Dept. of Computer Science and Engineering

Data Mining, COL761

Assignment 2

Group Members:
Ramneet Singh, 2019CS50445
Aryan Jain, 2019CS10334
Anjali Sharma, 2019CS50422

Supervisors:
Prof. Sayan Ranu

October 2022

1 gSpan vs FSG vs Gaston

- For all methods runtime increases with decreasing min sup values as number of frequent subgraphs increase with a decrease in support.
- The **growth rate** of runtimes increases with decrease in support values. As frequent subgraphs increase exponentially with decrease in support values.Furthermore, for high support values growth rate becomes constant for the same reason. For Gaston growth rate is almost linear for all the supports.
- On YEAST dataset FSG and gSpan have similar runtimes over different supports but gSpan is slightly faster than FSG which is expected.gSpan is faster as it avoids cost intensive problems like redundant candidate generation and isomorphism testing in comparison to FSG.
- FSG uses a bfs startegy that creates candidate subgraphs of size k+1 using k size subgraphs and then prunes the infrequent ones. On the other hand, gSpan do not generate candidates due to construction of DFS Code tree and follow depth first based pattern generation like fp growth and thus do not require pruning.
- Gaston is very fast as compared to other two methods as it finds frequent subgraphs in a number of phases of increasing complexity.Gaston uses an **embedding list** structure.It stores all embeddings, to generate only refinements that actually appear and to achieve fast isomorphism testing.
- Gaston performs the best as it proceeds with mining paths and then mining subtrees and then mining subgraphs with cycles. The expensive subgraphs isomorphism testing occurs only in subgraph mining phase. This approach helps as for paths and trees efficient isomorphism tests exist. For graphs,Gaston algorithm generates cycles that are larger than previously generated graphs cycles.

