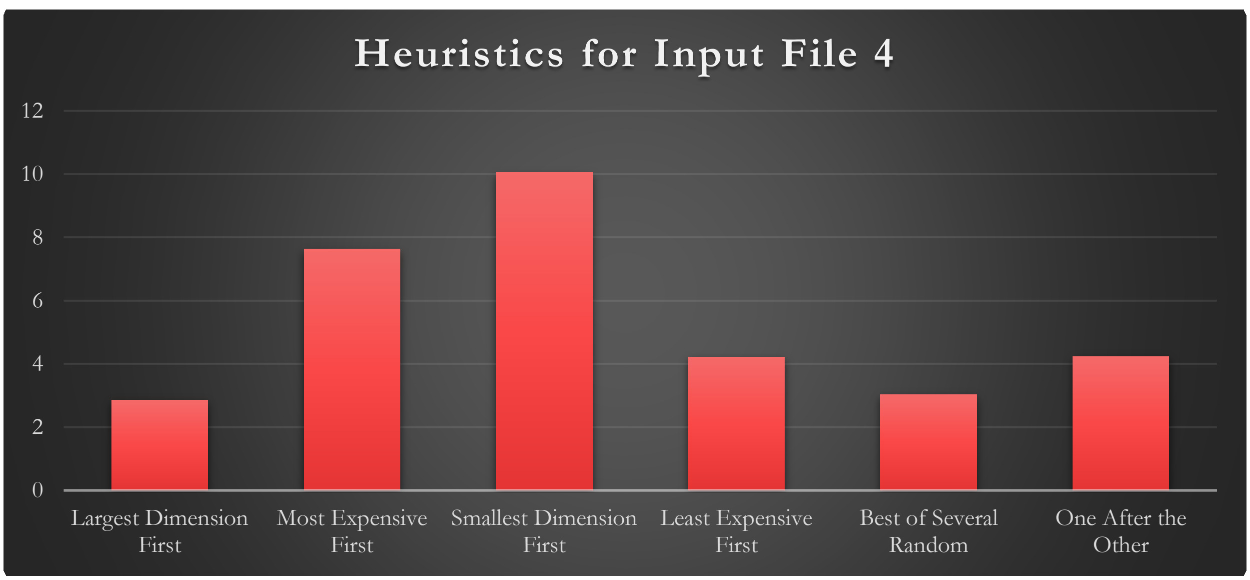
Figures 21 and 22 show the output of input file 3 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput4.txt

Figure 22

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput4.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput4.txt

