##

######### COL 761 - ASSIGNMENT 2 ########

##

QUESTION 1:

########################## OBSERVATION ##########################

1. We have noticed that the time decreases as the support increases.

2. The time decrease when we decrease the support is high (exponential) for gspan and fsg compared to Gaston (approximately linear)

3. Gspan is faster compared to fsg in most of the cases.

4. Gaston is very fast for lower support compared to gspan and fsg.

############################## ANALYSIS ##############################

After analysing the implementations of FSG, GSPAN and GASTON algorithms we observed few points on their performances. As we run the FSG, GSPAN and GASTON implementations on different support thresholds, we observed that:

- We have analysed that at a lower threshold there are higher number of frequent subgraphs (i.e. higher number of fragments). This increases in an exponential manner.
- As we decrease the support threshold, GASTON takes less amount of runtime as compared to FSG and GSPAN. GSPAN takes more time than GASTON but less time than FSG algorithm for the frequent subgraph mining. FSG takes more time than GSPAN and GASTON algorithms to mine frequent subgraphs. We can see this behaviour int the below plot.
- For the higher support threshold, the time taken by FSG, GSPAN and GASTON are comparatively similar as shown in the plot time taken by all the three algorithms is close to each other.

FSG implements a breadth first strategy that creates candidate subgraphs of size k+1 using k size subgraphs and then prunes the infrequent ones. Whereas, in gspan candidates generation does not takes place due to creation of Depth First Search (DFS) Code tree and follow depth first based pattern generation similar to fp growth and thus do not require pruning.

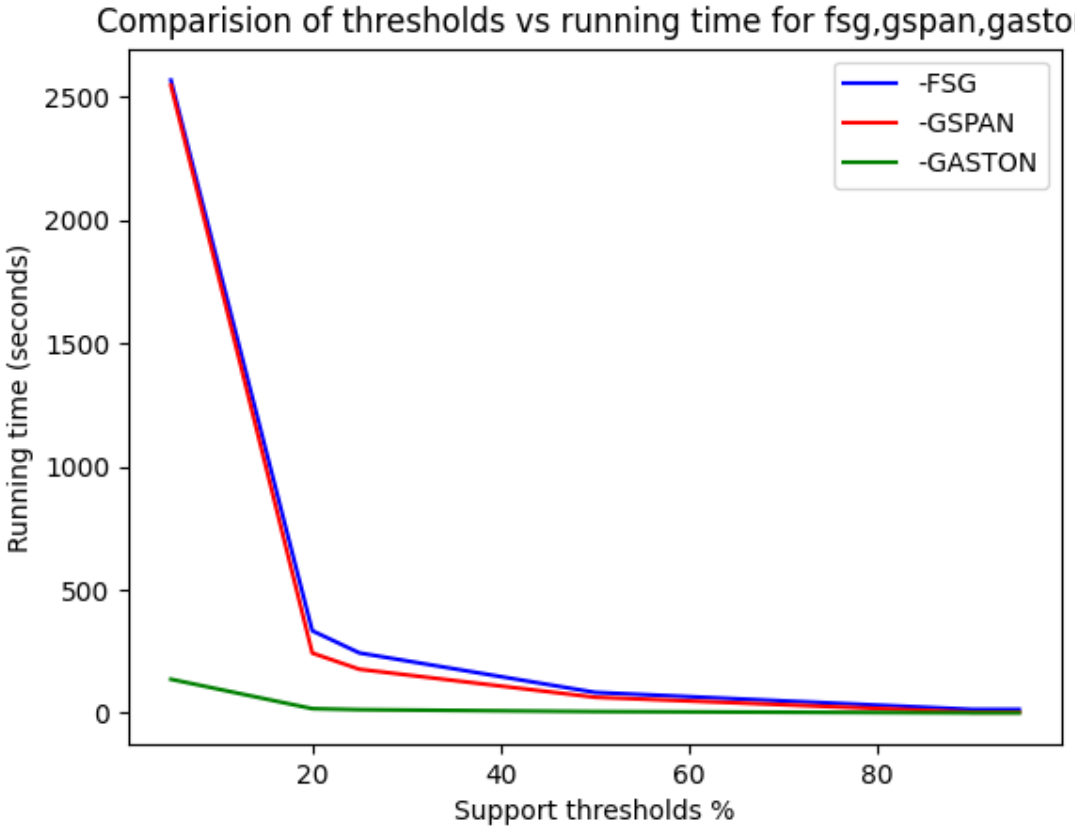
Gaston implementation performs the best as it proceeds mining paths ->mining subtrees -> mining subgraphs with cycles.

