CS 1240 Progset 2

Ipek Eryilmaz & Rafael Jacobsen

March 2025

Task 1: Analytical Solution for the Cutoff

Runtime: Conventional Matrix Multiplication

For each entry i, j of the product matrix, matrix multiplication

$$(XY)_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$

requires n multiplications and $n-1$ additions for each entry. Since we take each addition and multiplication to take unit time, the conventional matrix multiplication algorithm takes

$$T_{\text{conventional}}(n) = n^2(2n-1) = 2n^3 - n^2$$

for an $n \times n$ matrix. This is implemented in the python code as follows:

Algorithm 1 Conventional multiplication (A, B)

```
n =dimension of A and B
Output=zeroed array of size n × n
for i ∈ {0, ..., n − 1} do
    for j ∈ {0, ..., n − 1} do
        a = A[i, j]
        for k ∈ {0, ..., n − 1} do
            output[i, k]+ = a · B[j, k]
        end for
    end for
end for
```

The algorithm is represented in this way to minimize caching; it only has to cache each element in the first matrix one time. Originally we used np.sum but this led to problems

further down the line because of advanced algorithms in numpy code that lead to unexpected timings. Naively implementing the algorithm by caching each element in both arrays n times led to a much slower algorithm and a much higher crossover point.

Recursion Equation: Strassen's

Recall that given two $(2k) \times (2k)$ matrices X and Y of the form

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \text{and} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Strassen's runs by computing 7 matrix products of size k and combining them. In one recursion of the algorithm, the 7 matrices require one matrix multiplication each and the following number of matrix additions or subtractions:

- $P_1 = A(F - H)$
- $P_2 = (A + B)H$
- $P_3 = (C + D)E$
- $P_4 = D(G - E)$
- $P_5 = (A + D)(E + H)$
- $P_6 = (B - D)(G + H)$
- $P_7 = (C - A)(E + F)$

Since the calculation of the 7 matrices requires 10 matrix additions/subtractions and 7 matrix multiplications of size $n \times n$, this step requires

$$7T_{\text{Strassen}}(k) + 10k^2$$

where $T_{\text{Strassen}}(k)$ denotes the runtime of Strassen's for a matrix of size k.
We then compute the following sums

- $AE + BG = -P_2 + P_4 + P_5 + P_6$
- $AF + BH = P_1 + P_2$
- $CE + DG = P_3 + P_4$
- $CF + DH = P_1 - P_3 + P_5 + P_7$

which take time $8k^2$.

Therefore, we can write the recursion equation as

$$T_S(n) = T_S(2k) = 7T_S(k) + 18k^2$$

for an even n. For an odd $n = 2k - 1$, our algorithm runs by padding it with 0s to obtain a matrix of size $2k$ and runs Strassen without any further modifications, reducing the matrix size to $2k - 1$ afterwards. Therefore, for odd n,

$$T_S(n) = T_S(2k - 1) = T_S(2k) = 7T_S(k) + 18k^2$$

$$T_S(n) = \begin{cases} 7T_S\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2, & \text{even} \\ 7T_S\left(\frac{n+1}{2}\right) + 18\left(\frac{n+1}{2}\right)^2, & \text{even} \end{cases}$$

The python code performs a few speedups to avoid excessive matrix creation. The precise algorithm is as follows:

1. If the matrix dimension is less than the cutoff, multiply it with the conventional algorithm
2. Copy both matrices
3. If the matrix power is odd, pad both matrices with 0s on their last row and column
4. Split each matrix into four by indexing
5. Initialize two matrices of the reduced size to use as intermediaries and define $P_1, P_2, \ldots, P_7$ as our outputs for adding, subtracting, and multiplying the sub-matrices using np.add, np.subtract, and the recursive strassen algorithm.
6. Initialize our output matrix and directly add and subtract the P matrices into it using np.add and np.subtract.
7. If the original matrix was odd, de-pad the last row and column.

We optimized this in a few ways, most notably the use of intermediate matrices for adding and subtracting the split matrices A through H and routing the np.add and np.subtract output directly to the output array.

Cutoff

Observe that at the optimal value of n_0, computing the product of two $n_0 \times n_0$ using the conventional method will take time less than or equal to the time that would have taken if we ran Strassen's for one more iteration and then used the conventional algorithm, i.e. when

$$2n_0^3 - n_0^2 \leq 7\left[2\left\lceil\frac{n_0}{2}\right\rceil^3 - \left\lceil\frac{n_0}{2}\right\rceil^2\right] + 18\left\lceil\frac{n_0}{2}\right\rceil^2$$

For $n_0 = 2k, k \in \mathbb{Z}$

$$16k^3 - 4k^2 \leq 7\left(2k^3 - k^2\right) + 18k^2 \implies 2k^3 \leq 15k^2 \implies k \leq 7$$