Matrix Chain Loaded: 18 4 19 37 14 28 28 14 16 16 21 37 21 38 13 19 31 26 13 35

----------------------------------------------------------
| Heuristics                       |  Cost                 |
----------------------------------------------------------
| Largest Dimension First          | (108880)      2.86045  |
| Most Expensive First             | (290474)      7.6312   |
| Smallest Dimension First         | (382512)      10.0492  |
| Least Expensive First            | (160462)      4.21558  |
| Best of Several Random           | (115369)      3.03092  |
| One After the Other              | (161316)      4.23802  |
| Chain Multiplication             | (38064)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: _
```

Figure 23



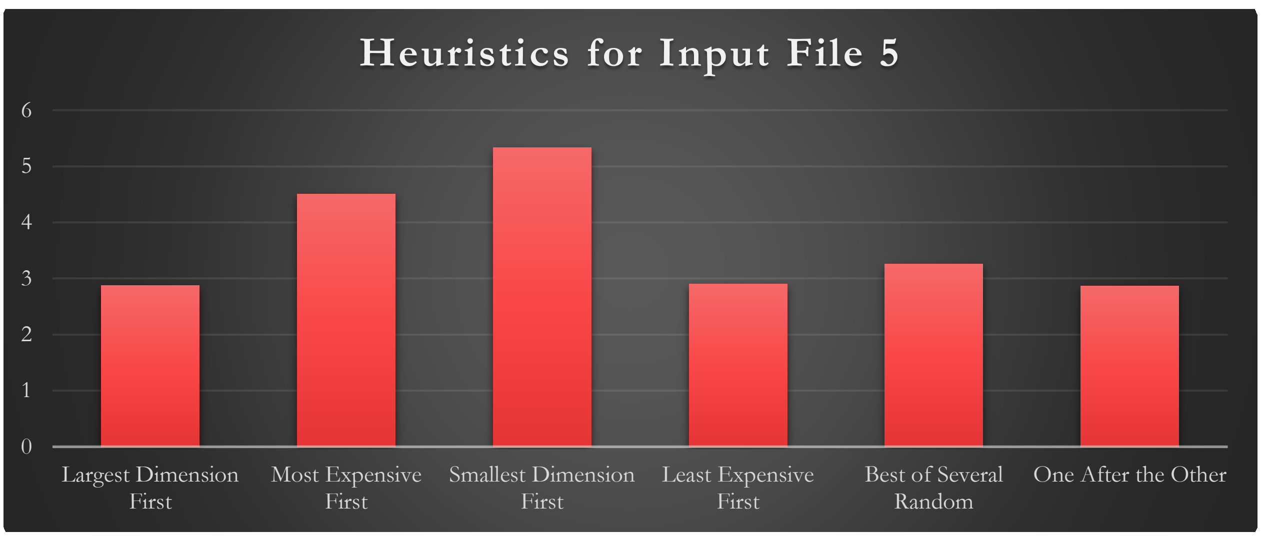
Figures 23 and 24 show the output of input file 4 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput5.txt

Figure 24

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput5.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput5.txt

Matrix Chain Loaded: 18 6 20 18 15 23 24 30 21 26 20 37 23 22 31 36 34 19 20 35

----------------------------------------------------------
| Heuristics                      |  Cost                   |
----------------------------------------------------------
| Largest Dimension First         | (197290)      2.87981  |
| Most Expensive First            | (308824)      4.50785  |
| Smallest Dimension First        | (365135)      5.32982  |
| Least Expensive First           | (198728)      2.9008   |
| Best of Several Random          | (223464)      3.26187  |
| One After the Other             | (196344)      2.866    |
| Chain Multiplication            | (68508)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Figure 25



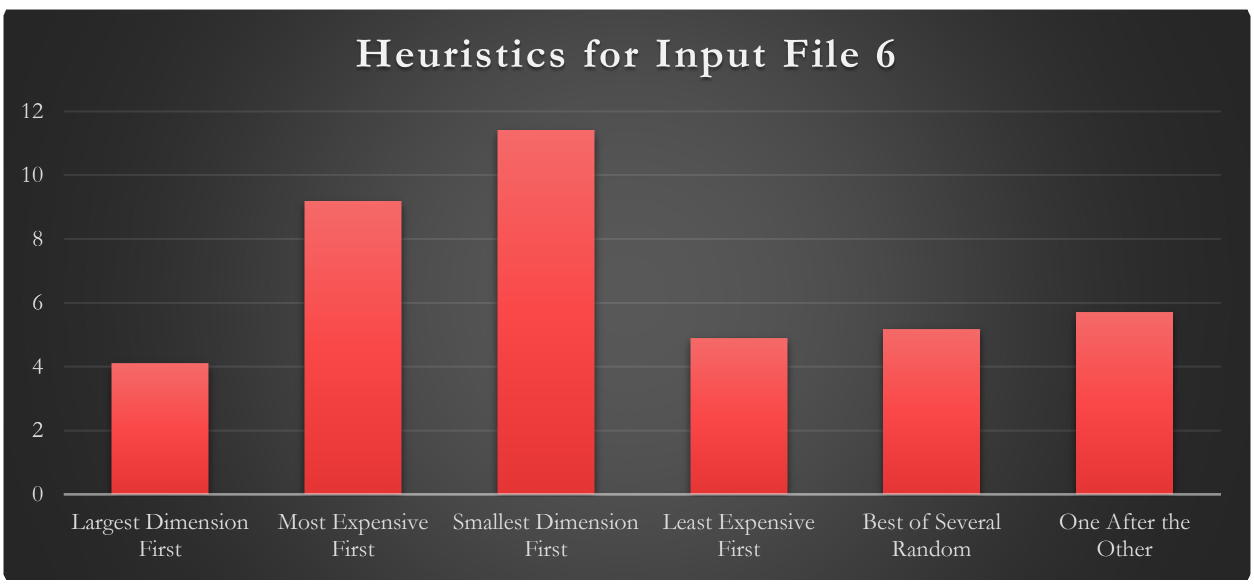
Figures 25 and 26 show the output of input file 5 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput6.txt

Figure 26

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput6.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput6.txt

Matrix Chain Loaded: 18 3 29 22 31 34 24 19 28 17 31 28 34 13 36 32 20 20 15 37

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (142712)      4.09245  |
| Most Expensive First           | (320049)      9.17782  |
| Smallest Dimension First       | (397470)      11.398   |
| Least Expensive First          | (170515)      4.88974  |
| Best of Several Random         | (180134)      5.16558  |
| One After the Other            | (198810)      5.70114  |
| Chain Multiplication           | (34872)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Figure 27



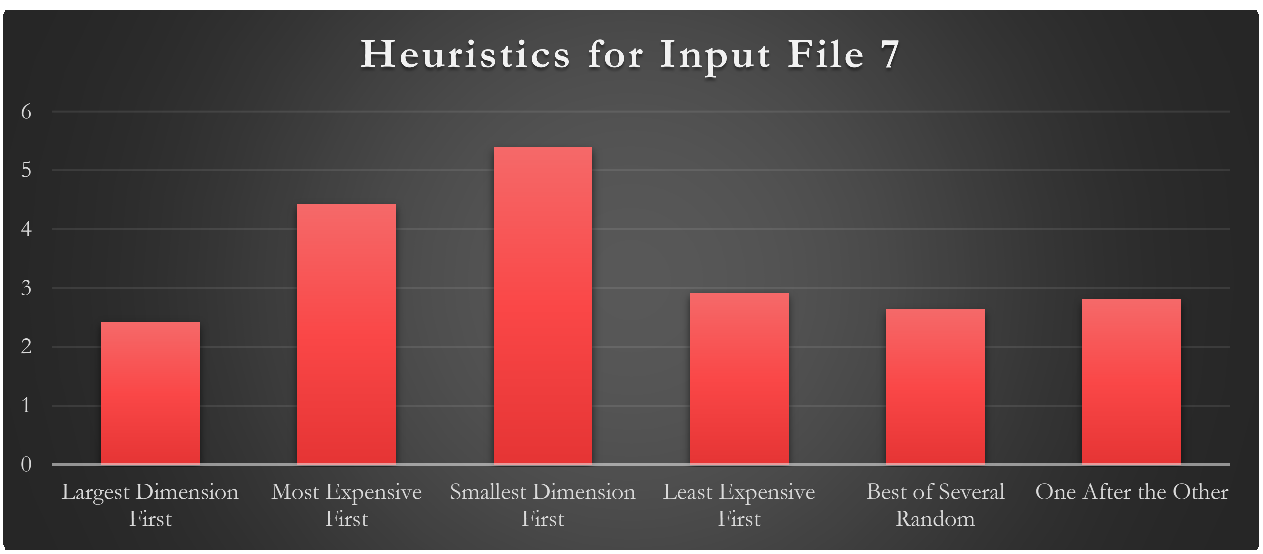
Figures 27 and 28 show the output of input file 6 and the results compared against each other respectively.

Heuristics from CS404SP17MatrixChainHeuristicsInput7.txt

Figure 28

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput7.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput7.txt

Matrix Chain Loaded: 18 6 15 17 15 15 21 19 13 34 27 13 22 32 26 39 22 14 16 37

----------------------------------------------------------
| Heuristics                      |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (129165)      2.42127  |
| Most Expensive First           | (235943)      4.42288  |
| Smallest Dimension First       | (288058)      5.39981  |
| Least Expensive First          | (155664)      2.91801  |
| Best of Several Random         | (140843)      2.64018  |
