CS 124 Programming Assignment 2 Writeup

Our goal in this experiment was to implement an optimized variant of Strassen's algorithm for matrix multiplication and use it to determine the number of triangles in five types of random, undirected graphs of size 1024 vertices, wherein for each graph, each edge of the graph was included with probability $p = 0.01, 0.02, 0.03, 0.04$, or 0.05 respectively. We reasoned that, for any graph, the expected number of triangles is $\binom{1024}{3}p^3$, since there are $\binom{1024}{3}$ possible vertex triplets and the probability that all three edges connecting a given triplet are included in the graph is p^3. Therefore, we expected that our results would be relatively close to these values for each p, with some room for experimental variability.

In order to create an optimized variant of Strassen's algorithm, we set out to find the largest matrix dimension size (n_0) at which matrix multiplication between two n x n matrices becomes more efficient with the conventional method than with Strassen's algorithm. By doing this, we could implement Strassen's algorithm such that if it is called with dimension size greater than n_0, then it would run normally, and if it is called with dimension size less than or equal to n_0, the algorithm would switch to the conventional method, thereby optimizing the overall run-time.

We began the construction of our variant of Strassen's algorithm by first analytically finding a theoretical n_0. To do this, we assumed that each arithmetic operation between two real numbers had a cost of 1, and then computed the total cost for both multiplication methods as a function of n.

- For the conventional method, we reasoned that, in order to compute one entry in the matrix product, we must do n multiplications (one for each row/column entry in the two matrix factors) and $n - 1$ additions (to compute the sum of the n entry products). Because the resulting matrix has n^2 entries, this gives a total cost of:

$$C(n) = n^2(2n - 1)$$

- For Strassen's algorithm, we first reasoned that its recurrence relation is

$$T(n) = 7T(n/2) + cn^2,$$

since the algorithm involves 7 multiplications of $n/2$ x $n/2$ matrices and various matrix additions/subtractions, which are of order n^2. By the Master's Theorem, the run-time must then be $O(n^{\log 7})$. Therefore, in order to solve the recurrence and find the cost of Strassen's algorithm as a function of n, we first guessed that the form of the equation would be

$$C(n) = a \cdot n^{\log 7} + b \cdot n^2$$

To find a and b, we started by manually conducting the multiplication with two 2 x 2 matrices. In this case, each sub-matrix is simply a real number, so computing P_1 through P_4 each took 2 operations (1 addition/subtraction and 1 multiplication) and computing P_5 through P_7 each took 3 operations (2 additions/subtractions and 1 multiplication). Then, to compute the entries in the matrix product, quandrants 1 and 4 each required 3 operations while quadrants 2 and 3 each required 1 operation. This gave a total cost of:

$$C(2) = 4(2) + 3(3) + 2(3) + 2(1) = 25$$

Then, we imagined conducting the multiplication with two 4 x 4 matrices, assuming that 2 x 2 matrix multiplication takes 25 steps. In this case, each sub-matrix is a 2 x 2 matrix, so computing P_1 through P_4 would take 29 steps (4 operations for the matrix addition/subtraction and 25 for the multiplication), and computing P_5 through P_7 would take 33 steps (8 for the two matrix additions/subtractions and 25 for the multiplication).

Then, computing quadrants 1 and 4 would take 3 matrix additions/subtractions, giving a total of 12 operations for each, and computing quadrants 2 and 3 would take 1 matrix addition/subtraction, giving a total of 4 operations for each. This would give a total cost of:

$$C(4) = 4(29) + 3(33) + 2(12) + 2(4) = 247$$

Using these values, we knew that a and b must be such that:

$$C(2) = a \cdot 2^{\log 7} + b \cdot 2^2 = 25$$
$$C(4) = a \cdot 4^{\log 7} + b \cdot 4^2 = 247$$