For $n_0 = 2k - 1$

$$2(2k-1)^3 - (2k-1)^2 \leq 7\left(2k^3 - k^2\right) + 18k^2 \implies 16k^3 - 28k^2 + 16k - 3 \leq 14k^3 + 11k^2 \implies$$

$$\implies 2k^3 - 39k^2 + 16k - 3 \leq 0 \implies k \leq \frac{39}{2}$$

Therefore,

$$n_0 = \begin{cases} 14 & n \text{ even} \\ 37 & n \text{ odd} \end{cases}$$

i.e. if n is even, compare it against 14 and use the conventional algorithm if $n \leq 14$, and the same if n is odd and $n \leq 37$.

Task 2: Experimental Results

We were expecting the experimental value for the cutoff to be different from the experimental value for a few reasons, including:

- We assumed all arithmetic operations take constant time. However, a more accurate assumption would be that the arithmetic operations depend on the size of the numbers and the runtime for addition differ from that for multiplication.

  Observe that Strassen's requires more integer additions than the naive algorithm (additions of 18 $\lceil\frac{n}{2}\rceil \times \lceil\frac{n}{2}\rceil$ matrices at each combine step) and the naive algorithm requires more additions.

  Recall that we found addition to be $O(n)$ and stated that the fastest known asymptotic bound for multiplication is $O(n \log n)$. Although, in practice, this will not be the exact runtime, we do expect multiplication to take longer than addition in numbers of the

same size. Therefore, for matrices with large numbers, we would expect the crossover point to be lower due to multiplication for larger numbers taking longer than addition. Strassen's being preferable in these cases.

However, for matrices of entries 0 and 1, the naive algorithm will only multiply 0s and 1s together, which will be constant time per each multiplication. For addition, however, we will have to consider numbers with length greater than 1 and such additions appear more in Strassen's. Therefore, we would expect the cutoff to be higher in practice.

- Additionally, the analytical solution does not consider the time required to manage the memory used for the matrices required for calculations. Since Strassen's requires calculating multiples matrices of size $\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil$, this might lead to the runtime of Strassen's being larger than expected, and lead to an increase in the cutoff.

The crossovers do not differ when we run on a matrix with uniformly distributed choices of 0 and 1 or of 0,1, and 2. We get an experimental crossover of (suprisingly) around 14 for even cases and 37 for odd cases.

This might be because the sizes of the matrices are small enough that the differences in the memory requires and the sizes of the numbers involved does not make a significant difference for the runtime.

With regards to how relevant the idea of a crossover point is, although there seems to be a range of values for which the two runtimes are about the same, since the dimension goes down by half at each iteration of Strassen's, the difference in switching to the naive algorithm versus having one more iteration is non-negligible. Preventing Strassen's from running until it reaches the base case does allow for a significant speedup.