| One After the Other            | (149670)      2.80565  |
| Chain Multiplication           | (53346)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Figure 29



Figures 29 and 30 show the output of input file 1 and the results compared against each other respectively.

Conclusion (15%)

Pros and Cons: Strategy A – Remove Largest Dimension First

- Pros:
  - This heuristic consistently provided the most accurate result when compared against the optimal cost
  - The results are the exact same as the optimal approach when n=3 and very close when n=4
  - This heuristic is not effected by length, mean, or dimensions of the matrix chain
  - Cost less than the optimal result at only n^2
  - Easy to implement into code
  - Easy to manually compute
- Cons:
  - Requires a bit of effort to understand if explaining to a co-worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution

Pros and Cons: Strategy B – Do Most Expensive Matrix Multiplication First

- Pros:
  - The results are the exact same as the optimal approach when n=3
  - Cost less than the optimal result at only n^2
  - Easy to implement into code
- Cons:
  - This heuristic is clearly negatively effected by length, mean, or dimensions of the matrix chain and more so as n gets larger
  - This heuristic consistently proved to be one of the worst heuristics in terms of accuracy
  - Requires a bit of effort to understand if explaining to a co-worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution
  - Not easy to manually compute

Pros and Cons: Strategy C – Remove Smallest Dimensions First

- Pros:
  - The results are the exact same as the optimal approach when n=3
  - Cost less than the optimal result at only n^2
  - Easy to implement into code
  - Easier to manually compute
- Cons:
  - This heuristic is clearly negatively affected by length, mean, or dimensions of the matrix chain and more so as n gets larger
  - This heuristic consistently proved to be the worst heuristics in terms of accuracy
  - Requires a bit of effort to understand if explaining to a co-worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution

Pros and Cons: Strategy D – Do Least Expensive Matrix Multiplication First

- Pros:
  - The results are the exact same as the optimal approach when n=3
  - Cost less than the optimal result at only n^2
  - Easy to implement into code
  - This heuristic consistently proved to be the best heuristics in terms of accuracy
- Cons:
  - Requires a bit of effort to understand if explaining to a co-worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution
  - Harder to manually compute

Pros and Cons: Strategy E – Random Execution Tree

- Pros:
  - The results are the exact same as the optimal approach when n=3 and 4
  - Cost less than the optimal result at only n^2
  - This heuristic consistently proved to be in the middle of the range of heuristics in terms of accuracy
  - This heuristic is not affected by length, mean, or dimensions of the matrix chain
- Cons:
  - Requires a bit of effort to understand if explaining to a co-worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution
  - Hard to implement by hand
  - Difficult to code as multiple functions were needed
  - Difficult to maintain as it required multiple functions

Pros and Cons: Strategy F – Ignorant Approach

- Pros:
  - The results are the exact same as the optimal approach when n=3 and 4
  - Cost less than the optimal result at only n^2
