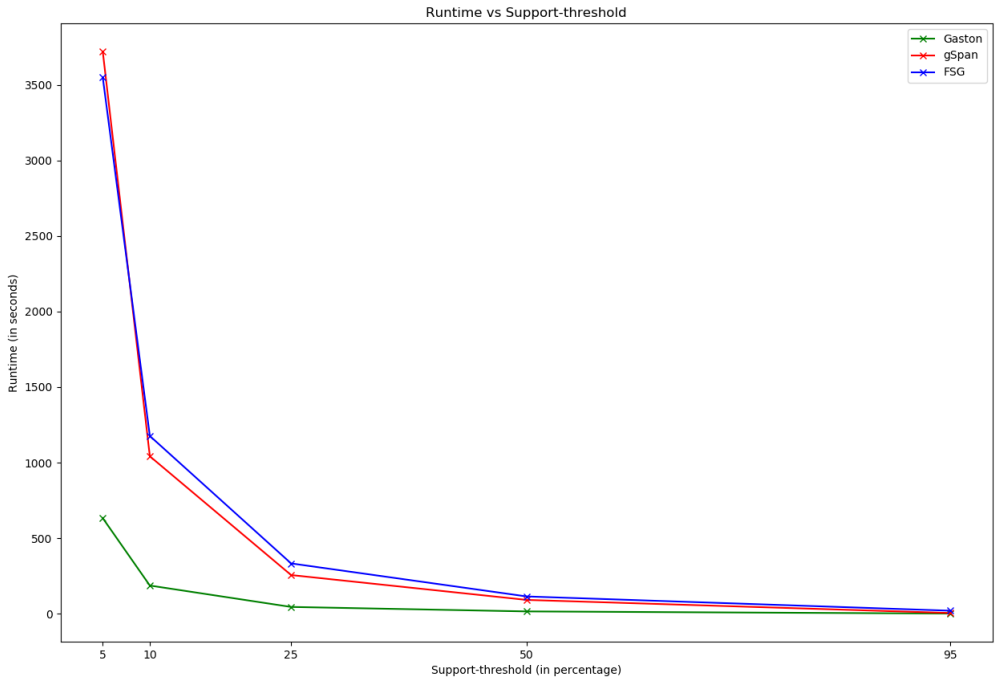
COL 761
Data Mining HW2

Members:
Dhruvil Sheth: 2022AIB2689
Pratik Vora: 2022AIB2687

Q1. Plot the running times and explain the trend observed in the running times. Specifically, comment on the growth rates and why one technique is faster than the others



Gaston is the fastest of all three. gSpan is faster than FSG. The order of all 3 algorithms in terms of running times is FSG > gSpan > Gaston.

FSG uses the BFS technique in which expansion of the subgraph occurs by 1 edge at a time whereas gSpan uses the rightmost expansion. FSG has an Apriori-based algorithmic approach, while gSpan and Gaston perform Pattern growth-based approach.

Thus, gSpan is faster as there is no candidate generation and false test. Frequent k+1 edge subgraphs grow from k edge frequent subgraphs directly. Also, the dataset shrinks quickly. At each iteration, the mining is performed in such a way that the whole graph dataset is shrunk to a smaller set of graphs with lesser edges and vertices.

gSpan uses the canonical representation of graphs called DFS code which determines the order of traversal of the edges and the concatenation of the edges. It uses the rightmost extension. Since it does not store embedding, subgraph isomorphism testing is done on all graphs in the appearance list.

While Gaston stores all the embedding and thus has fast isomorphism testing. Also, Gaston uses a hash table to avoid duplicates.

Also, we can notice that Gaston is fast at lower support thresholds but have relatively similar time as the support threshold increases.

At lower support values, frequent subgraphs mined are higher which increases in an exponential manner.

The decrease in time with increasing support threshold is exponential for FSG and gSpan but linear for Gaston.

gSpan does not generate candidates like FSG and is thus faster. Gaston is the fastest as it mines subtrees, then subgraphs with cycles due to which the subgraph isomorphism testing which is quite expensive happens only in the subgraph mining phase since for trees and paths efficient isomorphism algorithms exist. Gaston generates cycles that are larger than previously generated graph cycles.

Q2. Index based elimination using gSpan to speedup the subgraph isomorphism test.

We used gSpan-64 binary to first mine out frequent subgraph patterns with support>=40%. We obtained 372 such patterns (with edges >= 1). Top 50 subgraphs with highest supports were chosen to form the index of the graphs. After creating and loading the indices, we transformed every transaction in the database in form of feature vectors.

After the successful loading indices, we added a prompt for the user to provide a query file path. The query file is read to form query graph and its feature vector (using boost library's vf2 subgraph isomorphism test). The transactions from the database not containing this feature vector are eliminated and the rest are checked upon using the same vf2 test.