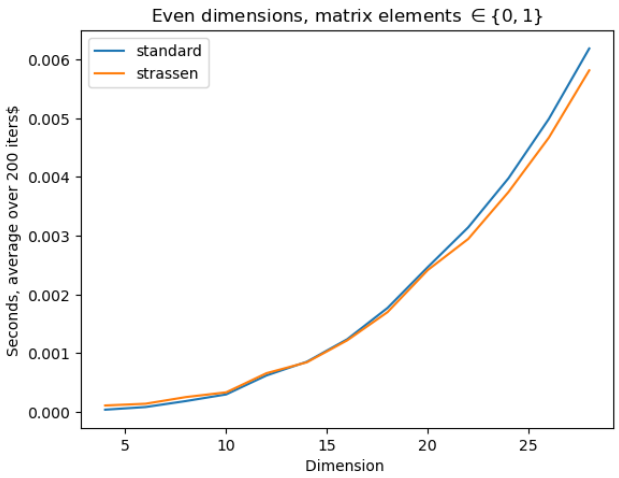

Figure 1: Crossover for even values for matrix with values in $\{0, 1\}$

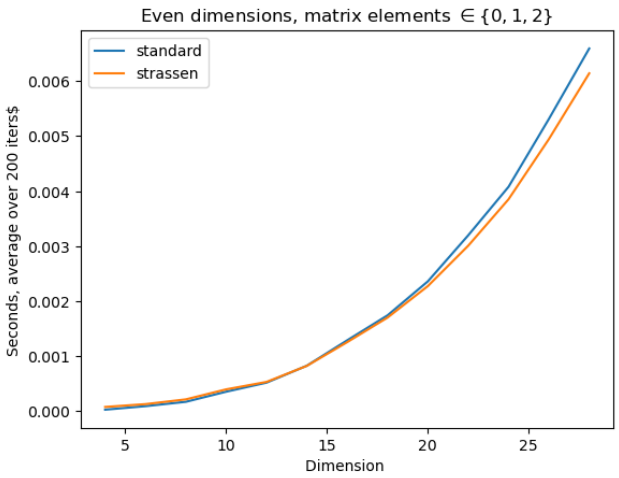

Figure 2: Crossover for even values for matrix with values in $\{0, 1, 2\}$

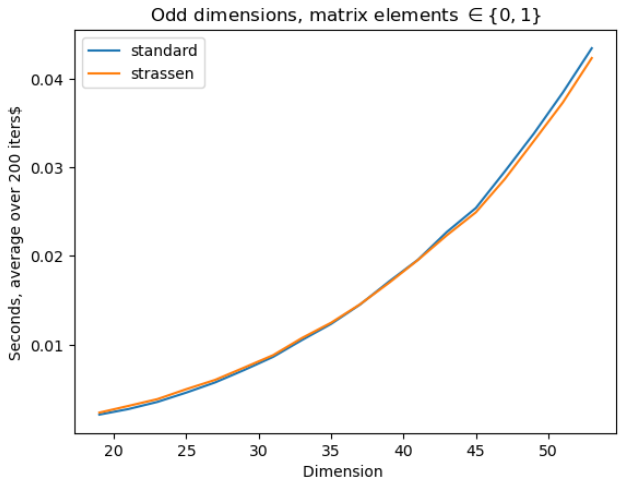

Figure 3: Crossover for odd values for matrix with values in $\{0, 1\}$

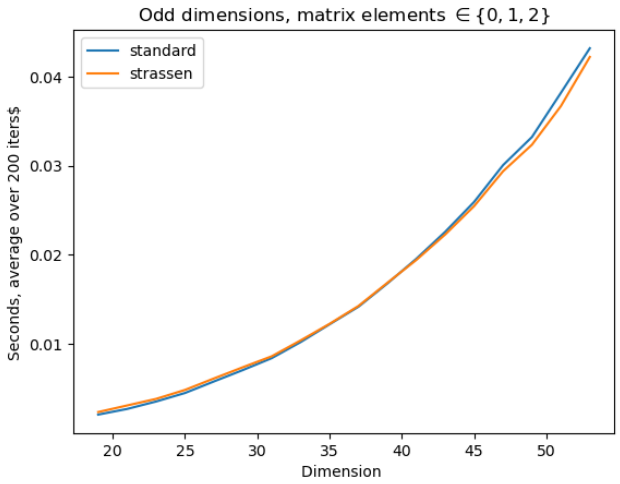

Figure 4: Crossover for odd values for matrix with values in $\{0, 1, 2\}$

Task 3: Triangle in Random Graphs

Recall that the adjacency matrix representation of a graph on n vertices (denote M) keeps track of the existence of an edge by setting the entry i j equal to 1 if and only if there is an edge from i to j. For undirected graphs, $(i, j) \in E \iff (j, i) \in E$.

Therefore, we start by generating a graph by initializing a zero matrix and setting entries in the upper triangle of the matrix (excluding the diagonal) to 1 by the given probability. Then, by the definition of matrix multiplication, for a matrix A^m, the i j entry will correspond to the number of paths from vertex i to j.

If there exists a triangle (i.e. a cycle) consisting of vertices $a \to b \to c \to a$, then there will be at least two path from a to a and b to b etc., one for each direction. Therefore, the trace fo M^3 divided by 6 gives us the number of triangles in a matrix. The expected value for the number of triangles is given by $\binom{1024}{3}p^3$. As shown below, this matches our experimental results:

p	$\binom{1042}{3}p^3$	Experimental
0.01	178.433	185
0.02	1427.464	1429
0.03	4817.692	4956
0.04	11419.713	11261
0.05	22304.128	22056

Table 1: Expected vs. experimental numbers of triangles

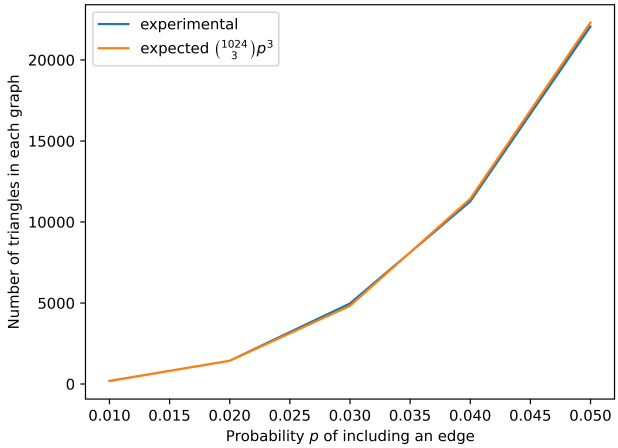

Figure 5: Graph of number of triangles for each p, expected vs. experimental.