COL380 ASSIGNMENT 2
ANKIT KUMAR(2020CS10323)
PRAVIN KUMAR(2020CS10369)

1.Method used:

Naive Approach 1: Firstly just implemented the k size groups using basic mintruss algorithm without using any filteration so my iterations were so large and therefore almost taking the time of about 3 min on graph 7 and all the load is dsitributed on same processor/

Naive Approach 2:
Now to improve or reduce the iterations we prefilter the graph using the three given algorithms in the assignment pdf , so my time reduces a 10 times from previous approach as the number of vertex and edges are already removed in this prefiltering so time is very less but still this algo distribute the load on a single processor only so not a best algo in terms of distribution.

Below are the time tables based on different test cases running on the naive approach 2 ::

Test 2	**Test 3**	**Test 5**	**Test 6**	**Test 8**
3.522	495.32	65.123	192.293	8.322

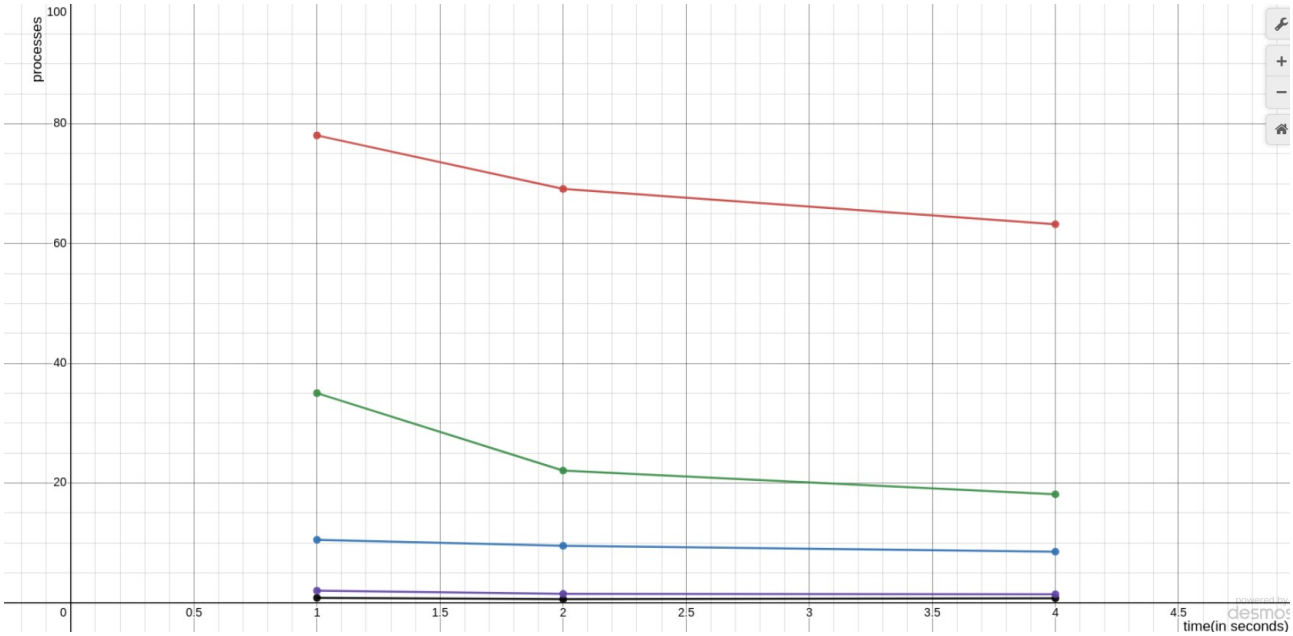
Here time is in seconds.

Distributed approach using open_mpi(final algorithm)::

Now to furthur improve above approach 2 we used mpi interface in which for each support calculation in approach 2 we distributed the calculations among the ranks so that our edge calculations is fast and efficient and also we used maps and sets to reduce to store the edges and supports so that calculations will be fast . This furthur reduces the time to calculate as for graph 3 to 60 seconds.For distributing the support calculations used MPI_recv and MPI_send functions.

Runtime analasyis on the above distributed algorithm::

