INDIAN INSTITUTE OF TECHNOLOGY DELHI

COL761

Report

Assignment − 2

Abstract

The aim of this assignment is to experiment with different frequent subgraph mining algorithms and use this knowlege to design a pipeline for fast querying of subgraphs given an input database of graphs. Additionaly, we also analyze the elbow plot for K-Means algorithm and attempt to explain the number of clusters through it.

Harsh Agrawal	2019CS10431
Pranjal Aggarwal	2019CS50443

1 Frequent Subgraph Mining

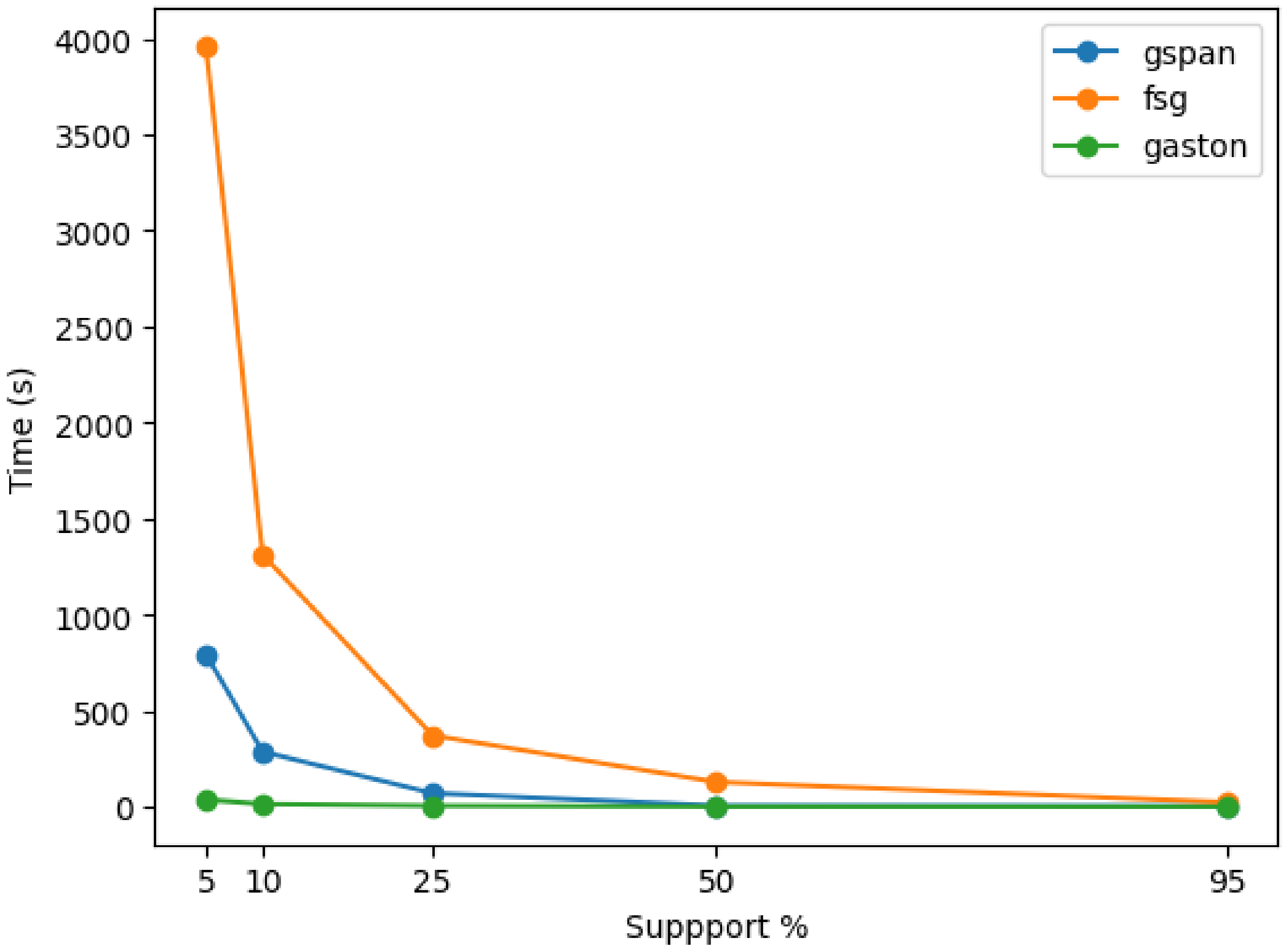

Figure 1: Runtimes of various algorithms at changing support level.

- **FSG:** It can be clearly observed that the running time of FSG is the highest. This is because it, needs to scan the database for every candidate set and check if they meet the requisite support. Additionally, the canonical labelling can often be very expensive in case of symmetric graphs (which is usually the case with chemical compounds).
- **Gspan:** Gspan requires considerably lower time than FSG, because of its depth first approach and various optimizations that can be applied therein. Being a depth-first algorithm, the space required is much lesser and the cost of I/O operations is considerably lower. Moreover, at increasing depths the number of candidate graphs in the database rapidly shrinks which makes it easier to execute the support test. Also, computing DFS code is relatively cheap than canonical labelling which has no guarantees on the runtime. Note that during join of two k-size subgraphs, FSG considers all possible candidates, whereas Gspan grows the subgraph in a predefined direction, pruning away graphs if they are already checked or are infrequent.
- **Gaston:** It can be observed that the runtime of Gaston is lowest irrespective of the support. This follows from a novel observation that most of the frequent subgraphs are simple and rarely complex in structure. That is to say, simple paths and trees are more common in real-life datasets than complex cylic graphs. Using a novel strategy where graphs are extended in the order path $\rightarrow$ tree $\rightarrow$ graph, it was able to achieve exceptionally low runtimes on the chemical compound dataset.

