COL761: Assignment 1 - Part 2

Abhay Sharma	Akash Dabaniya	Mridul Singh
2012CS50272	2021CS1011	2018CS50412

February 4, 2026

1 Part 2: Frequent Subgraph Mining

1.1 Introduction

Frequent Subgraph Mining (FSM) is a technique used to find common subgraph patterns in a set of graphs.
FSM is more difficult because graphs have complex structures and checking graph similarity is computationally expensive.
Here compare the runtime performance of three well-known algorithms FSG, gSpan, and Gaston at different minimum support values. The experiments are done out using the Yeast dataset, which is widely used in biological graph mining.

1.2 Dataset Description

The yeast dataset contains graphs that have biological structures, such as protein interaction networks or molecular structures.

1.3 Algorithms Overview

1.3.1 1. FSG (Frequent Subgraph Discovery)

FSG is a classical algorithm based on the Apriori principle and follows a Breadth-First Search (BFS) approach. It was originally proposed by Kuramochi and Karypis [1].

- **Mechanism:** FSG works in a level-by-level manner, similar to frequent itemset mining.
  - In the first step, it finds all frequent subgraphs containing only one edge.
  - In the next step, it joins two frequent subgraphs of size k to create candidate subgraphs of size $k + 1$.
  - Each generated candidate is checked against the dataset to see whether it appears frequently enough.

  Example: Suppose we have frequent subgraphs with edges (A–B) and (B–C). FSG will combine them to form a new candidate graph A–B–C. This new graph is then checked in all graphs of the dataset to see how many times it occurs.

- **Cons:** FSG generates a very large number of candidate subgraphs. For each candidate, it performs subgraph isomorphism checking, which is a costly and slow operation. As the graph size increases, this process becomes very time-consuming, making FSG unsuitable for large or dense datasets.

1.3.2 2. gSpan (Graph-Based Substructure Pattern Mining)

gSpan is a pattern-growth based algorithm that follows a Depth-First Search (DFS) strategy, introduced by Yan and Han [2].

- **Mechanism:** gSpan avoids candidate generation completely.
  - It starts with a small frequent subgraph.
  - The subgraph is expanded by adding one edge at a time in a depth-first manner.
  - Each graph is represented using a DFS code, which gives a unique string representation for that graph.
  - Only graphs with the minimum (canonical) DFS code are explored further, which avoids duplicate patterns.

  Example: If a graph can be drawn in two different ways, gSpan converts both into DFS codes. Only the one with the smallest DFS code is considered valid. This ensures the same graph is not mined multiple times.

- **Pros:** Since gSpan does not create candidate graphs explicitly, it saves memory and reduces unnecessary computation. This makes it faster than FSG in most cases.

1.3.3 3. Gaston (Graph / Sequence / Tree Extraction)

Gaston is a hybrid algorithm developed by Nijssen and Kok [3], based on the observation that most frequent patterns in real-world graph datasets are simple paths or trees, not full cyclic graphs.

- **Mechanism:** Gaston divides the mining process into three clear phases:

  1. **Path Mining:** In this phase, Gaston finds simple linear paths. This step is very fast and runs in linear time.
  2. **Tree Mining:** The frequent paths are extended to form tree structures. Since trees do not contain cycles, checking them is still efficient.
  3. **Graph Mining:** Only when required, trees are further expanded into cyclic graphs. Expensive subgraph isomorphism checks are performed only at this stage. ¡/enumerate

     Example: In a molecular dataset, a chain like C–C–O is first found as a path. This path can be extended to a tree by adding branches. Only if a ring structure is possible, Gaston checks for cyclic graphs, avoiding unnecessary computations.
  4. **Pros:** Gaston is very efficient because it postpones expensive graph isomorphism checks. Most patterns are discovered during the fast path and tree phases, making Gaston the fastest among the three algorithms for many datasets.

1.4 Experimental Results

All three algorithms were run on the Yeast dataset using minimum support values of 5%, 10%, 25%, 50%, and 95%. For each case, the total running time was noted. The comparison of running time is shown in Figure 1. The y-axis is shown in logarithmic scale so that the difference in performance at low support values can be seen clearly.

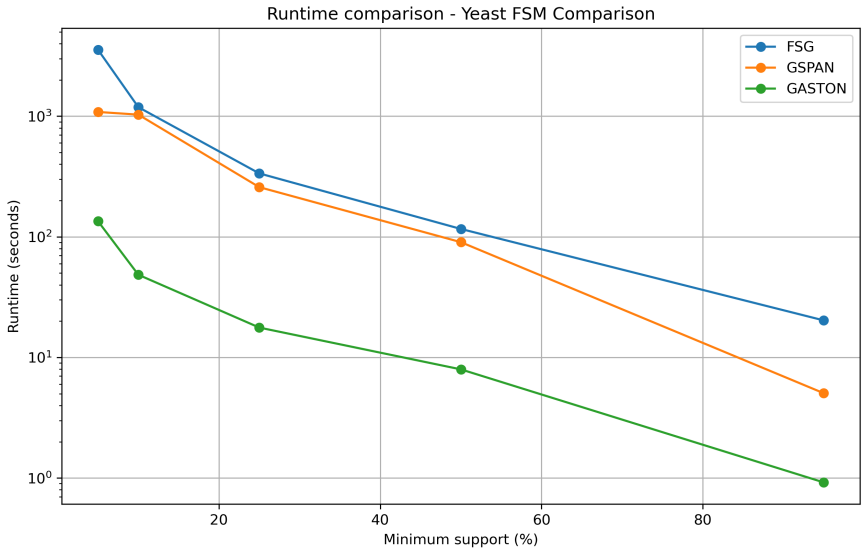

Figure 1: Running time comparison of FSG, gSpan, and Gaston on the Yeast dataset for different minimum support values (logarithmic scale).

1.5 Analysis of Observations

- **Low Support (5%–10%):** At low support values, FSG takes the maximum time. This is because it creates many candidate graphs and repeatedly checks subgraph matching, which is a difficult problem. gSpan performs much better since it does not generate candidates, but it still takes extra time due to DFS code checking. Gaston performs the best, as most of the frequent patterns are simple paths or trees, which are easy to find.
- **Medium Support (25%–50%):** When the support value increases, many unwanted subgraphs are removed early. Because of this, the running time of all algorithms reduces. The time difference between FSG and gSpan becomes smaller, while Gaston continues to be the fastest.
- **High Support (95%):** At very high support values, all three algorithms take less time. Only a few simple subgraphs satisfy the support condition. Most of the time is spent in scanning the dataset rather than expanding graph structures.

References

[1] M. Kuramochi and G. Karypis, "Frequent Subgraph Discovery," *ICDM*, 2001.

[2] X. Yan and J. Han, "gSpan: Graph-Based Substructure Pattern Mining," *ICDM*, 2002.

[3] S. Nijssen and J. Kok, "Gaston: Efficient Frequent Subgraph Mining," *KDD*, 2004.

Disclaimer

Large Language Models were used during this assignment for clarifying the workflow of graph indexing pipelines and for assistance with LaTeX syntax. The implementation logic and experimental conclusions are our own work.