Solving this system of equations gave $a = 7$ and $b = -6$, giving a cost function of:

$$C(n) = 7n^{\log 7} - 6n^2$$

Therefore, in order to find n_0, we wanted to find n such that:

$$7n^{\log 7} - 6n^2 > n^2(2n - 1)$$

Plugging a few values of n into this inequality showed that the largest integer for which this is true is 654, giving us a theoretical cross-over point of:

$$n_0 = 654$$

Next, we implemented conventional matrix multiplication and Strassen's algorithm using C++, in an effort to find an empirical cross-over point for the optimized algorithm. In order for Strassen's algorithm to work for all dimension sizes, we designed the Matrix struct such that, if at any point when dividing a matrix into sub-matrices, Strassen's algorithm encounters an odd dimension, or if the original matrix factors have odd dimensions, we treat the matrix as if there is an extra row and column to the bottom and right of the matrix wherein each entry is 0. In order to conserve space, we do this without allocating new memory for these entries, but instead merely using them for the computations, while keeping track of the actual dimensions of the original matrix so that we could reconstruct the resulting matrix product without the extraneous 0 values. Additionally, with regard to run-time, we figured that this would be the optimal way to generalize the algorithm because we would only ever pad at most $\log n$ rows/columns with 0s. If we were to instead pad the original matrix with 0s until its dimensions were the next highest power of two, we could, in the worst case, end up nearly doubling the size of the matrix with extraneous 0 values, which could increase the algorithm's run-time. Finally, to test the correctness of both algorithms, we simply created pairs of equally-sized matrices with randomly generated integer entries, conducted the multiplication, and then compared the results against WolframAlpha's matrix multiplication calculator.

After implementing and testing both algorithms, we set out to find n_0 by recording the run-time of both algorithms with different dimension sizes and determining the largest dimension size for which Strassen's algorithm takes longer than the conventional. Knowing that the theoretical value is $n_0 = 654$, we guessed that the actual value would likely lie between $n = 600$ and 700. After testing a few n values in this range, we found that the conventional method became faster once the matrices were of size 658 x 658, giving an empirical cross-over point of:

$$n_0 = 658$$

We were surprised at how close this value was to the theoretical, since our above analysis was based on the simplifying and highly idealized assumption that the only costs with regard to efficiency were arithmetic operations and that each one has a cost of 1. However, we reasoned that the way in which we handled Strassen's algorithm for general n helped optimize the run-time

and keep n_0 close to the theoretical value. Therefore, in order to implement the optimized variant of the algorithm, we simply added a conditional to Strassen's algorithm checking whether the dimension size was greater than 658 - if it was, we would run (or continue running, if called recursively) Strassen's algorithm, otherwise, we would switch to the conventional method.

Now that we had an optimized Strassen's algorithm, we could use it to conduct our experiment. We accomplished this by first creating, for each $p = 0.01, 0.02, 0.03, 0.04$, and 0.05, a 1024 x 1024 matrix A, which would serve as the adjacency matrix for the randomly generated graph, and then filling the entires (edge weights) as either 1 (with probability p) or 0 (with probability 1 - p) using the mt19937 pseudo-random number generator and STL's Bernoulli distribution function. Then we computed A^3 using the optimized Strassen's algorithm and then found the number of triangles in the graph by adding the entries in the diagonal and dividing by 6. For each p value, we performed 5 iterations of this process (in order to mitigate experimental variability), and then took the averages of these iterations, which are shown as the observed values in the chart below:

p	Observed	Expected
0.01	175.60	178.43
0.02	1447.60	1427.46
0.03	4748.20	4817.69
0.04	11510.00	11419.71
0.05	22142.40	22304.13

As shown above, the number of triangles we found was very close to the amount that we expected for each p. While all of the values are either slightly higher or slightly lower than their expectation, we attributed this to mere experimental variability and reasoned that the difference probably isn't stark enough to be statistically significant.

Ultimately, we learned many things throughout this experiment, especially with regard to efficiency considerations. For instance, when determining how to generalize Strassen's algorithm, we had to consider the effects that different padding methods would have on the run-time and space usage of the algorithm. Additionally, we saw an interesting way in which matrix multiplication and the adjacency matrix representation could be used to solve graph problems. Finally, and perhaps most importantly, by analytically and empirically determining the cross-over point between Strassen's algorithm and the conventional matrix multiplication algorithm, we saw how the oft-theoretical contribution of asymptotically faster algorithms can be reconciled with the potential practicality of asymptotically slower ones.