ASSIGNMENT-2 REPORT

BY

BOMMAKANTI VENKATA NAGA SAI ADITYA
2019CS50471

APPAKONDA ABHIJITH REDDY
2019CS10330

KURISETI RAVI SRI TEJA
2019CS10369

Contents

1 Problem-1

1.1 Pafi/FSG

Support Threshold	**Time Taken to Run in seconds**
5%	3952.40
10%	1308.47
25%	370.33
50%	127.57
95%	22.19

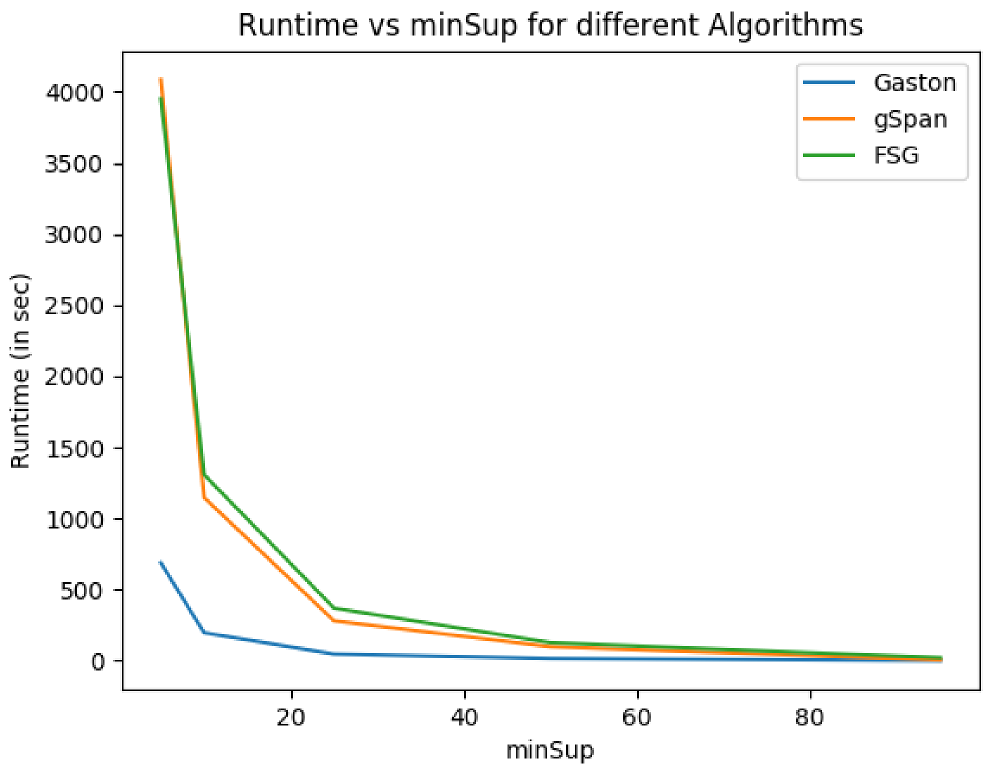
1.2 gSpan

Support Threshold	**Time Taken to Run in seconds**
5%	4086.13
10%	1147.86
25%	280.58
50%	99.125
95%	5.33

1.3 Gaston

Support Threshold	Time Taken to Run in seconds
5%	689.09
10%	197.23
25%	47.56
50%	16.01
95%	1.11

1.4 Plots of Runtimes of Various Algorithms



All runtimes reported on **Yeast** dataset.

1.5 Observations

1. gSpan and Gaston uses Depth-First Approach.

2. Pafi(FSG) uses a Breadth-First Search approach.

3. gSpan is faster than FSG because FSG uses an apriori approach in which all candidates are generated.In FSG,we need to check isomorphism of every generated candidate which takes a lot of time as opposed in gSpan where a lot of candidates are eliminated before generation thereby making gSpan faster than FSG. In gSpan, there is no candidate generation step and no false testing. All frequent (k + 1) edge subgraphs grow from k-edge frequent subgraphs directly. At each iteration of gSpan the graph database is quickly shrunk to the one containing smaller set of graphs because we prune a subtree of DFS Code Tree at a node, if the node is not minimal DFS Code. Also, DFS codes are generated only by extending the right most path. So, gSpan is better than FSG.