The expensive subgraphs isomorphism testing occurs only in subgraph mining phase. The idea behind approach is that for graphs, polynomial time algorithms are not available but for paths and trees, efficient isomorphism algorithms exist.

For graphs, Gaston algorithm creates cycles that are greater than previously generated graphs cycles.

We computed time on 5,10,25,50 and 95 threshold for fsg, Gaston and gspan. As we can see in above graph that all thresholds are giving output in under 1 hour.

- FSG follows the apriori approach, It has overhead of candidate generation phase. So, FSG takes much time than all non-apriori implementations.
- Gspan follows non-apriori pattern growth approach. It is based on dfs approach. It uses dfs codes which are canonical representation of graphs. In the growth phase, fragments are added only from the right most path in the DFS tree. Gspan computes canonical lexicographical dfs code for each refinement because the applied pruning rules can not properly prevent isomorphic fragment generation. If the canonical lexicographical code is not minimal then it can be pruned.
- Gaston is also a non-apriori pattern growth approach. For fast isomorphism testing, in order to generate refinements Gaston keeps all the embeddings. There are efficient ways to enumerate paths or noncyclic trees. Gaston detects the duplicates by hashing and graph isomorphism test.



QUESTION 2:

What should be the frequency threshold?

We fixed the threshold as follows:

if Number of graphs greater than 50000

then

support threshold = "0.60"

elif Number of graphs greater than 40000

then

support threshold = "0.55"

elif Number of graphs greater than 30000

then

support threshold = "0.50"

elif Number of graphs greater than 20000

then

support threshold = "0.40"

elif Number of graphs greater than 10000

then

support threshold = "0.30"

else

support threshold = "0.20"

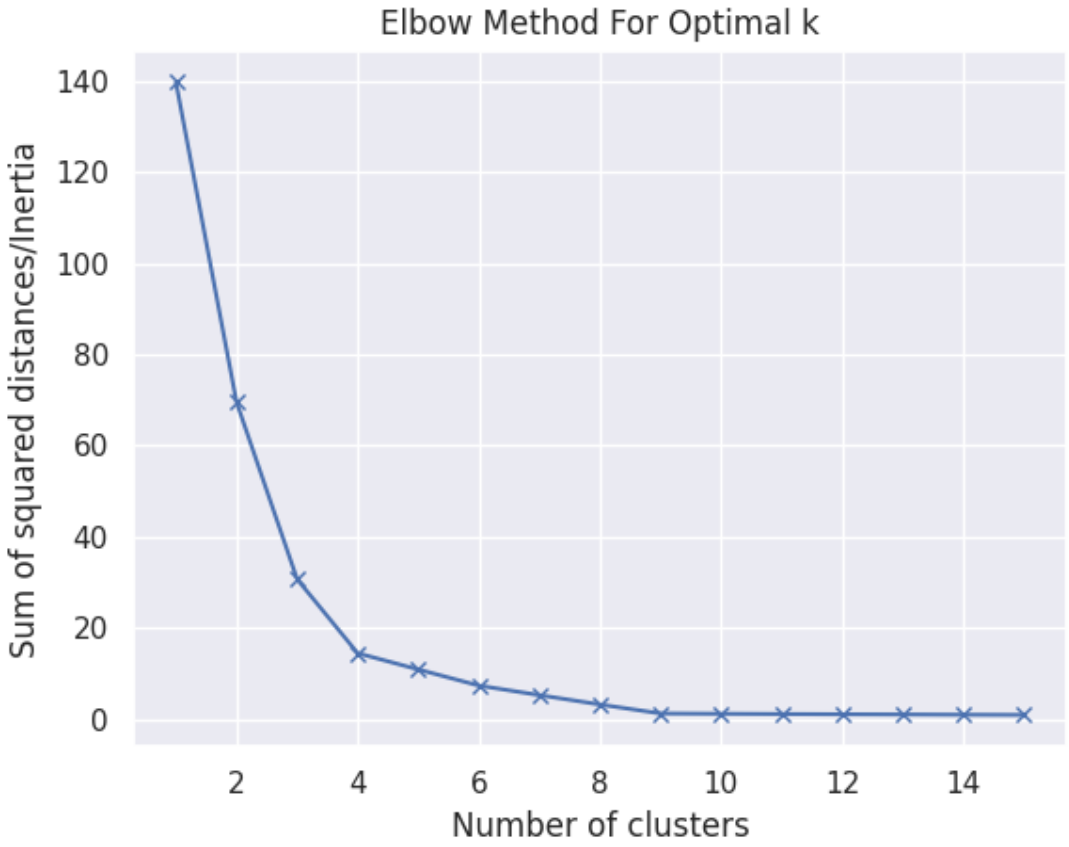
For the yeast dataset we have seen gSpan was fast and uses DFS approach that is FP-tree based. We tried several other threshold values but the total query increasing time was increasing as I was increasing the threshold values. So, I decided to fix the threshold as per graph numbers so that we can work keeping the indexing time limit given to us.

