Assignment 1 - Part 2

Frequent Subgraph Mining Analysis

Team: JCB Diggers

Department of Computer Science and Engineering
IIT Delhi

February 2, 2026

1 Introduction

Frequent pattern mining is a fundamental task in data mining with applications in various domains, including bioinformatics, chemical compound analysis, and social network analysis. Frequent Subgraph Mining works by finding subgraphs that frequently occur in a graph database.

This report presents a comparative performance analysis for frequent subgraph mining algorithms: **gSpan**, **FSG**, and **Gaston** using the Yeast dataset. Our results demonstrate the significant performance advantages of Gaston over the other algorithms for low support thresholds.

2 Algorithms being compared

- **gSpan**: A depth-first search-based algorithm that uses a canonical labeling system (DFS Code) to avoid redundant subgraph exploration. It extends subgraphs by adding edges one at a time.
- **FSG (Fast SubGraph)**: A breadth-first search-based algorithm that generates candidate subgraphs by joining smaller frequent subgraphs. It uses a canonical representation to avoid duplicates. Also known as PAFI.
- **Gaston**: An algorithm that combines aspects of both depth-first and breadth-first search. It efficiently mines frequent paths and trees before exploring more complex cyclic graphs.

3 Details of the Experiment

- **Dataset**: Yeast dataset.
- **Libraries**: Compiled binaries for gSpan, FSG, and Gaston.
- **Support Thresholds**: 95%, 50%, 25%, 10%, and 5%.
- **Environment**: Linux Virtual Machine (Baadal).
- **Platform**: Baadal platform (IIT Delhi).
- **Metrics**: Runtime in seconds.

4 Results

Table 1: Runtime comparison of gSpan, FSG, and Gaston on the Yeast dataset.

Support Threshold (%)	**gSpan Runtime (s)**	**FSG Runtime (s)**	**Gaston Runtime (s)**
95	4	20	1
50	81	108	13
25	240	309	35
10	916	1107	137
5	3094	3435	478

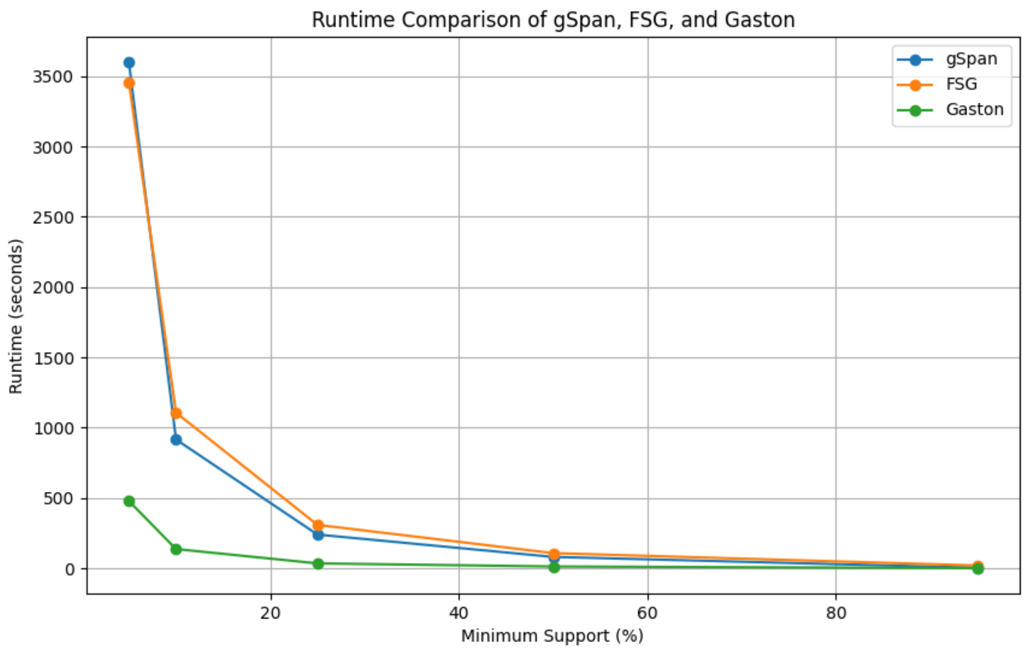

Figure 1: Runtime comparison of gSpan, FSG, and Gaston across different support levels.

5 Analysis and Discussion

Figure 1 and Table 1 clearly demonstrate the performance difference between gSpan, FSG, and Gaston.

- **Performance Hierarchy**: Across all support levels, **Gaston** consistently outperformed both gSpan and FSG by a significant margin. It was approximately 6–7 times faster than gSpan and 7–8 times faster than FSG at lower support levels.
- **Scalability**: As the support threshold decreases, the runtime for all algorithms increased exponentially. This is expected as the number of frequent subgraphs grows rapidly with lower support.
- **Gaston's Advantage**: Gaston proved to be the most efficient, likely due to its hybrid approach. By prioritizing paths and trees (which are cheaper to mine) before cyclic graphs, Gaston avoids much of the expensive isomorphism testing that purely DFS or BFS approaches might incur early on.
- **Comparison of gSpan and FSG**:
  - **gSpan (DFS)** performed better than FSG. Its pattern-growth approach using DFS codes avoids the expensive candidate generation and join steps required by Apriori-like algorithms.
  - **FSG (BFS)** was generally the slowest. Being a Level-wise (Apriori-based) approach, it must generate and verify candidate subgraphs at each size level (k-patterns from (k-1)-patterns), which becomes a major bottleneck as the graph size increases. However at threshold = 5%, FSG has given a better performance than gSpan.

6 References

- For code refinement, and grammatical helps: ChatGPT
- For latex formatting and refinement and research: Perplexity