CS 124 Programming Assignment 2: Spring 2022

Your name(s) (up to two): Christy Jestin, Emil Liu

Collaborators: (You shouldn't have any collaborators but the up-to-two of you, but tell us if you did.)

No. of late days used on previous psets: 7
No. of late days used after including this pset: 7

Analytic Approach

We will utilize the convention where odd matrices are padded with an extra row and column of zeroes on the bottom and right side. Thus for an input of dimension $n \times n$, the recurrence has dimensions $\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil$. Assuming that the cost of any single arithmetic operation is 1 and all other operations are free, we know that the recurrence for Strassen's algorithm can be written as follows:

$$T_1(n) = 7T_1(\lceil \frac{n}{2} \rceil) + 18(\lceil \frac{n}{2} \rceil)^2$$

Strassen's algorithm breaks down the original multiplication problem into 7 multiplication subproblems of dimension $\lceil \frac{n}{2} \rceil$ and 18 additions of matrices with $(\lceil \frac{n}{2} \rceil)^2$ elements each. One multiplication is needed per P_i. 10 additions are needed to calculate P_1 through P_7, and 8 additions are needed to construct the four quadrants of the output from P_1 through P_7. We also know that the time complexity for the conventional algorithm can be written as follows:

$$T_2(n) = (2n-1)n^2$$

With the conventional algorithm, we loop through each of n^2 entries of the output matrix, and for each of those entries we need to take the dot product of a row and a column. This takes $n-1$ additions and n multiplications for a cost of $2n-1$. We multiply by the number of entries to get $(2n-1)n^2$.

Note that we are tasked with finding the cross-over point, n_0, such that for matrices above this size, we recurse at least one more time with Strassen's algorithm, and for matrices below this size, we switch to the conventional algorithm.

It would be incorrect to solve for the crossover point by setting $T_1(n)$ equal to $T_2(n)$ since we would be comparing pure Strassen's algorithm that goes all the way down to a base case of 1 (T_1) to the conventional algorithm (T_2). Instead we'd like to model the choice of doing at least one more Strassen recursion versus switching to conventional matrix multiplication. Let $M(n)$ denote the time needed to run modified Strassen's on an n dimensional input with an optimal crossover point n_0. We always have the choice of doing no more Strassen recursions and just doing conventional multiplication, so $M(n) \leq (2n-1)n^2$ for all n.

This is an upper bound, so if $7(\frac{n}{2})^2(n-1)+18(\frac{n}{2})^2$ is smaller than $(2n-1)n^2$, then we should definitely do at least one more Strassen recursion. In addition, since we don't use Strassen's below the crossover point, $7(\frac{n_0}{2})^2(n_0-1)+18(\frac{n_0}{2})^2$ is the correct expression for $M(n_0)$. Note that these expressions only hold for even n, but the form is very similar for odd n.

Additionally, since n_0 is the crossover point, doing one level of Strassen and then conventional should be very similar to doing only conventional. Thus we solve for n_0 by setting

$7(\frac{n}{2})^2(n-1)+18(\frac{n}{2})^2$ equal to $(2n-1)n^2$.

$$7(\frac{n_0}{2})^2(n_0-1)+\frac{18}{4}(n_0)^2 = 2n_0^3 - n_0^2$$
$$\frac{1}{4}n_0^3 - \frac{15}{4}n_0^2 = 0$$

We obtain that $n_0 = 15$. Above and at this point, we want to use Strassen's and below, we want to use conventional. Since we were using the even forms, this is the crossover point for even n.

Now we'll go back and use the odd forms to find the corresponding crossover point. When the matrix is odd, we pad our matrix with a column of 0s and a row of 0s at the end, and $\lceil\frac{n}{2}\rceil = \frac{n+1}{2}$. Therefore, the revised equation is:

$$T(n_0) = 7(\frac{n_0+1}{2})^2(n_0) + 18(\frac{n_0+1}{2})^2$$

And the new equation to solve is:

$$7(\frac{n_0+1}{2})^2(n_0) + 18(\frac{n_0+1}{2})^2 = 2(n_0)^3 - (n_0)^2$$

Note that the right-hand side of the equation is still the cost of the standard algorithm as we do not need to pad for the standard. The solution to this $n = 37.17$. Above this point, we want to use Strassen's and below, we want to use conventional.

Thus we see that when using Strassen's on even dimensions, the cross-over point is significantly different than using Strassen's on odd dimensions. Intuitively, the odd cross-over point is higher than the even one because of the fact that we have to pad our matrix with 0s, which means that it costs more to process.

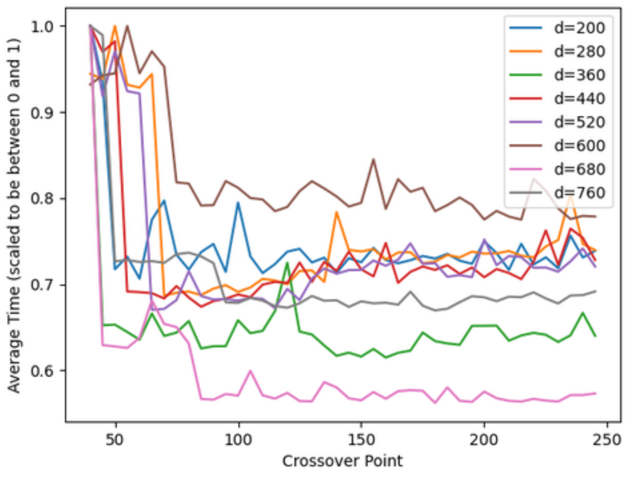
Experimental Approach

We implemented the standard matrix multiplication algorithm and Strassen's algorithm in C++. We optimized our standard multiplication algorithm such that our for loop nesting is i, then k, then j where i is the row index of the first matrix, j is the column index of the second matrix, and k is the index along the row/column. Looping through a row is faster than looping over a column since the memory is contiguous, and our ordering is an improvement since the standard order i, j, k uses more iterations down a column.