4. Gaston is faster than gSpan because in Gaston we initially mine patterns of type paths and trees whose isomorphism is easier to check for and then move to complex graphs and hence this reduces the time to check for isomorphism thereby reducing the total time taken as compared to gSpan where we proceed by adding edges of particular type in each step which could create complex cycles in the early stages of search which takes more time to check isomorphism.

5. Hence order of times taken will be FSG>gSpan>Gaston which is evident from the above plots.

2 Problem-2

The subgraph patterns mined from D are frequent Subgraphs of D. We used gSpan to mine frequent subgraphs of the given database while indexing. We used frequency threshold as 30% for gSpan. Now, during query processing first we load the index structure. Then, path to query file is taken as input. Initially, all graphs are candidate graphs. Now, for each frequent pattern f in index structure, we check if it is subgraph isomorphic to query graph. If yes, the candidates graphs will be the intersection of candidate graphs so far and the graphs containing f. So, the graphs in the given database get pruned. Now, we check if the given query graph is subgraph isomorphic to the candidate graphs. Then, we get the output.

If frequency threshold is high, then candidate graphs will be high. So, there will be more subgraph isomorphism checks. If frequency threshold is low, frequent subgraphs will be high, so again there will be more sub-graph isomorphism checks. We tried different values of frequency threshold and found 30% as optimal.

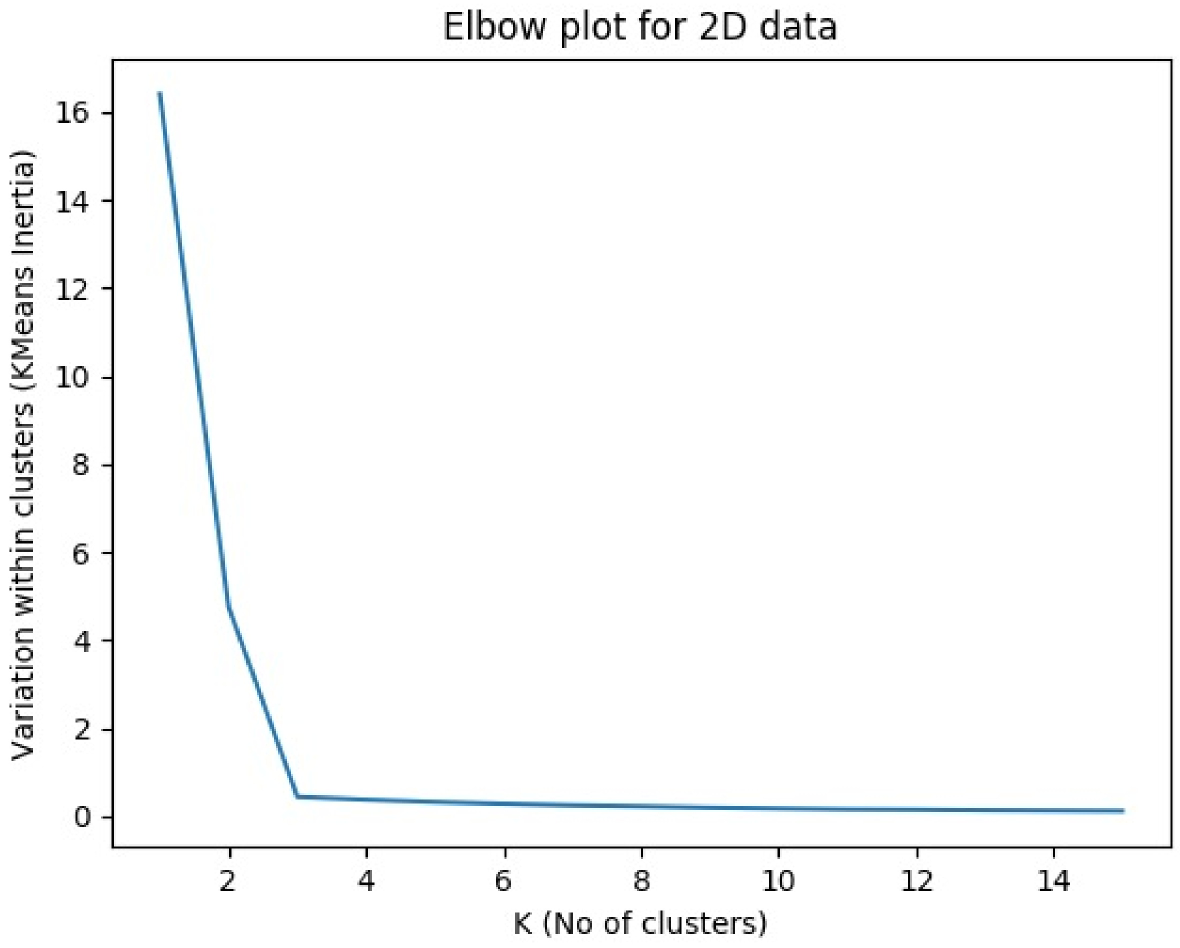
The value of 'm' is no of frequent subgrpahs of given database with support threshold = 30%