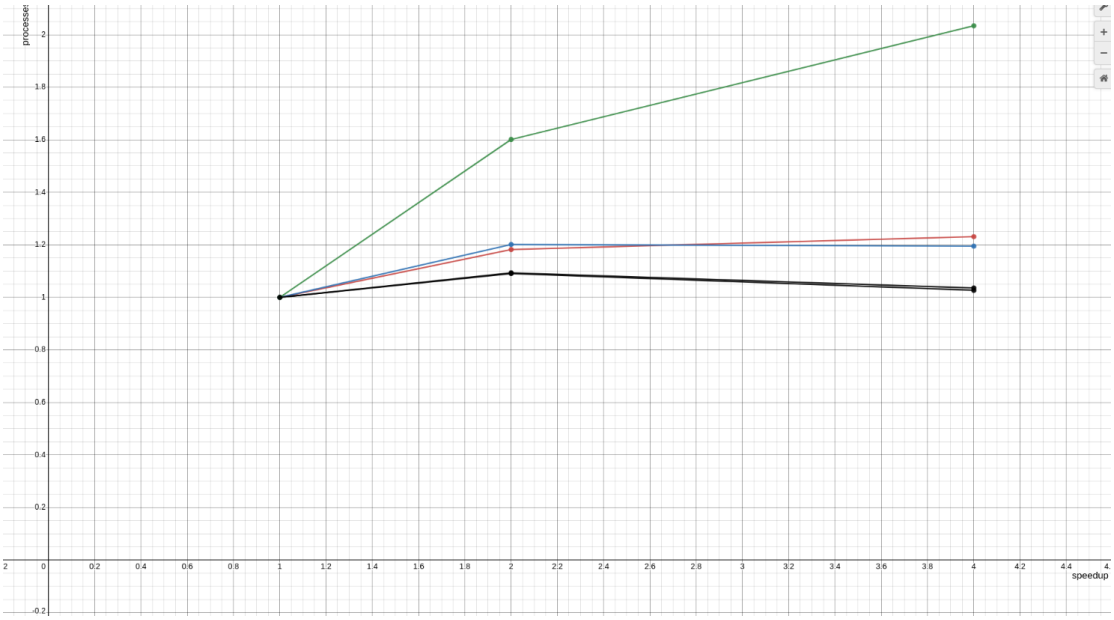
number_of_process	**Test 2**	**Test 3**	**Test 5**	**Test 6**	**Test 8**
1	0.8331	78.0541	10.5323	35.0121	2.0123
2	0.6021	69.1134	9.5124	22.0826	1.5123
4	0.7711	63.2333	8.5487	18.1178	1.4312



on X axis is time and on y axis the number of process or cores and each color represents different different test cases.

2. Speedup analasyis on the above algorithm::

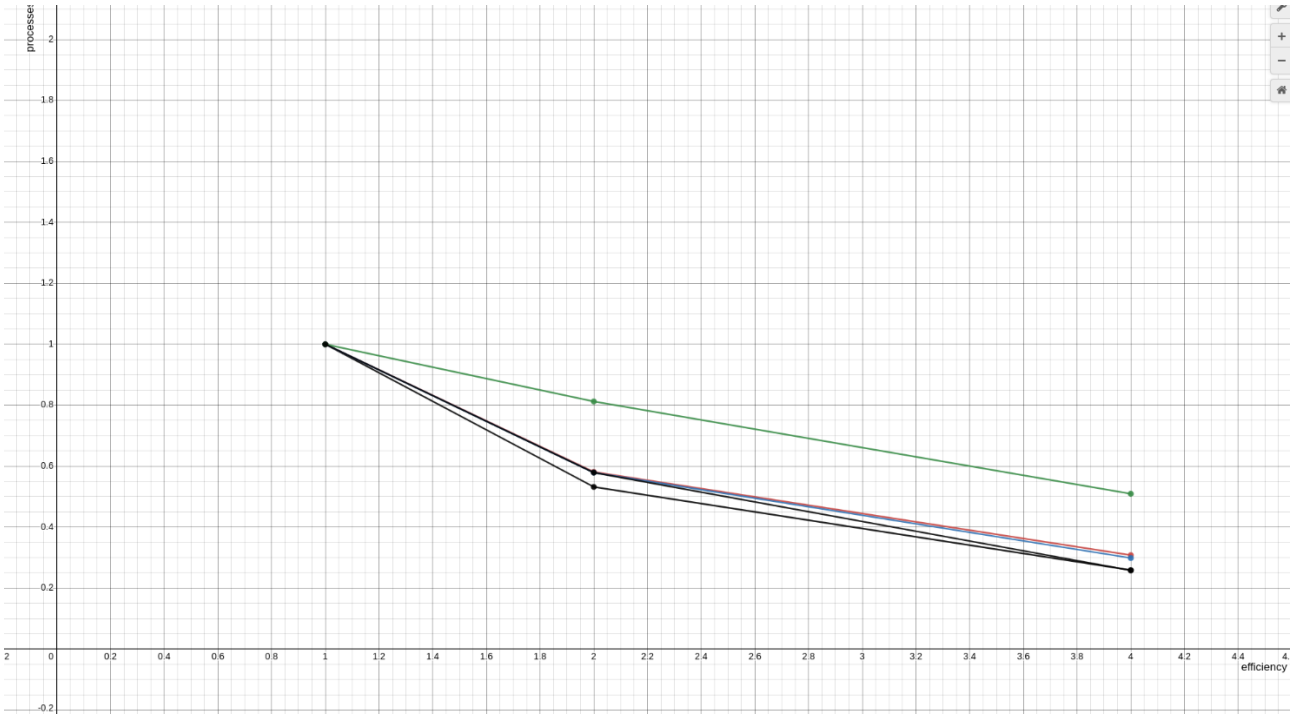
number_of_process	**Test 2**	**Test 3**	**Test 5**	**Test 6**	**Test 8**
1	1	1	1	1	1
2	1.0932	1.1823	1.2015	1.6012	1.09083
4	1.0359	1.2312	1.19495	2.0341	1.0269



on X axis is speedup and on y axis the number of process or cores and each color represents different different test cases.

3.Efficiency analasyis on above algorithm::

number_of_process	Test 2	Test 3	Test 5	Test 6	Test 8
1	1	1	1	1	1
2	0.5319	0.5812	0.5789	0.8123	0.5788
4	0.2590	0.3079	0.2987	0.5093	0.2581



on X axis is speedup and on y axis the number of process or cores and each color represents different different test cases.

2.Observations:

As from the above graphs it is clearly visible that our program is quite scalable as the for each process runtime is decreasing and the speedup is increasing.

THANK YOU