COL761/COL7361/AIL7026 Data Mining

Assignment 1, Semester-II, 2025-26

Submitted by

Team Eleventh Hour

Q2 Report: Frequent Subgraph Mining – Performance Comparison of gSpan, FSG, and Gaston

1. Introduction

Frequent Subgraph Mining (FSM) focuses on identifying subgraphs that appear repeatedly across a collection of graphs. Given a set of graph transactions, where each instance is represented as a graph, FSM aims to discover structural patterns that occur above the user-defined support threshold. It has important applications in graph-structured data analysis, with key applications in bioinformatics, chemical compound analysis, and social networks.

In this task, we evaluate three classical FSM algorithms:

- **FSG**
- **gSpan**
- **Gaston**

Experimented using the **Yeast dataset** at support thresholds **5%, 10%, 25%, 50%, and 95%**. For each algorithm, execution time was recorded and plotted against its support thresholds and shown in Figure 1. Also the experiment required reformatting the dataset into the input formats expected by each library.

2. Algorithms Overview

2.1 FSG

FSG uses a **level-wise Apriori expansion** strategy and enumerates subgraphs by **growing patterns one edge at a time**. It relies on canonical labeling to avoid duplicates (Kuramochi & Karypis, 2004).

Key characteristics:

- Uses breadth-first search (BFS) candidate generation
- Uses canonical labeling to prune duplicates
- Frequency counting involves repeated subgraph isomorphism checks
- Highly sensitive to low support values because the number of candidates grows exponentially

This explains why FSG tends to be **slowest at low support thresholds** where many large patterns are frequent.

2.2 gSpan

gSpan introduces the **DFS-code**, a canonical labeling scheme that eliminates the need for candidate generation. Instead of Apriori-style enumeration, it uses **depth-first pattern growth**, which reduces the number of intermediate subgraphs produced.

Key characteristics:

- Uses lexicographically minimal DFS code to avoid expensive isomorphism checks
- Avoids BFS candidate explosions
- Efficient for sparse graph datasets

Its runtime is typically better than FSG at low thresholds because it avoids breadth-first combinatorial blow-ups.

2.3 Gaston

Gaston decomposes FSM into three categories: **paths**, **trees**, and **cyclic graphs**. It mines them in this order because:

Key characteristics:

- Paths are far cheaper to mine
- Trees are moderately expensive
- Cyclic graphs are most expensive

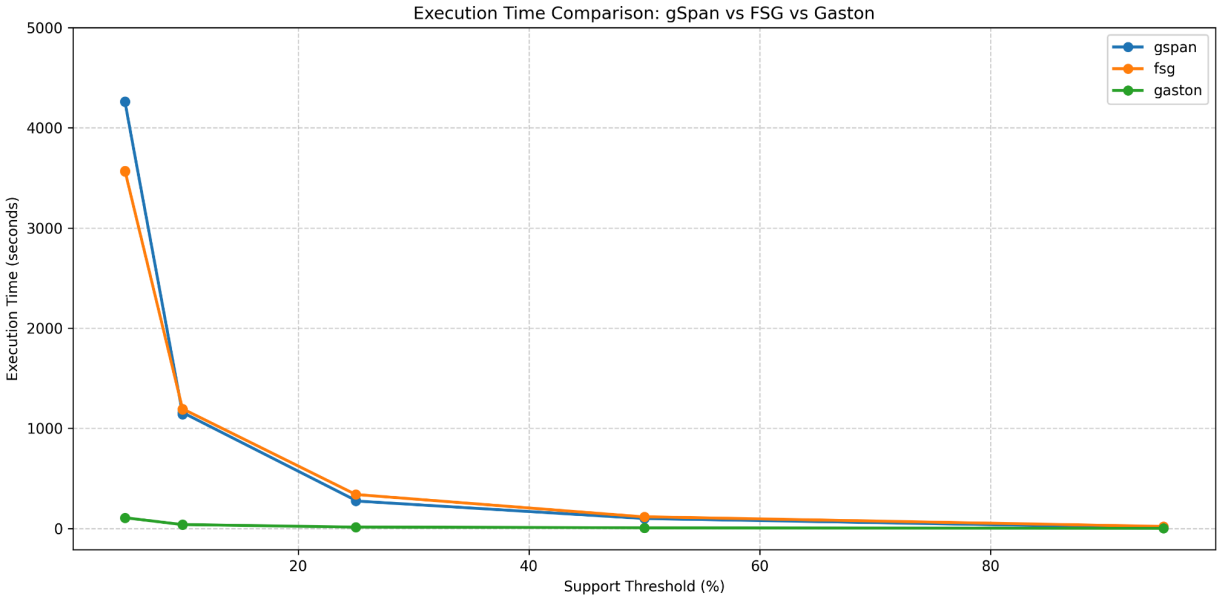
This hierarchical decomposition drastically reduces search space. This is why Gaston is usually **fastest**, especially at low support values, where paths dominate the frequent patterns.