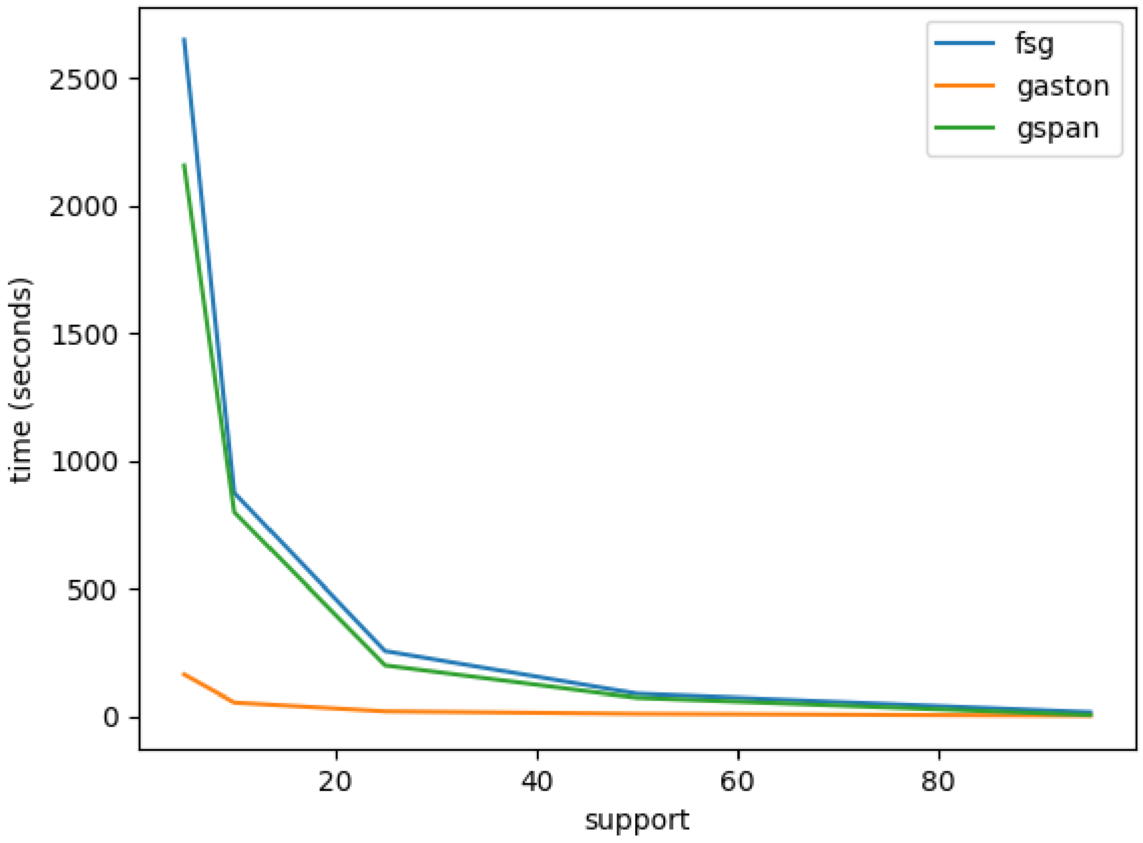


Figure 1: Running time Analysis for gSpan, FSG and Gaston

2 Elbow Plot

2.1 For dimension 2

- Figure 2 shows Elbow plot for generated dataset for dimension 2.
- We observe elbow point from graph to be 9 and also calculate it using a plot of ratio of slopes vs clusters.
- This is validated from Figure 3 (peak at x=9) and Figure 4 (dataset shows 9 clusters).

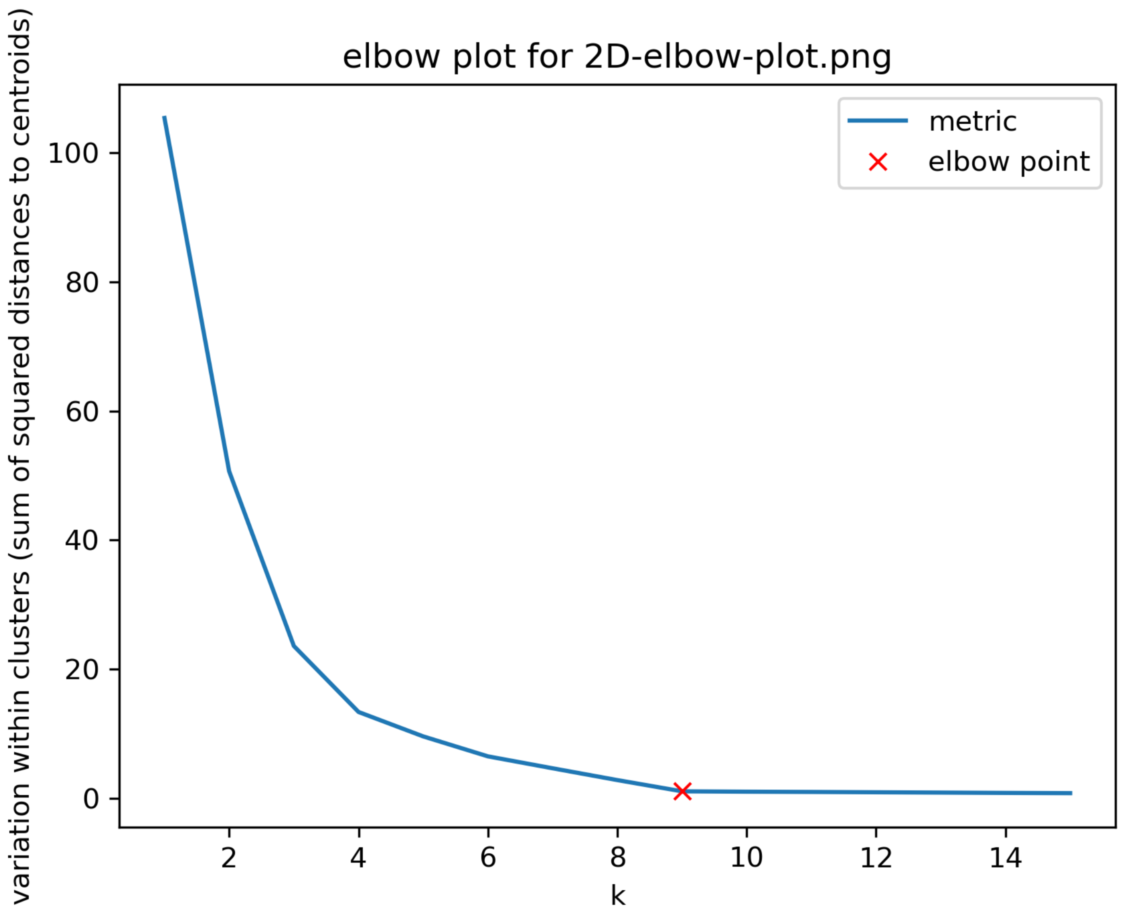
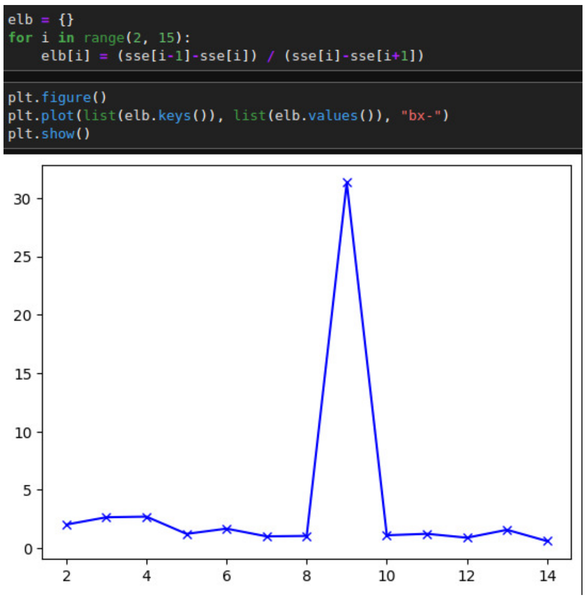