The output is stored in output_CS5190471.txt file. Each line corresponds to a graph is query graph file. The graphs ID's are not sorted.

3 Problem-3

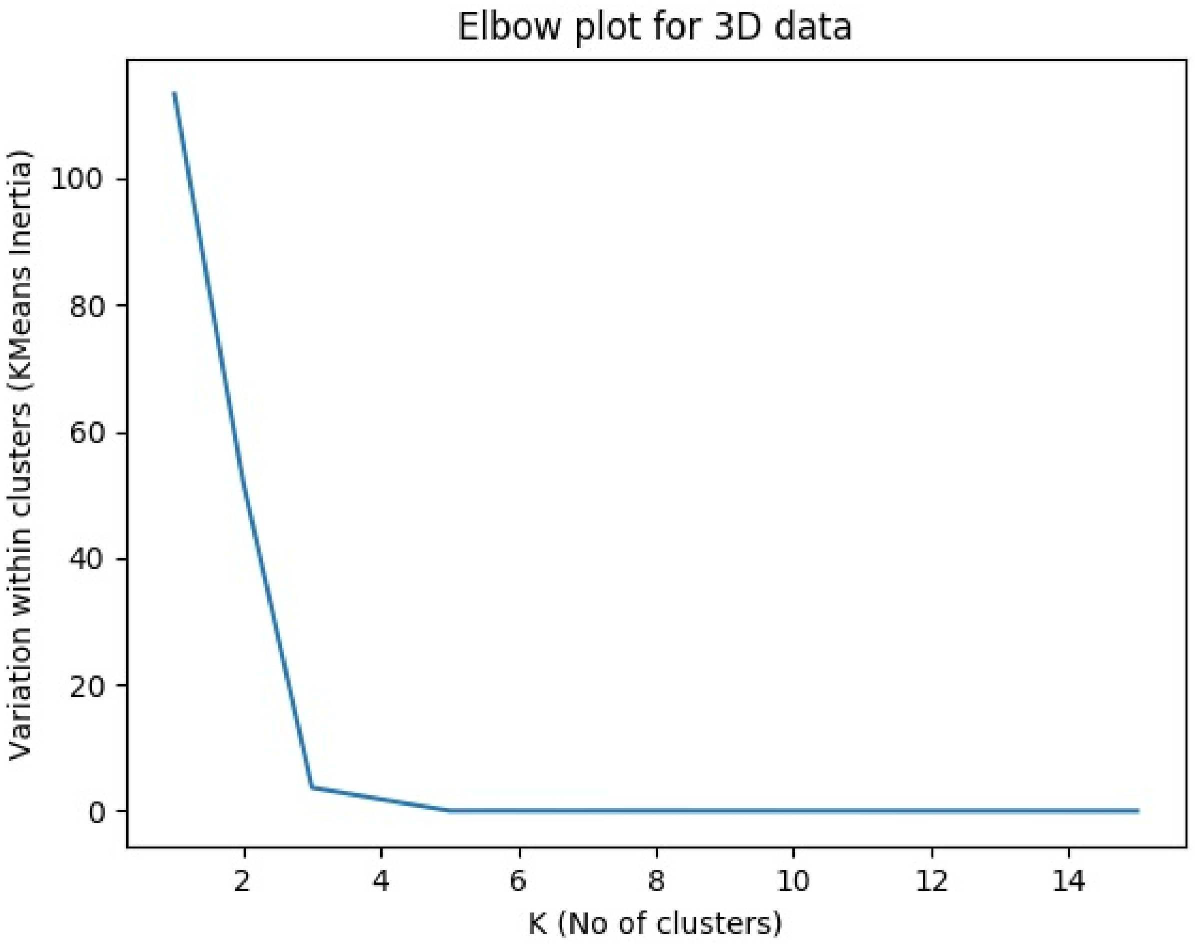
3.1 k-Means Clustering Plots

3.1.1 Plot for Dimension=2



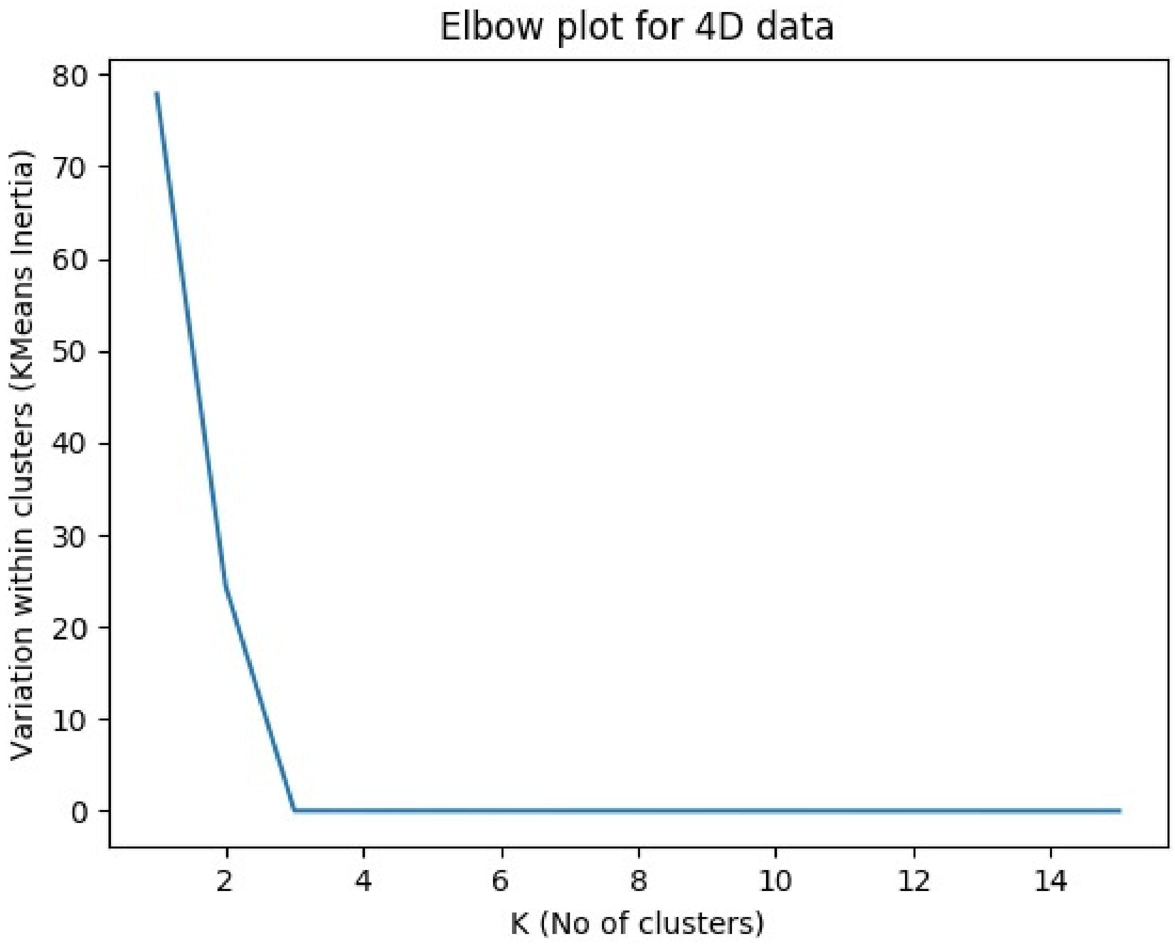
Number of clusters=**3**

3.1.2 Plot for Dimension=3



Number of clusters=**3**

3.1.3 Plot for Dimension=4



Number of clusters=**3**
Reason: The optimal number of clusters is 3 because after k = 3, the kmeans inertia decreases linearly. So, k = 3 is the elbow in all these plots. So, optimal number of clusters is 3.