For dataset: Yeast/167.txt_graph and query: sample_dataset.txt

Output with and without indexing: (Note: Transaction IDs obtained for a query are sorted lexicographically)

42614805

25077042

11970112 25077041

NULL

NULL

NULL

NULL

NULL

NULL

NULL

(Note: NULL is only representation here. In the actual output file, the line is kept empty signalling lack of transactions containing particular query graph).

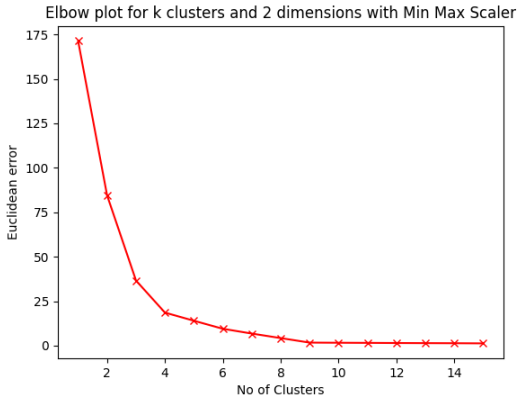
Time taken: (sum total for all the queries)

Without indexing: 298000ms ~ 300s (5 mins) With indexing: 150000ms ~ 150s (2.5 mins)

Q3. Elbow plot to determine the correct value of k in k-means clustering on the dataset.

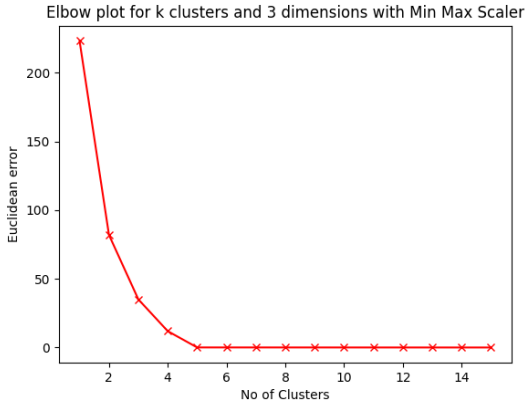
We have used Min Max Scaler to pre-process the data.
Dataset used – AIB222689_generated_dataset_2D, AIB222689_generated_dataset_3D, AIB222689_generated_dataset_4D

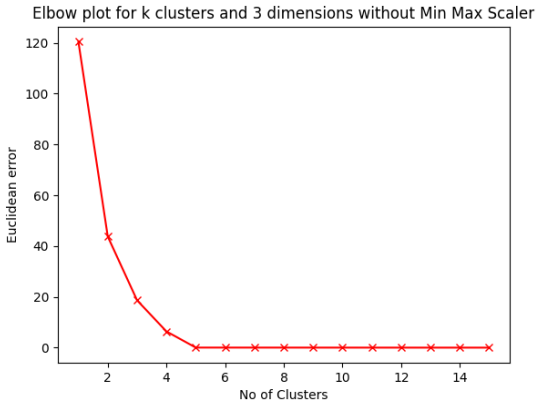
Dimension = 2



Optimal k for Dimension 2 = 4

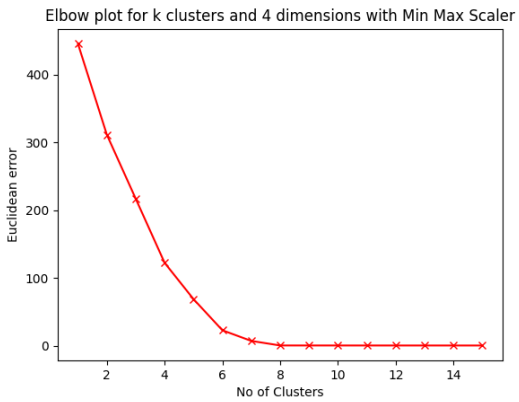
Dimension = 3

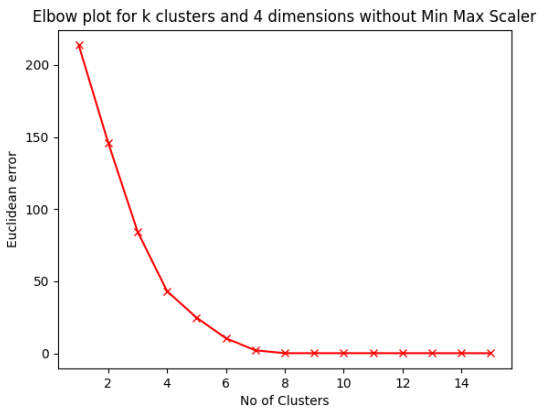




Optimal k for Dimension 3 = 4

Dimension = 4





Optimal k for Dimension 4 = 4