

# CS 124 Programming Assignment 2 Write Up

Vladimir Petrov and Arjun Puri

29 March 2022

## Introduction

Conventional matrix multiplication over two matrices  $A, B \in \mathbb{R}^{n \times n}$  calculates entry  $i, j$  of the product  $AB$  as follows:

$$AB[i][j] = \sum_{k=0}^{n-1} A[i][k]B[k][j].$$

Assuming common operations like addition, multiplication, array access, etc are  $\Theta(1)$ , we see that this algorithm takes  $\Theta(n)$  over  $n^2$  entries, yielding a time complexity of  $\Theta(n^3)$ .

However, Strassen came up with a very clever algorithm to improve this  $\Theta(n^3)$  time complexity to  $\Theta(n^{\log_2(7)})$ . The basic idea is to perform 7 matrix multiplications on  $\frac{n}{2} \times \frac{n}{2}$  submatrices and add and subtract them in clever ways. Assuming the relevant (constant in  $n$ ) number of matrix additions between the submatrix products take time  $\Theta(n^2)$ , Strassen's algorithm has recursive runtime of

$$T(n) = 7T(n/2) + \Theta(n^2).$$

The master method then yields our claimed time complexity of  $\Theta(n^{\log_2(7)})$ .

However, the asymptotic notation above may be hiding some very large constant factors in Strassen's algorithm; for small matrices, conventional matrix multiplication may indeed be faster. Traditional Strassen's recurses to a base case of  $1 \times 1$  matrices, but recursing down to an experimental cutoff point at which point we switch to traditional matrix multiplication will yield performance benefits.

The purpose of our code will be to experimentally determine what this cutoff point is using randomly generated integer matrices with entries between 0 and 4. We will also use our Strassen's implementation to calculate the number of triangles in randomly generated graphs of size 1024, which can be done via matrix multiplication of the adjacency matrix (the theory of which is described later).

## Analytically Estimating Cutoff

We first analytically estimate the crossover point between Strassen's and traditional matrix multiplication.

The problem statement seeks to find the value of  $n_0$  for which Strassen's runs with minimum time. We estimate this by finding the cutoff  $n_0$  at which Strassen's first runs faster than traditional matrix multiplication. (These are not necessarily the same, but they should be close. Further discussion is given in the experiments section.)

Let  $T(n)$  denote the time for traditional matrix multiplication. We then see that traditional multiplication runs  $n$  multiplications over  $n^2$  terms, yielding total time  $n^3$  from multiplication. Moreover, each  $n^2$  term runs through  $n - 1$  additions, yielding  $n^2(n - 1)$  time through addition. We thus find that

$$T(n) \approx n^3 + n^2(n - 1) = 2n^3 - n^2.$$

Let  $S(n)$  now denote the time for Strassen's matrix multiplication with a threshold at  $n_0 - 1$ . We next assume that our Strassen's is implemented such that all calculations are done in-place (as we do in our code) to simplify time considerations from memory copying. In this case, the total time is again the number of multiplications and additions performed. Note again the recursion

$$S(n) = 7S(n/2) + f(n)$$

for  $n \geq n_0$  and for  $n < n_0$ .

$$S(n) = 2n^3 - n^2.$$

We seek to estimate the functional form of  $f(n) \in \Theta(n^2)$  to exactly solve this recursion.

Note that Strassen's first calculates 10 intermediate sums, each of which does  $n^2/4$  additions. We then perform the recursive step, which is already baked-into the recursion. It next computes 4 intermediate sums. Two of these perform  $3(n^2/4)$  additions, while two perform  $n^2/4$  additions. This yields

$$f(n) = 10(n^2/4) + 2 \cdot 3(n^2/4) + 2(n^2/4) = 9/2 n^2.$$

We seek to find the value of  $n_0$  such that  $S(n_0) < T(n_0)$ , so that Strassen's is faster than traditional multiplication for all input greater than the threshold. Note that  $\frac{n_0}{2} \leq n_0 - 1$  for  $n_0 \geq 2$ , which we assume. Thus

$$S(n_0) = 7S(n_0/2) + 9/2(n_0)^2 = 7(2(n_0/2)^3 - (n_0/2)^2) + \frac{9}{2}n_0^2.$$

Mathematica solves

$$7(2(n_0/2)^3 - (n_0/2)^2) + \frac{9}{2}n_0^2 < 2n_0^3 - n_0^2$$

to yield  $n_0 > 15$ . Thus the first point at which Strassen's outperforms traditional multiplication is at a threshold of 16.