2 Index structure

We highlight the approach we used to solve the problem.

- Mined frequent subgraphs at a support of 10%. We employed `gaston` [1]for the task as it was relatively fast.
- We built an index structure in the form of a matrix M where $M_{ij} = 1$ if i^{th} graph contains the j^{th} frequent subgraph and 0 otherwise.
- Additionally, we also built a tree such that a frequent subgraph G_p is parent of a frequent subgraph G_c if G_c was formed by addition of a single edge in G_p.
- During the querying stage, we use the **Uniform Cost Search** algorithm to expand nodes for subgraph isomorphism tests. The aim is to reduce the number of subgraph isomorphisms using **VF3** [2] as much as possible. We start with a priority queue adding all the root node(s) of our tree. The priority queue priortizes node on the value of their support in the database. If for a given subgraph in the priority queue, it is not the subgraph of the query graph then its subtree is not visited and it is simply popped from the queue. Otherwise all of its children are added to the priority queue. At any stage we have the list of all the remaining candidate graphs that have the same features as those currently tested. If at any stage the number of candidates is less than a certain percentage of the number of features left to test, the search is terminated, and subgraph isomorphism is performed on all the remaining candidate graphs.

Using the above algorithm we are able to achieve really fast runtimes on the benchmarking datasets that were provided.

[1] Gaston does not output the transaction ids for each subgraph. We patched the source code to also output the transaction ids.

[2] The original version of VF3 does not support edge labels, we patched the source code to support edge labels

3 Clustering - Elbow Plot

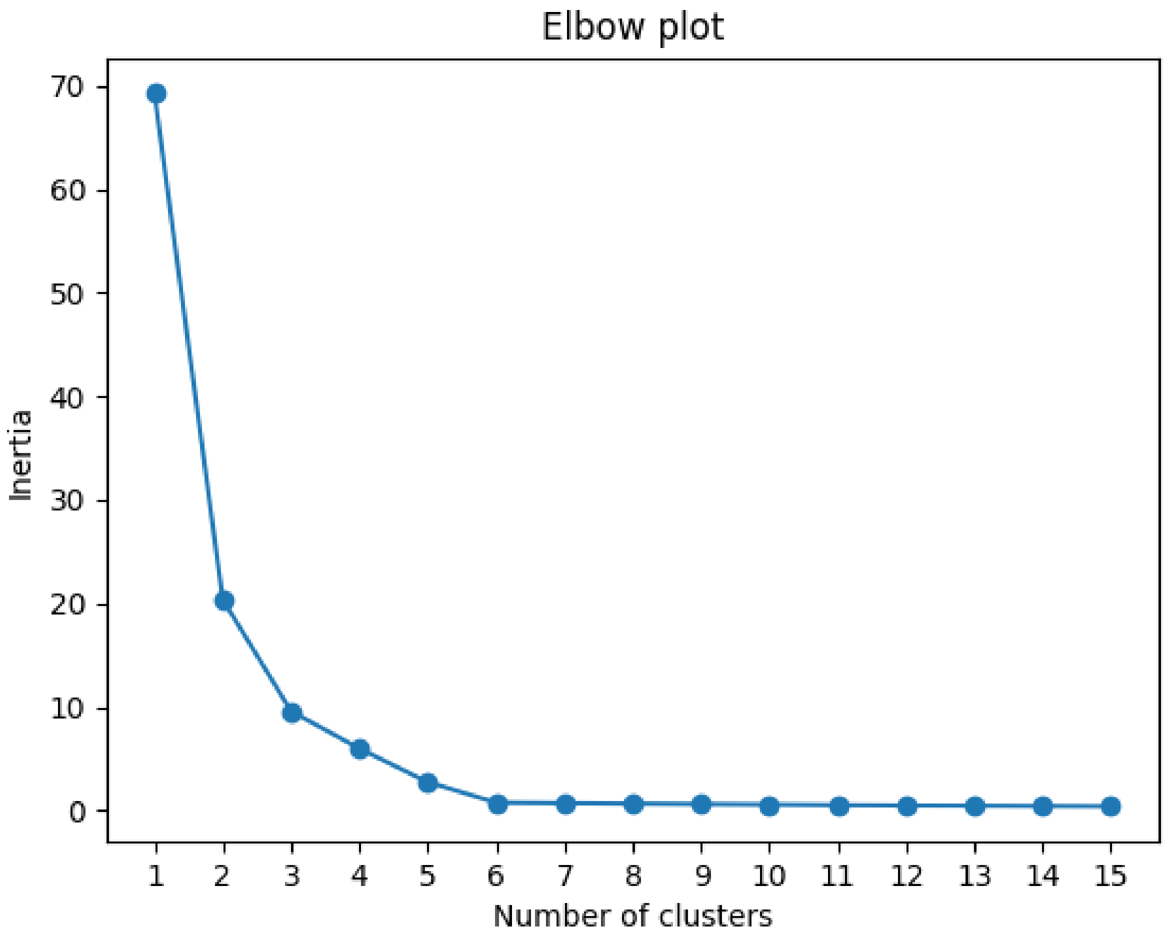

Figure 2: Elbow Plot for $d = 4$

From the elbow plot it is visible that the inertia rapidly decreases till $K = 6$ and remains almost constant thereafter. This means the clustering is almost complete when $K = 6$ and any more clusters do not add any value to our clustering and might themselves be very near to each other. Thus $K = 6$ is the optimal number cluster centers value for this dataset.