  - This heuristic consistently provided to be in the middle of the range of heuristics in terms of accuracy however, that is just based on the random order and could easily vary
  - This heuristic is not affected by length, mean, or dimensions of the matrix chain
  - Easy to implement by hand
  - Easy to code
- Cons:
  - Requires a bit of effort to understand if explaining to a co worker
  - Is slightly difficult to explain to others but mostly because of their unfamiliarity with matrix chain
  - Does not provide the same result as the optimal solution
  - Difficult to code as multiple functions were needed
  - Difficult to maintain as it required multiple functions

Epilogue (5%)

Reflections

I tried for the longest time to try to implement my code using 2d arrays before finally moving on and using single and double vectors. I did not know that Yb, Ya, and Yw where supposed to be ratios of the cost verses the optimal cost. The decision to use vectors afterwards allowed the use of many features that simplified the process in the later steps allowing easy modification. I realized towards the end that I was computing the cost of the averages with the optimal method of the best case scenario which resulted in inaccurate results. I had to redo all the figures at the last moment.

Lessons Learned

I have a new appreciation for the efficiency of algorithms and heuristics. I have learned how to use and test heuristics and when to apply them. I learned how to effectively multiply chain matrices together in many different way. I have learned a new aspect of computer science where I needed to gather results and report findings based on algorithm implementation after being coded.

Next Time

Next time, I would defiantly have spent more time making the program more robust and allow the user to input into the terminal the matrix chain. I would also make a better UI where my charts always line up.

Lessons for Future Students

I would recommend that you start the day you get the project. Possibly cancel your spring break plans to finish this project one time. I ended up having spring break as part of the time to finish the project followed by 4 exams the week before the project was due. Do not attempt to go out of town for spring break especially if you too work full time. The different heuristics are very poorly explained so you will need plenty of time to get clarification.

Appendix (0%)

Program Listing and Exact Outputs as Generated by the Given Inputs

Figure 30

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: experiment


-------------------------------------------------------------------------------------------------
| Optimal Cost                |        n = 10                                                   |
|                             |        Mo[1,10]                                                 |
| Heuristics                  |         Yb                     Ya                      Yw       |
-------------------------------------------------------------------------------------------------
| Largest Dimension First     | (5807) 1.04857  -      (10173) 1.11448   -     (19788) 1.06182 |
| Most Expensive First        | (7336) 1.32467  -      (15004) 1.64373   -     (25881) 1.38876 |
| Smallest Dimension First    | (8324) 1.50307  -      (16978) 1.85999   -     (29396) 1.57738 |
| Least Expensive First       | (6597) 1.19122  -      (10872) 1.19106   -     (19644) 1.05409 |