However, our implementation of Strassen's, in order to stay in-place, ends up undoing the 10  $n^2/4$  additions on the intermediate matrices (as we must undo some of the arithmetic operations performed in-place to preserve the appropriate summands for final calculation). This yields an additional  $10n^2$  additions, giving  $f(n) = 7n^2$ . This yields the recursion  $S(n) = 7S(n/2) + 7n^2$ , which yields the inequality

$$7(2(n_0/2)^3 - (n_0/2)^2) + 7n_0^2 < 2n_0^3 - n_0^2,$$

which results in  $n_0 \geq 26$ . Since our implementation only depends on powers of 2 and recurses always to a base-case power of 2, our personal theoretical cutoff should be the next greatest power of 2: 32. (Our implementation pads on extra 0's for non powers of 2, so we always recurse down to a base-case that is a power of 2.)

## Implementations and Experimental Crossover Point Result

Our implementation of trivial matrix multiplication is done in the standard manner.

We implement Strassen's using the standard algorithm described in the lecture notes. We optimize to prevent needless memory copying by implementing the algorithm in-place (except for new memory allocation for our result matrix); we also allow for variable base case by setting a local variable *split\_point* which determines when we switch to standard matrix multiplication. For non powers of 2, we simply pad extra 0's onto the matrices appropriately (as discussed earlier).

We present two notions of crossover point. One is the  $n_0$  at which Strassen's first outperforms traditional matrix multiplication, as was done in the theoretical analysis. To analyze this, we programmed a helper *find\_first\_cross* which found the cutoff  $n_0$  when Strassen's was first faster than traditional matrix multiplication. This helper binary searches through all cutoff recursive thresholds to find the smallest one that is faster than traditional matrix multiplication. The binary search is justified by the theoretical analysis above (as we should be faster than traditional matrix multiplication past a certain point, and slower for all thresholds below), and it ends up holding in some experiments we did as well (which timed Strassen's faster past a threshold, and slower below a given threshold). Our results are shown below.

Dimension	Cutoff
4	4
8	8
16	16
32	32
64	34
128	16
256	32
512	32
1024	16

This is remarkably consistent with our theoretical result of 32 for our implementation. Indeed, we suspect the difference between 16 and 32 cutoffs is so small in practice that other random considerations come into effect (hence the 16 cutoff for 1024).

Our second notion is the  $n_0$  at which Strassen's runtime is minimized. To find this, we implement a function *find\_optimal\_split* which takes in a local variable *dimension*. It then initializes two random matrices of the appropriate dimension with random values between 0 and 4. It then iterates through all possible *split\_points* at which we can switch to traditional matrix multiplication (i.e., powers of 2 from 1 to *dimension*; non powers of 2 are equivalent to cutoffs at the next lower power of 2) and returns the split that takes the minimum amount of time to execute. This is then the second experimental crossover point.

Results are shown below.

Dimension	Cutoff
4	4
8	8
16	16
32	32
64	64
128	64
256	64
512	32
1024	32

It seems that, on our system, Strassen’s is optimized at around a cutoff of  $32 \times 32$  matrices. We believe that the different cutoff values for some small matrices (for example the  $n = 256$  case) is likely due to vanishingly small time differences at such small matrices regardless of optimization, so that randomness can lead to one cutoff being faster than another. Moreover, cache concerns/low-level explanations could also explain discrepancy at lower values of  $n$ . This is also why we rely on the largest result of  $n = 1024$  as our decided cutoff.

The result here is practically consistent with our theoretical analysis. There are things to take into account, though.