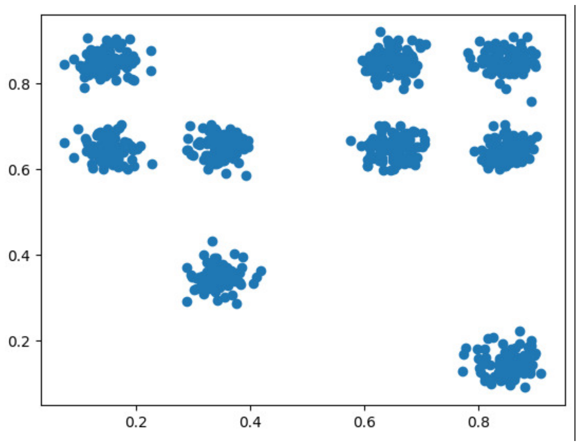
Figure 2: Elbow plot for 2 dimensions



```
elb = {}
for i in range(2, 15):
    elb[i] = (sse[i-1]-sse[i]) / (sse[i]-sse[i+1])
```

```
plt.figure()
plt.plot(list(elb.keys()), list(elb.values()), "bx-")
plt.show()
```

(a) Plot for ratio of slopes for 2D



(b) 2D Dataset - shows 9 clusters

2.2 For dimension 3

- Figure 4 shows Elbow plot for generated dataset for dimension 3 and k=4.
- We observe elbow point from graph to be 4 and also calculate it using a plot of ratio of slopes vs clusters.
- This is validated from Figure 5a (peak at x=4) and Figure 5b (dataset shows 4 clusters).

