Programming Assignment

Your name(s): Di Zhen

Overview:

Strassen's divide and conquer matrix multiplication algorithm for n by n matrices is asymptotically faster than the conventional $O(n^3)$ algorithm. This means that for sufficiently large values of n, Strassen's algorithm will run faster than the conventional algorithm. For small values of n, however, the conventional algorithm may be faster. Indeed, the textbook Algorithms in C (1990 edition) suggests that n would have to be in the thousands before offering an improvement to standard multiplication, and "Thus the algorithm is a thoeretical, not practical, contribution." Here we test this armchair analysis.

Here is a key point, though (for any recursive algorithm!). Since Strassen's algorithm is a recursive algorithm, at some point in the recursion, once the matrices are small enough, we may want to switch from recursively calling Strassen's algorithm and just do a conventional matrix multiplication. That is, the proper way to do Strassen's algorithm is to not recurse all the way down to a "base case" of a 1 by 1 matrix, but to switch earlier and use conventional matrix multiplication. That is, there's no reason to do a "base case" of a 1 by 1 matrix; it might be faster to use a larger-sized base case, as conventional matrix multiplication might be faster up to some reasonable size. Let us define the *cross-over point* between the two algorithms to be the value of n for which we want to stop using Strassen's algorithm and switch to conventional matrix multiplication. The goal of this assignment is to implement the conventional algorithm and Strassen's algorithm and to determine their cross-over point, both analytically and experimentally. One important factor our simple analysis will not take into account is memory management, which may significantly affect the speed of your implementation.

Tasks:

1. Assume that the cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) is 1, and that all other operations are free. Consider the following variant of Strassen's algorithm: to multiply two n by n matrices, start using Strassen's algorithm, but stop the recursion at some size n_0, and use the conventional algorithm below that point. You have to find a suitable value for n_0 – the cross-over point. Analytically determine the value of n_0 that optimizes the running time of this algorithm in this model. (That is, solve the appropriate equations, somehow, numerically.) This gives a crude estimate for the cross-over point between Strassen's algorithm and the standard matrix multiplication algorithm.

   Answer:

   Let $S(n), T(n), H(n)$ be the runtime of Strassen's algorithm, conventional matrix multiplication, and hybrid Strassen's algorithm. $S(n) = n^2(2n-1) = 2n^3 - n^2$ because multiplying two $n \times n$ matrices takes n multiplications and $n-1$ additions for each resulting number in a resulting $n \times n$ matrix, and there are n^2 numbers on the resulting matrix. $T(n) = 7 \cdot T(\lceil n/2 \rceil) + 18 \cdot (\lceil n/2 \rceil)^2$. Because Strassen's algorithm multiplies two $\lceil n/2 \rceil \times \lceil n/2 \rceil$ matrices 7 times and adds or subtracts them 18 times. The runtime of hybrid Strassen's is as follows:

$$H(n) = \begin{cases} 7 \cdot H(\lceil n/2 \rceil) + 18 \cdot (\lceil n/2 \rceil)^2 & n > n_0 \\ 2n^3 - n^2 & n \leq n_0 \end{cases}$$

   We want to choose n_0 such that the runtime is minimized, that is

$$H(n) = 7 \cdot \min(H(\lceil n/2 \rceil), S(\lceil n/2 \rceil)) + 18 \cdot (\lceil n/2 \rceil)^2$$

Since when $n \leq n_0, H(n) = S(n)$, we want to find the smallest n_0 such that $H(n) < S(n)$, that is

$$7 \cdot (2(\lceil n/2 \rceil)^3 - (\lceil n/2 \rceil)^2) + 18 \cdot (\lceil n/2 \rceil)^2 \leq 2n^3 - n^2$$

If n is odd, let $n = 2k + 1$, k as a non negative integer, we solve $7(2k + 1) \cdot (k + 1)^2 + 18(k + 1)^2 \leq (2k + 1)^2 \cdot (4k + 1)$, and get $k \geq 19, n \geq 39$. If n is even, let $n = 2k$, k as a positive integer, we solve $7(2k^3 - k^2) + 18k^2 \leq 16k^3 - 4k^2$, and get $k \geq 8, n \geq 16$. Therefore, the cross-over point for odd n is 39 and for even n is 16.

2. Implement your variant of Strassen's algorithm and the standard matrix multiplication algorithm to find the cross-over point experimentally. Experimentally optimize for n_0 and compare the experimental results with your estimate from above. Make both implementations as efficient as possible. The actual cross-over point, which you would like to make as small as possible, will depend on how efficiently you implement Strassen's algorithm. Your implementation should work for any size matrices, not just those whose dimensions are a power of 2.

