Computer Systems 2(CEN 502)

Project 1 Report

Group – 22

Group Members

Mayank Gupta (1209469518)

Aman Karnik (1209536091)

Yogesh Kubal (1209536364)

Summary :

We have been provided a big sea ice concentration anomaly data set and we need to analyse the correlation of ice thickness at one place with the other. The data is weekly and stored in multiple data files for each week, where each value represents the percentage deviation of ice thickness of a cell(predefined area size on the map) from the average computed over the years for which data is captured. We have compared the results with the Small-world random graph and found the value for our data and a small world random graph of similar size to be comparable. It is also noticed from the histogram plotted over the degree of each node that while majority of the nodes have degree close to 0 , yet there are some nodes for which it even exceeds 100. Which in a way strengthens our notion of a small world graph having small clusters separated by multiple nodes.

Introduction:

Before moving on to the big data set we have been given a small data set covering a span of 16 years wherein each year contains a total of 52 weeks thereby accounting for a total of (16x52) 832 arrays. Any given week in the data set contains an array of 63x63 elements of 32 bit floating point binary numbers. The two dimensional small dataset of 63x63 elements for (16x52) 832 weeks can be visualized as a three dimensional array. Although we just convert this 2d array into single 1d array of length 63*63 = 3969 . This just makes life easier in terms of making the code less complex and error prone and also helps in easier visualisation of data. The programming language used for the implementation of the code is Python 2.7 and for plotting the histograms Matlab is used. Most of our efforts went into optimising the code and making it as fast as possible. We learned some stuff on the way and it ll be discussed in the coming sections.

The code is essentially in 5 parts :

1) Reading the data :

We initially tried using 'sed'(serial editor) and hexdump to convert the data into hex values by taking a hexdump to a file , formatting it properly & then read those hex values . We played with this for some time trying to read all the data first and building a single data file. Although we were able to dump all the data from all the weeks and years into a single file. Reading back from the hexdump was again giving errors . We also spent some time bending our brains around how endianness works and eventually did figured it out. But it was slow and we also couldn't justify an extra read&write operation which couldn't be justified. So we decided to use python for read & write.

The function iterates over all the files in the data folder and reads the data in 4 bytes that is 32 bits, unpacks it and stores it in an array. Storing in the array was done to increase speed as memory read is faster than file read write although in our case it wouldn't have made much of a difference anyways. I believe it was again done to make life easier.

We directly use the values provided and do not subtract them from the mean as the mean of the percentage deviation would be 0. It can be proved mathematically and we have attached a figure at the end explaining the maths behind it.

2) Computing the correlation of each node with every other node

This was probably the most expensive part computationally and took some time to optimise. python has a built in multiprocessing library and we forked 7 processes to parallelise the task of correlation computation.

Lesson 1 - Increasing the no. of process beyond 5 didn't really help . According to our understanding it was mainly because of the limited no. of cores (4 in our case) and increasing no. of processes wouldn't have gotten us any more of CPU time.

Lesson 2 - Using multithreading instead of multiprocesses doesn't help to exploit concurrency because it still gets one cycle at the CPU which is for the parent process.

Lesson 3 - Using any of the 6 virtual consoles in Ubuntu instead of running the one with X server(primary display) decreases the computation time further by atleast a factor of 2. The reason could be absence of a running X server and other applications which are running by default thereby giving more CPU space to the program.

3) Finding the degree for each node using threshold correlation coefficient value

We use an adjacency array which is not really a standard 2d array as each row is of different length. Some novelty that python enables. After calculating the degree for each node we calculate the mean to later use for comparison of our graph with a random graph.

4) Calculating clustering coefficient

The algorithm is to basically take intersection of each row in adjacency array with the other row. Again here we convert the row representing a node 'i' into a set and just take XOR of all the pairs (x,y) such that x & y both are in the neighbourhood of 'i' or in the row representing it. The XOR is again a fast operation and the division of sum by 2 is just right shift by 1 , which is again much faster. The significant amount that is saved is in the XOR. Eventually the mean clustering coefficient is calculated

5) Calculating the characteristic path length

This one again took some thinking. We couldn't decide on using Bellman ford or Djikstar's for this. Then we realised the edges were all of same weight 1 !! Is it possible to use a faster algorithm . A slight modification (actually not really a modification) in BFS helps us to compute the characteristic path length in time O(V(V+E)). We could see from the histogram that this is almost like a sparse matrix and had a feeling that no. of edges is of order (V) probably . Effectively what this meant was worst case running time of O(V*E) and it didn't bother us at all actually.