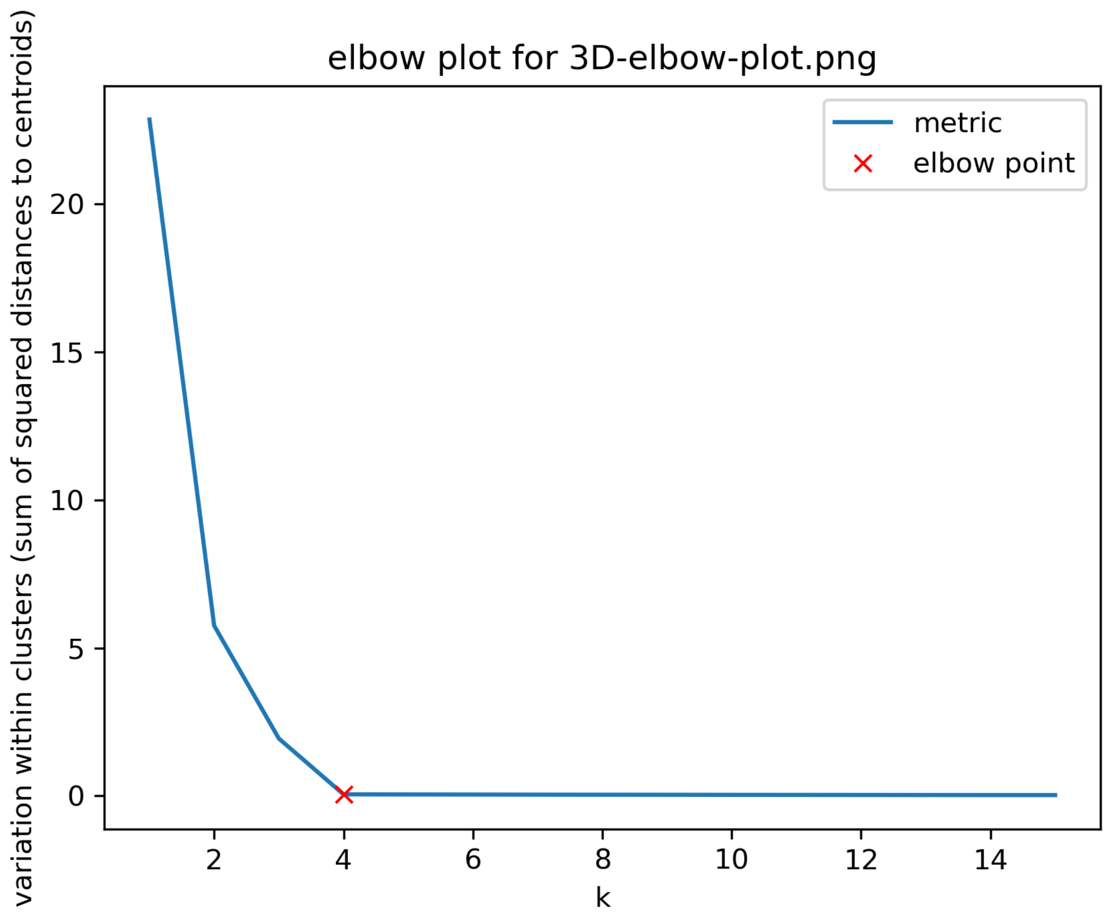
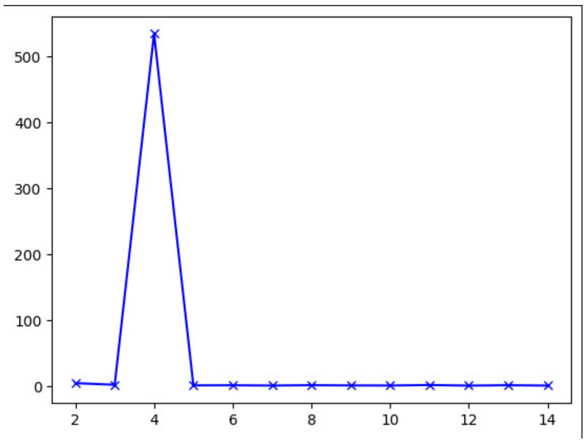
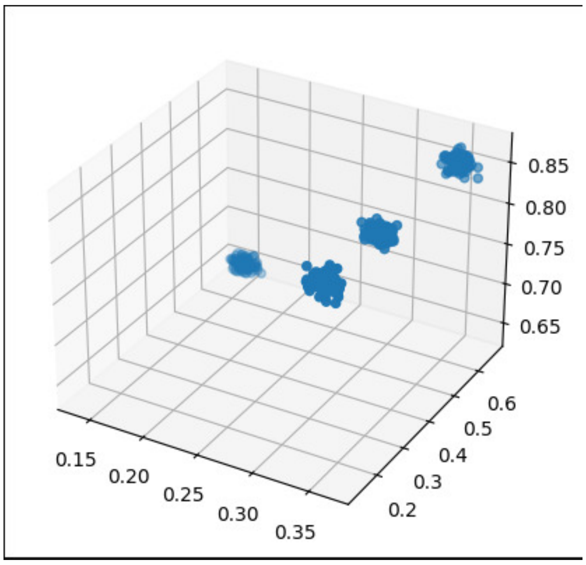