   To test your algorithm, you might try matrices where each entry is randomly selected to be 0 or 1; similarly, you might try matrices where each entry is randomly selected to be 0, 1 or 2, or instead 0, 1, or -1. We will test on integer matrices, possibly of this form. (You may assume integer inputs.) You need not try all of these, but do test your algorithm adequately.

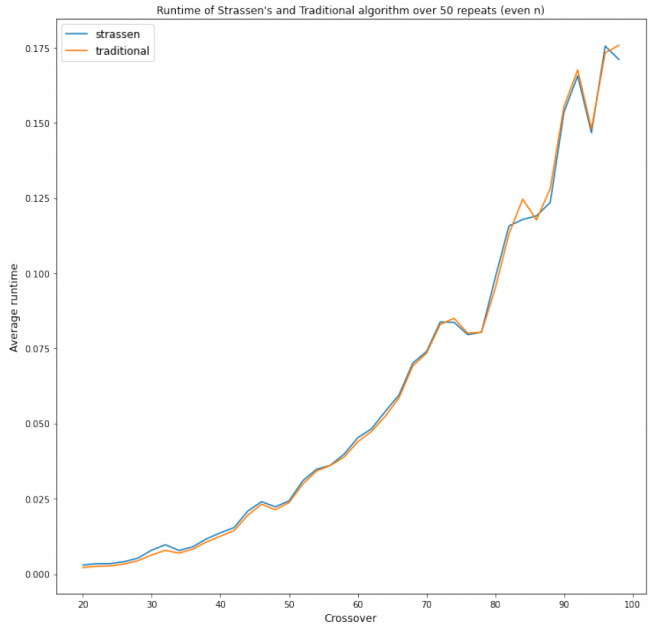
   Answer:

   Before hybrid Strassen's algorithm is performed, we take care of matrix with odd dimension. When n is odd, we pad an addition row at the bottom of the matrix and an addition column on the right of the matrix. When n is even (originally or after padding), we perform hybrid Strassen's algorithm as follows: the algorithm is recursively called until it reaches the cross-over point, after which the traditional matrix multiplication is performed. The algorithms are implemented in Python.

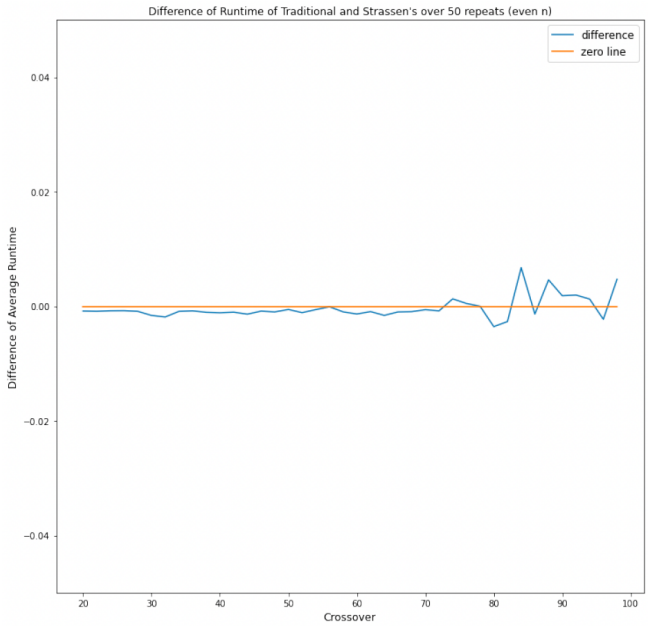
   The matrices are initialized to contain only 0, 1, 2 integers randomly. The algorithm is tested for even n and odd n separately, and reports average runtime over at least 50 repeats. The range of n is $(20, 100)$ at the beginning, after observing clear trend, the range is chosen to be $(20, 100)$. By analyzing the plots of runtime, we finds corresponding cross-over points. Two plots compare the runtime of hybrid Strassen's and traditional algorithm on crossover value of 20 to 100. Additional two plots show the runtime difference between the two algorithms (Figure 1). The cross-over points are identified by finding a point on the plot such that the Strassen's gets better than traditional.

   It can be observed that the runtime of both algorithm are similar at the first several cross-over points, but when n goes large, traditional algorithm takes more time than hybrid strassen's. Also, when size is odd, at the beginning, the runtime for hybrid strassen is higher than traditional multiplication. This can be explained by the fact that padding takes additional time. Experimentally, the cross-over point is found to be about 80 for even size n and about 44 for odd size n. The cross-over points are larger than the analytic results. This is expected because there are some additional time taken by implementation such as memory allocation.