What should be the value of m? The number of frequent subgraphs is likely to be very high. How do you prune it down to m?

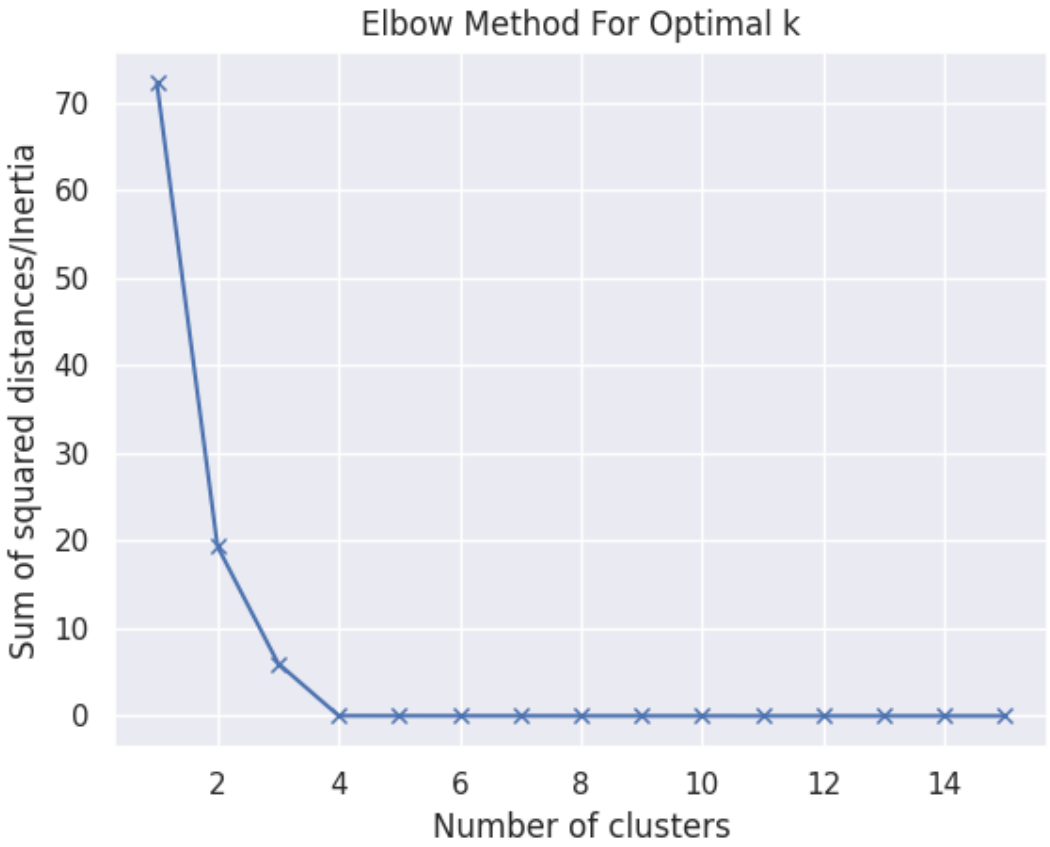
The value of m that we decided is 500 + |{ G where is a single edge $e \mid e \epsilon S\}|$ G e graph.