Our algorithm when run on a 2nd Gen core i5 2.5 ghz machine using the virtual console in Ubuntu crunches the small data in approx 3 minutes. When run on a latest core i7 machine the running time was never over 1min 30 secs for the small data set. On top of everything we are

using pypy instead of python to fire our code. pypy is much faster and more optimised than python and has almost all of the libraries of python. One can use it to run any python code

Body:

While working with the small data set the intermediate elements of a 2d array visualized as 3 dimensional will need to be stored and worked upon so as to determine the data set parameters needed to be calculated.

Tasks completed are:

Task 1:

- A correlation based graph for two threshold values r corresponding to 0.9 and 0.95. A degree distribution histogram is calculated and plotted.
- Clustering coefficient and the characteristic path length of the graph are computed and stored in the Results file.
- The clustering coefficient for the random graph G_random and the Characteristic path length are caculated as k/n & log(n)/log(k) respectively. n - 3969(no. of nodes) , k - 24 (mean vertex degree)
- The value of both these quantities of the graph generated from given data is more than that of G_random.

Task 2:

- We split the small data set into 2 parts of eight years each. The

Task 3:

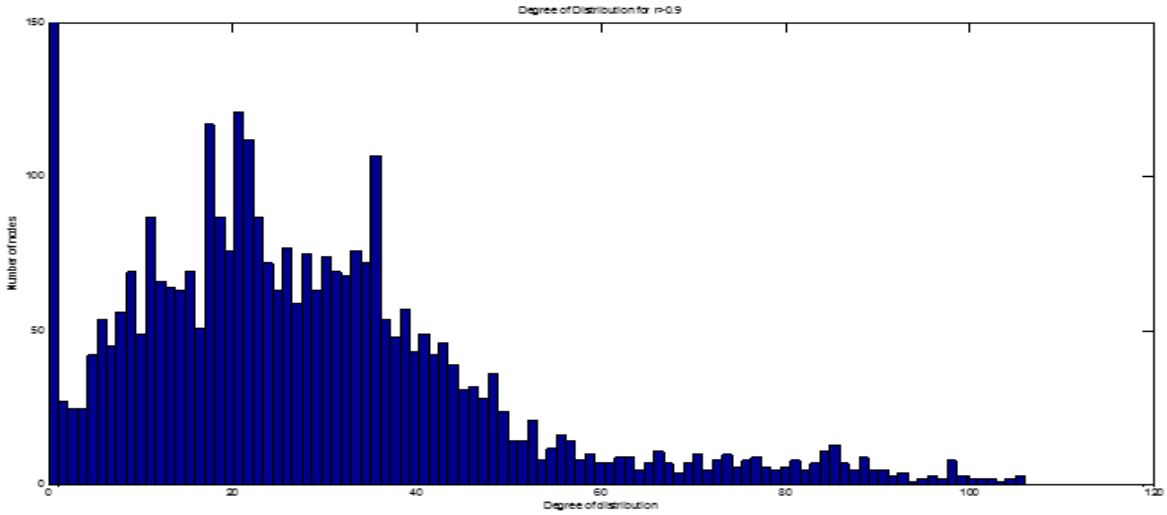
- Finally, repeat Task 1, except that when computing the correlation-based graph, consider a time lag of s = {1, 2, 3, 4} weeks.