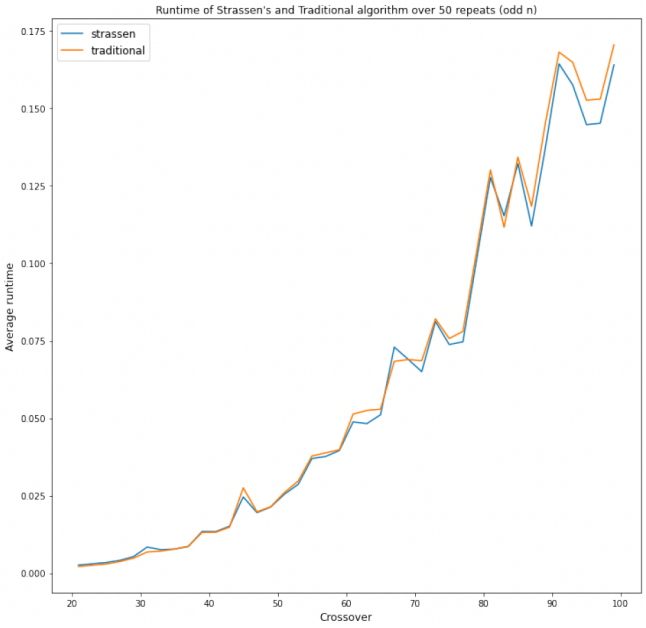
3. Triangle in random graphs: Recall that you can represent the adjacency matrix of a graph by a matrix A. Consider an undirected graph. It turns out that A^3 can be used to determine the number



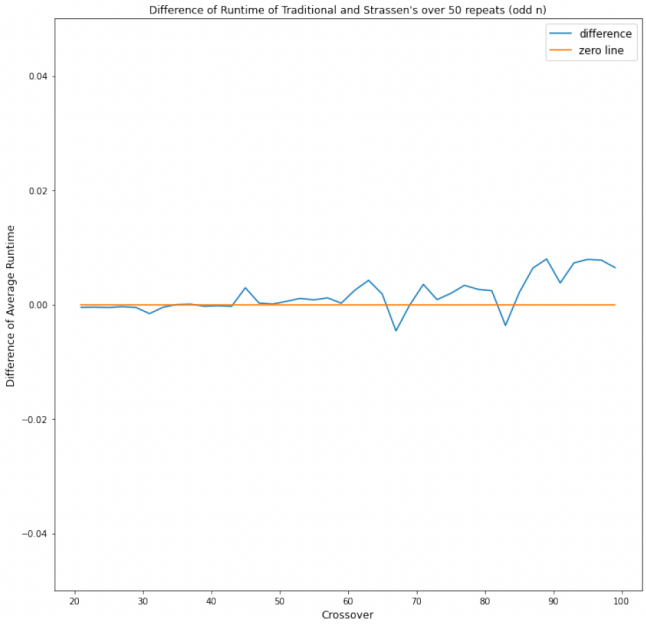
(a) even size



(b) even size



(c) odd size



(d) odd size

Figure 1: Plots comparing the runtime of traditional algorithm and the Strassen's over different cross-over points.

of triangles in a graph: the (ij)th entry in the matrix A^2 counts the paths from i to j of lenth two, and the (ij)th entry in the matrix A^3 counts the path from i to j of length 3. To count the number of triangles in in graph, we can simply add the entries in the diagonal, and divide by 6. This is because the jth diagonal entry counts the number of paths of length 3 from j to j. Each such path is a triangle, and each triangle is counted 6 times (for each of the vertices in the triangle, it is counted once in each direction).

Create a random graph on 1024 vertices where each edge is included with probability p for each of the following values of p: $p = 0.01, 0.02, 0.03, 0.04$, and 0.05. Use your (Strassen's) matrix multiplication code to count the number of triangles in each of these graphs, and compare it to the expected number of triangles, which is $\binom{1024}{3}p^3$. Create a chart showing your results compared to the epectation.

Answer:

A graph with 1024 vertices and randomly generated edges with probability distribution following Bernoulli(p), $p = \{0.01, 0.02, 0.03, 0.04, 0.05\}$, was generated. To find the triangles, the adjacency matrix A is cubed first to be A^3 then the diagonal entries are summed up and divided by 6. The average triangle numbers are counted over 50 repeats. Since size is even, the cross-over point is set to be 80 for hybrid Strassen's algorithm. The simulated number of triangles and the expected number of triangles are compared over all p. Table 1 below shows that the simulated numbers are consistent with the expected numbers. The Strassen's counts and the expected counts of triangles are very close. If the number of repeats is large enough, the strassen's result will be closer to the expected count.

p	**Strassen's Count**	**Expected Count**
0.01	178	178.433
0.02	1426	1427.464
0.03	4823	4817.692
0.04	11421	11419.714
0.05	22319	22304.128

Table 1: Comparison of Strassen's count of triangles and the expected number of triangles.