Figure 4: Elbow plot for 3 dimensions



(a) Plot for ratio of slopes for 3D



(b) 3D Dataset - shows 4 clusters

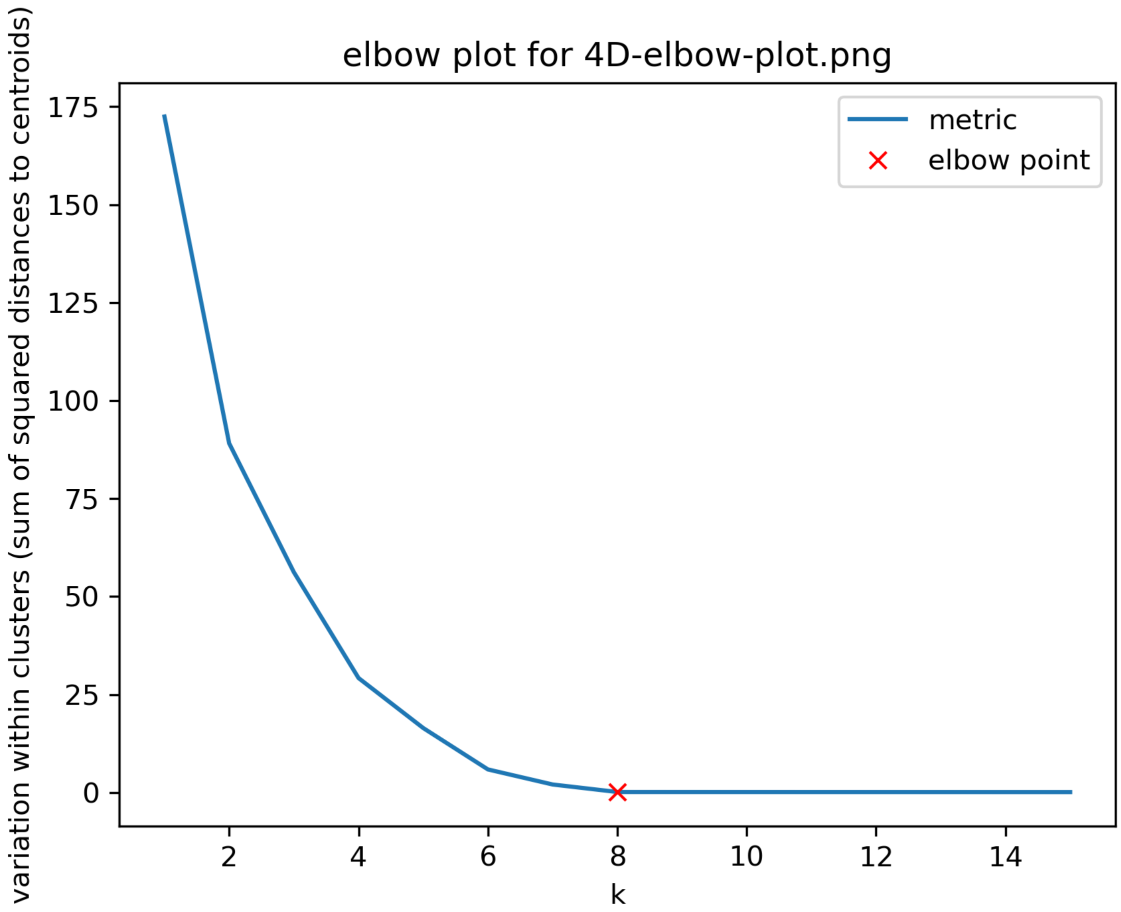


Figure 6: Elbow plot for dimension 4 , k=8

3 Subgraph Search

We run gaston on the input dataset with support threshold depending on the number of graphs. Then, we prune out graphs which are not sufficiently discriminative based on the factor gamma as mentioned in the paper "Graph Indexing: A Frequent Structure-based Approach" by Yan et. al. The discriminative frequent subgraphs are used as features. The features are then used as mentioned in the assignment document to prune the graph search space. Finally we use subgraph isomorphism on the remaining graphs in the search space. Note: We don't sort the ids when we output them for a given query.