For our implementation of Strassen's algorithm, we made sure to pass in matrices by reference when possible - as in, we did not unnecessarily copy matrices, which are large blocks of data, and instead we altered them in place when possible. For example, in our add, subtract, pad, and unpad functions, we use a reference to the matrices by passing in the memory address of the matrix. These optimizations were possible in C++. Furthermore, we dealt with the case when d is odd by padding the matrix with zeroes (adding one column and one row), and then unpadding it at the end of the algorithm. This works because, at the end, the last row and the last column of the result will all still be 0s from the way that matrix multiplication works, and the other values will remain unaffected because adding by 0.

We also considered padding the matrix of size d with zeros until its size is a power of 2 (instead of just adding a single row and column to matrices with odd d). However, we thought that this probably won't be an improvement since for many values of d, the padded matrix would became much larger than they needed to be, which resulted in a lot of extra memory usage. In addition, we would have many multiplications involving matrices of all zeroes which seems pointless.

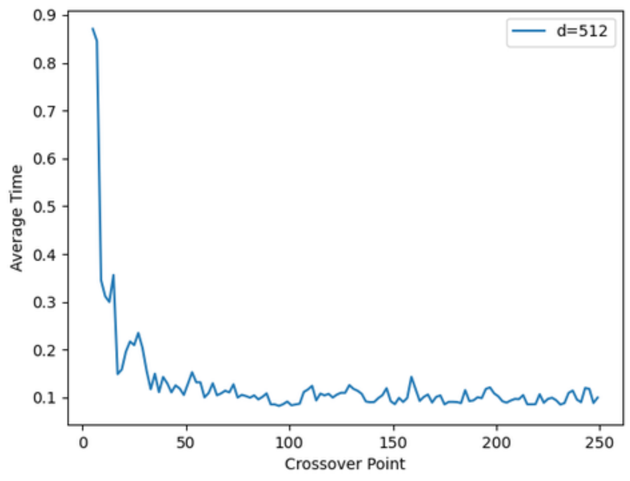
In order to find the crossover point experimentally, we tested different crossover points for randomly generated matrices of the same dimension. All matrix entries were either 0 or 1 with equal probability. We tried values of d between 200 and 760, and we incremented the crossover point in steps of 5. We measured how long it takes to run the algorithm (the modified Strassen's that recurs until the crossover point) over 5 trials and took the average. Then to plot all the data on the same graph, we scaled all the time values for a certain dimension d by the max value for that dimension. Thus the time values were all less than 1, but the graph still showed the relationship between the average times for different crossover points. The following is the graph of Crossover Point versus Time Taken:



In order to determine the crossover point, we look out for the minimum time taken for a given dimension. This minimum point marks the crossover because this point marks the ideal combination of Strassen and then the standard matrix multiplication algorithm. We can see that for the various tested dimensions, the cross-over point varies slightly - they all, however, range around the 80 to 90 area. We can see that after this point, the time starts to slowly go up or level off, and, when the crossover point passes d, then the algorithm just uses standard multiplication for the entire problem. Note that we didn't need to use any odd values of d because as long as d is not a power of 2, we will call Strassen on an odd number at some point in the recursion. Furthermore, even though we are taking the average time to calculate matrix products over 5 trials, as we can see in the graph, there is still a lot of noise that results in some spikes for how long it takes. The spikes are presumably related to odd and even numbers and random number generation.

This crossover point is significantly higher than our expected theoretical answer from part one. However, this makes sense since our theoretical calculation didn't consider the time needed to allocate/deallocate memory, assign values, construct submatrices etc. All of these implementation details add to the n^2 coefficient in practice.

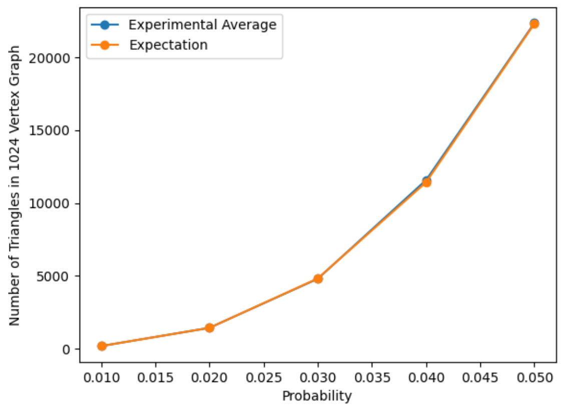
Furthermore, as a more specific case study, observe the following graph of $d = 512$, where the X-axis is the crossover points in increments of 2, against the average time across 5 trials.



Notice that when the crossover point is very low - such as 1, the algorithm takes very long because it uses Strassen's all the way down to a base case of 1. When we increase the crossover point, there is a rapid drop in time. Reaching a local minimum around the crossover point being set to 100. After this point, our time increases slightly, which fits with our expected model of how crossover points should work.

Triangles in Random Graphs

The following plot shows the experimental average for the number of triangles over 5 random trials, where the X-axis is the probability that we inputted and the Y-axis is the number of triangles (over 5 trials).



As we can see in our plot of the experimental average versus the theoretical expectation, our code returns an average number of triangles that is very similar to the expected number of triangles for the given probabilities.