First, we approximated that the minimizer of Strassen’s runtime is the first recursive base case at which Strassen’s runs faster than traditional matrix multiplication (i.e., our first notion was what we calculated analytically). A priori, this is not necessarily the case, but experimentally they ended up being quite close, justifying this assumption.

We also assumed that multiplication is as expensive as addition on our architecture when analyzing the theoretical performance. This is also not necessarily the case, but the assumption was consistent with theoretical results. Indeed, running a test comparing the runtime of multiplication and addition on our architecture yields practically identical results.

## Graph Results

We turn next to use our matrix multiplication algorithm to calculate the number of triangles of random graphs on  $n = 1024$  vertices.

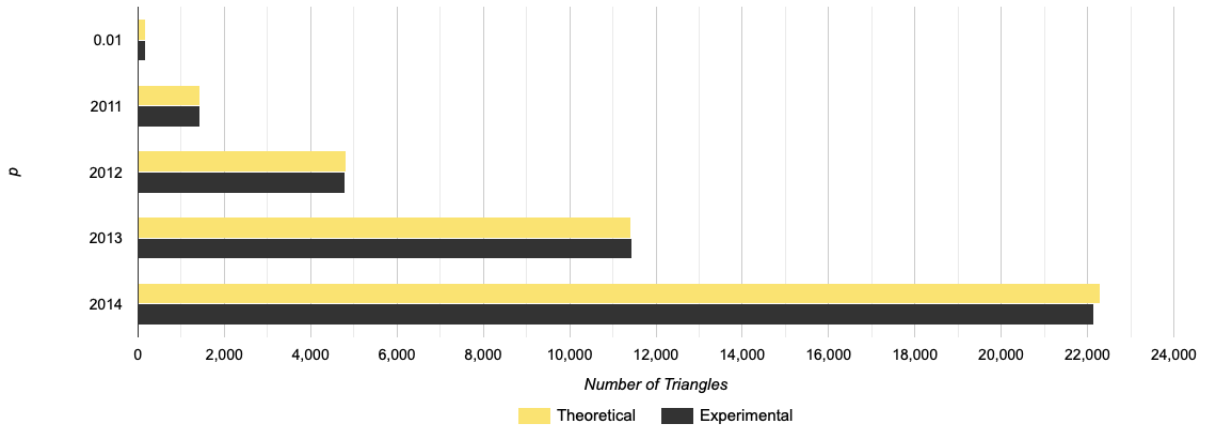
On a given graph  $G$  with adjacency matrix  $A$ , the number of triangles is given by  $\text{tr}(A^3)/6$ , where  $\text{tr}(A^3)$  denotes the sum of diagonal entries of  $A^3$ . We thus initialize such adjacency matrices by including a given edge with probability  $p$ .

We thus initialize such a random adjacency matrix by including a given edge with probability  $p$ . We calculate  $\text{tr}(A^3)/6$ , repeat 5 times, and average. We then compare our number of triangles to the theoretical average, which is easily calculated via linearity of expectation to be  $\binom{1024}{3}p^3$ .

A table of our experimentally determined calculations against theoretical calculations is given below.

p	Experimental	(Approximate) Theoretical
0.01	174.2	178.433
0.02	1430.2	1427.46
0.03	4795.2	4817.69
0.04	11435.2	11419.7
0.05	22145.4	22304.1

The bar chart below summarizes our results.



## Concluding Remarks

Our experimental crossover of 32 relies on the fact that we are multiplying integer matrices; speed of multiplication vs addition over floating point numbers likely has a different ratio and could result in a different optimal crossover.

Strassen's algorithm is indeed faster than naive multiplication using the base-case optimizations above, even for  $n$  that are arguably small (e.g.  $n = 256$ ). This indicates that Strassen's is NOT practically useless, as was implied by the analysis quoted in the proset. We are curious whether such practical adjustments can be made for matrix algorithms that are asymptotically better than even Strassen's.

Strassen's on non-square matrices would require a totally different analysis and likely have cutoffs dependent on appropriate matrices.