| Best of Several Random      | (6002) 1.08378  -      (10125) 1.10922   -     (19500) 1.04636 |
| One After the Other         | (6804) 1.06931  -      (12711) 1.39253   -     (21330) 1.55206 |
-------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------
| Optimal Cost                |        n = 15                                                   |
|                             |        Mo[1,15]                                                 |
| Heuristics                  |         Yb                     Ya                      Yw       |
-------------------------------------------------------------------------------------------------
| Largest Dimension First     | (10920) 1.19084  -     (16509) 1.17351   -     (26718) 1.16464 |
| Most Expensive First        | (15363) 1.44294  -     (25535) 1.81511   -     (35883) 1.56414 |
| Smallest Dimension First    | (13168) 1.43599  -     (29141) 2.07144   -     (42749) 2.47748 |
| Least Expensive First       | (9384) 1.02334  -      (18005) 1.27985   -     (34451) 1.50172 |
| Best of Several Random      | (10659) 1.16238  -     (17244) 1.22576   -     (29261) 1.27549 |
| One After the Other         | (12558) 1.06659  -     (22451) 1.59589   -     (35568) 1.86572 |
-------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------
| Optimal Cost                |        n = 20                                                   |
|                             |        Mo[1,20]                                                 |
| Heuristics                  |         Yb                     Ya                      Yw       |
-------------------------------------------------------------------------------------------------
| Largest Dimension First     | (16064) 1.13551  -     (23768) 1.20766   -     (34279) 1.25804 |
| Most Expensive First        | (27852) 1.77198  -     (37714) 1.91626   -     (52930) 1.95227 |
| Smallest Dimension First    | (30890) 1.76676  -     (45082) 2.29064   -     (61789) 2.26765 |
| Least Expensive First       | (17510) 1.09492  -     (27941) 1.41969   -     (38439) 1.79983 |
| Best of Several Random      | (16180) 1.14371  -     (25635) 1.30253   -     (36295) 1.33202 |
| One After the Other         | (16359) 1.02295  -     (34537) 1.75484   -     (51425) 1.94497 |
-------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------
| Optimal Cost                |        n = 25                                                   |
|                             |        Mo[1,25]                                                 |
| Heuristics                  |         Yb                     Ya                      Yw       |
-------------------------------------------------------------------------------------------------
| Largest Dimension First     | (16902) 1.00992  -     (27631) 1.19838   -     (37849) 1.40661 |
| Most Expensive First        | (25303) 1.51189  -     (44210) 1.91742   -     (55879) 1.95415 |
| Smallest Dimension First    | (39501) 2.36024  -     (54271) 2.35378   -     (65166) 2.56277 |
| Least Expensive First       | (19793) 1.18266  -     (31469) 1.36483   -     (46772) 1.7399  |
| Best of Several Random      | (19475) 1.16366  -     (30438) 1.32012   -     (42393) 1.48253 |
| One After the Other         | (22200) 1.22227  -     (41865) 1.81572   -     (58395) 2.04214 |
-------------------------------------------------------------------------------------------------
```