Task 4:

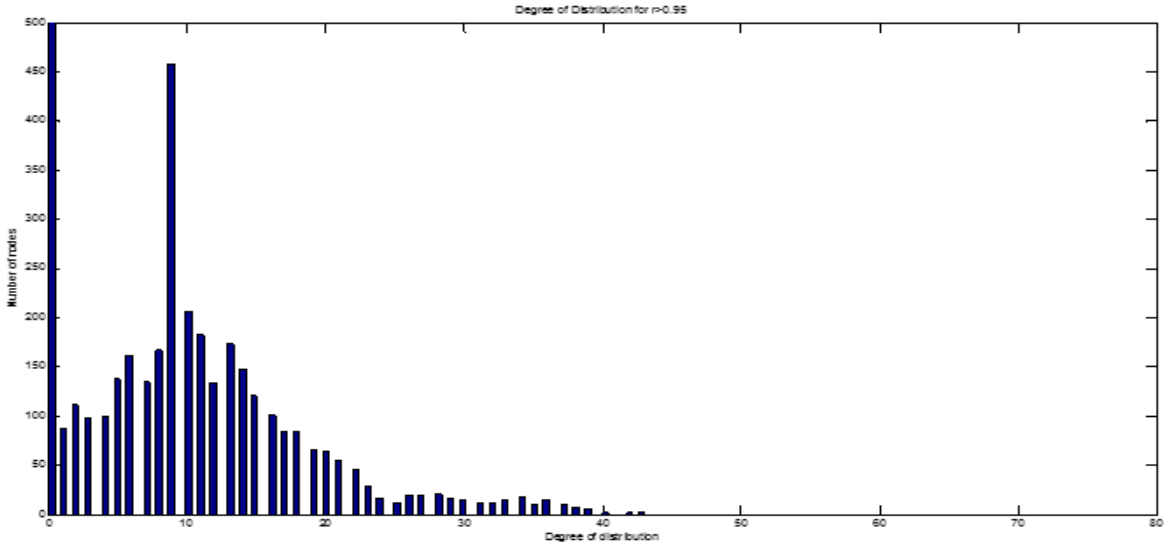
- Write a report that summarizes your findings for the 27-year period, and the three 9-year periods. Compare and contrast the differences in the graph statistics between the 27-year period, the 9-year periods, and the lagged correlation. Add a section to the report that discusses your representation of the graph, and any optimizations you made in computations. Depending on the order in which you calculate the statistics, you can likely save and make use of previous results to cut down on the computation time.

Histograms obtained from matlab :