Keeping this was very beneficial for us in terms of saving time and expensive subgraph isomorphism tests.

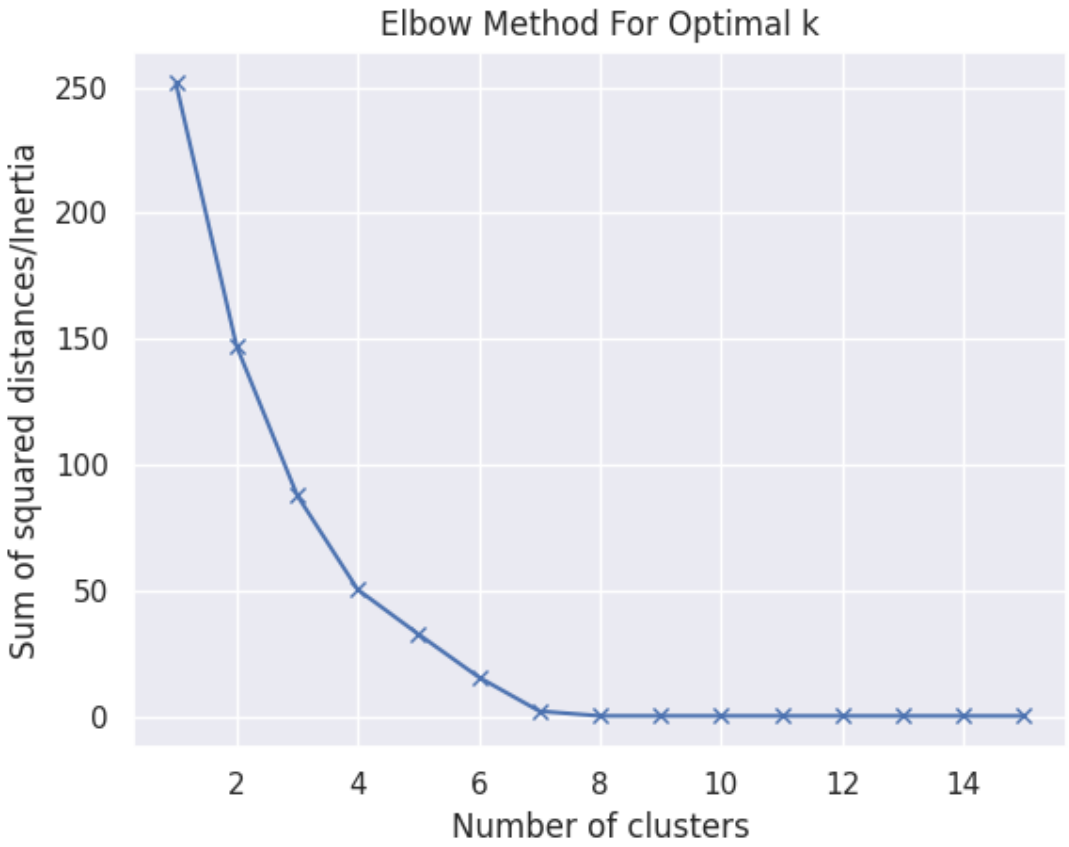
QUESTION 3:



For 2D dataset, optimal value of k = 4



For 3D dataset, optimal value of k = 4



For 4D dataset, optimal value of k = 7