Heuristics from CS404SP17MatrixChainHeuristicsInput1.txt

Figure 31

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput1.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput1.txt

Matrix Chain Loaded: 18 6 28 28 23 26 29 27 23 24 25 25 27 26 24 29 22 29 26 36

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (235326)      3.19597  |
| Most Expensive First           | (385002)      5.22873  |
| Smallest Dimension First       | (351728)      4.77684  |
| Least Expensive First          | (215670)      2.92903  |
| Best of Several Random         | (228246)      3.09982  |
| One After the Other            | (212256)      2.88266  |
| Chain Multiplication           | (73632)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Heuristics from CS404SP17MatrixChainHeuristicsInput2.txt

Figure 32

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput2.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput2.txt

Matrix Chain Loaded: 18 4 23 23 25 29 24 24 22 23 26 25 26 27 23 24 28 29 23 38

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (241289)      5.19258  |
| Most Expensive First           | (388474)      8.36003  |
| Smallest Dimension First       | (309019)      6.65015  |
| Least Expensive First          | (201918)      4.34531  |
| Best of Several Random         | (208884)      4.49522  |
| One After the Other            | (198450)      4.27068  |
| Chain Multiplication           | (46468)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Heuristics from CS404SP17MatrixChainHeuristicsInput3.txt

Figure 33

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput3.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput3.txt

Matrix Chain Loaded: 18 2 26 24 29 22 27 26 27 28 25 28 24 27 26 28 28 24 26 38

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (250626)      9.93444  |
| Most Expensive First           | (419140)      16.6141  |
| Smallest Dimension First       | (374892)      14.8602  |
| Least Expensive First          | (215676)      8.54907  |
| Best of Several Random         | (235150)      9.32099  |
| One After the Other            | (215676)      8.54907  |
| Chain Multiplication           | (25228)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: _
```

Heuristics from CS404SP17MatrixChainHeuristicsInput4.txt

Figure 34

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput4.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput4.txt

Matrix Chain Loaded: 18 4 19 37 14 28 28 14 16 16 21 37 21 38 13 19 31 26 13 35

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (108880)      2.86045  |
| Most Expensive First           | (290474)      7.6312   |
| Smallest Dimension First       | (382512)      10.0492  |
| Least Expensive First          | (160462)      4.21558  |
| Best of Several Random         | (115369)      3.03092  |
| One After the Other            | (161316)      4.23802  |
| Chain Multiplication           | (38064)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: _
```

Heuristics from CS404SP17MatrixChainHeuristicsInput5.txt

Figure 35

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput5.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput5.txt

Matrix Chain Loaded: 18 6 20 18 15 23 24 30 21 26 20 37 23 22 31 36 34 19 20 35

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (197290)      2.87981  |
| Most Expensive First           | (308824)      4.50785  |
| Smallest Dimension First       | (365135)      5.32982  |
| Least Expensive First          | (198728)      2.9008   |
| Best of Several Random         | (223464)      3.26187  |
| One After the Other            | (196344)      2.866    |
| Chain Multiplication           | (68508)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Heuristics from CS404SP17MatrixChainHeuristicsInput6.txt

Figure 36

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput6.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput6.txt

Matrix Chain Loaded: 18 3 29 22 31 34 24 19 28 17 31 28 34 13 36 32 20 20 15 37

----------------------------------------------------------
| Heuristics                     |  Cost                  |
----------------------------------------------------------
| Largest Dimension First        | (142712)      4.09245  |
| Most Expensive First           | (320049)      9.17782  |
| Smallest Dimension First       | (397470)      11.398   |
| Least Expensive First          | (170515)      4.88974  |
| Best of Several Random         | (180134)      5.16558  |
| One After the Other            | (198810)      5.70114  |
| Chain Multiplication           | (34872)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```

Heuristics from CS404SP17MatrixChainHeuristicsInput7.txt

Figure 37

```
Optimal Matrix Chain and Heuristics

--------------------------------------------------------------
This program can test a matix chain with the following heuristic strategies:
Strategy A - Remove Largest Dimension First
Strategy B - Do Most Expensive Matrix Multiplication First
Strategy C - Remove Smallest Dimensions First
Strategy D - Do Least Expensive Matrix Multiplication First
Strategy E - Random Execution Tree
Strategy F - Ignorant Approach
--------------------------------------------------------------

You can either import a matrix chain from a file or see the Matrix Chain Heuristic Experiment.


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery: file
Please enter the file name you wish to import the matrix chain: Heuristics from CS404SP17MatrixChainHeuristicsInput7.txt
Please enter the file name you wish to import the matrix chain: Please enter the file name you wish to import the matrix chain:
Found CS404SP17MatrixChainHeuristicsInput7.txt

Matrix Chain Loaded: 18 6 15 17 15 15 21 19 13 34 27 13 22 32 26 39 22 14 16 37

----------------------------------------------------------
| Heuristics                      |  Cost                    |
----------------------------------------------------------
| Largest Dimension First         | (129165)      2.42127  |
| Most Expensive First            | (235943)      4.42288  |
| Smallest Dimension First        | (288058)      5.39981  |
| Least Expensive First           | (155664)      2.91801  |
| Best of Several Random          | (140843)      2.64018  |
| One After the Other             | (149670)      2.80565  |
| Chain Multiplication            | (53346)       1        |
----------------------------------------------------------


Please enter the "FILE" for file entery or enter "EXPERIMENT" for manual entery:
```