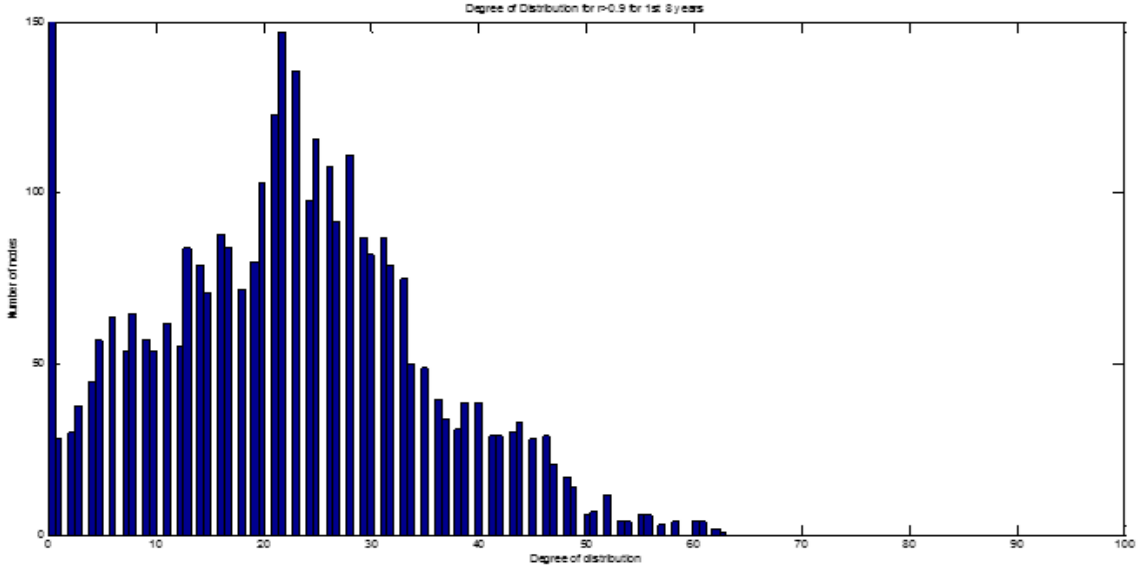
1. For degree distribution r > 0.9



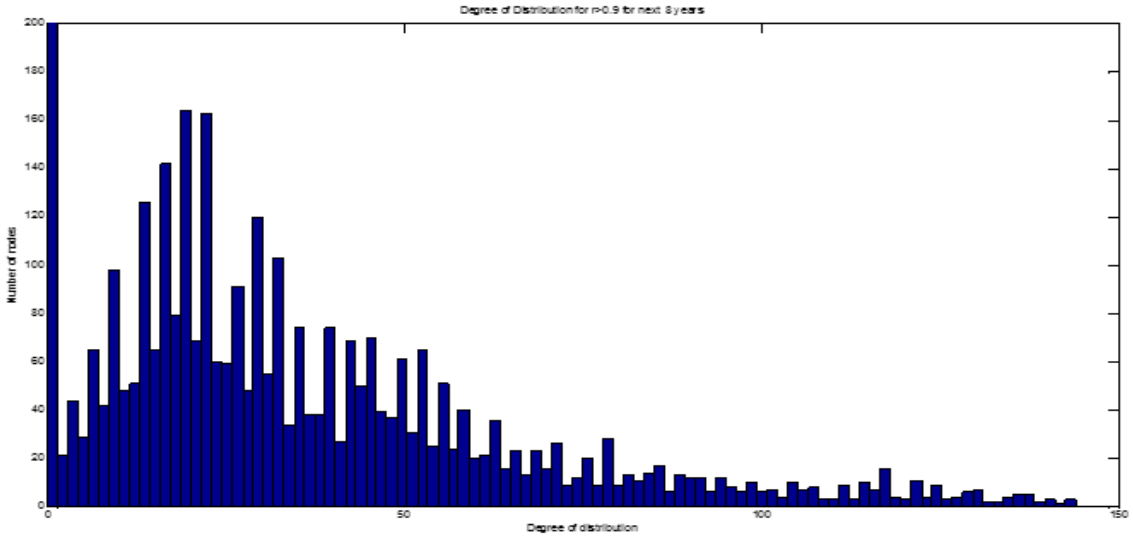
2 . For degree distribution r > 0.95



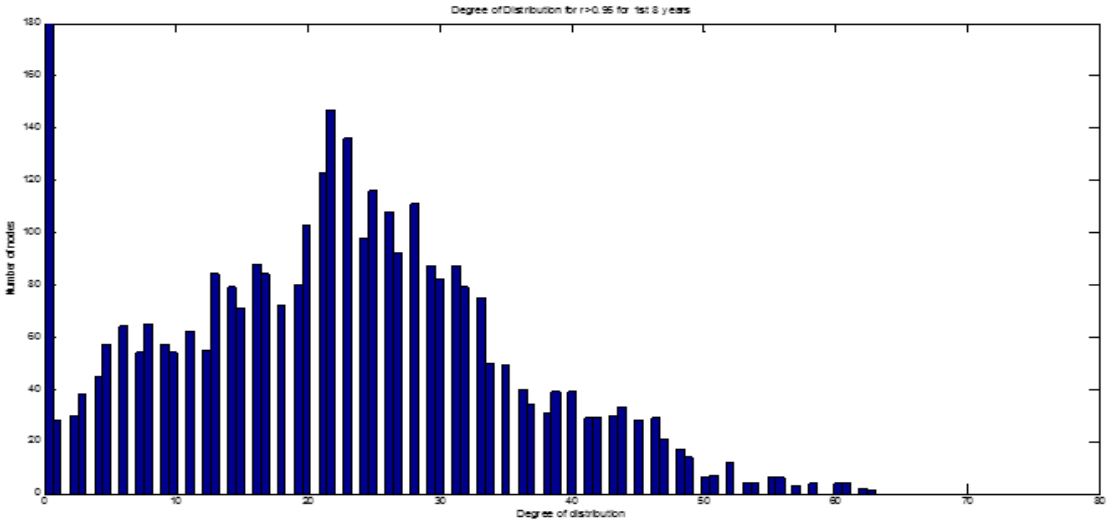
3. For degree distribution r > 0.9 for the first 8 years



4. For degree distribution r > 0.9 for the next 8 years



5. For degree distribution r > 0.95 for the first 8 years



6. For degree distribution r > 0.95 for the next 8 years