3. Experimental Setup

- Dataset: **Yeast (graph dataset)**
- Minimum supports: **5%, 10%, 25%, 50%, 95%**
- Tools used: official implementations of **gSpan**, **FSG**, and **Gaston**
- Preprocessing: dataset converted to tool-specific formats
- Hardware: Standard Linux environment (Baadal), Python 3.10.12

4. Results

4.1 Execution Time Plot



5. Comparative Analysis

5.1 Trend with Increasing Support

As support threshold increases:

- The number of frequent subgraphs decreases exponentially**.**
- The search space shrinks and subgraph isomorphism checks reduce significantly.
- As a result execution time dropped sharply, consistent with all three algorithms.

5.2 FSG: The Apriori Bottleneck

- FSG uses a **breadth-first, level-wise** approach. It generates candidates of size k+1 by joining frequent subgraphs of size k.
- The candidate generation and pruning phases are extremely expensive. Each candidate requires a subgraph isomorphism test, which is NP-complete. In a complex dataset like Yeast, the number of candidates explodes at low thresholds.

5.3 gSpan: Eliminating Candidate Generation

gSpan introduced the **DFS Lexicographic Order** and **Minimum DFS Codes**.

- It maps each graph to a unique canonical label, allowing it to grow subgraphs depth-first without ever generating candidates.
- By avoiding the "generate-and-test" cycle of FSG, it prunes the search space much more effectively.

5.4 Gaston: Hierarchical Search-Space Decomposition

Gaston is the fastest because it recognizes that **most frequent subgraphs are simple structures.**

- Most frequent patterns in the Yeast dataset are paths or trees. Gaston uses highly optimized algorithms for these simpler structures and only invokes the expensive general graph mining logic when a cycle is actually detected.
- Unlike gSpan, which treats everything as a general graph from the start, Gaston "starts quick" and only slows down when necessary.

6. Conclusion

From the experiment and reference to the published algorithms:

- **Gaston is consistently fastest**, especially at very low support thresholds, due to its hierarchical mining and lightweight embedding structures.
- **gSpan performs better than FSG** because of DFS-code–based pruning and avoidance of candidate explosion.
- **FSG is the slowest**, particularly at 5% support, due to BFS-level candidate generation and more frequent isomorphism checks.

As support increases, **all three converge** to small execution times because the number of frequent subgraphs sharply declines. The observed trends clearly align with the theoretical expectations outlined in the respective research papers.

7. References

a. Research Papers

i. gSpan, X. Yan and J. Han, “gSpan: Graph-Based Substructure Pattern Mining,” *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2002.

ii. FSG / PAFI, M. Kuramochi and G. Karypis, “An Efficient Algorithm for Discovering Frequent Subgraphs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1038–1050, 2004.

iii. Gaston, N. Vanetik, Y. Gudes, and A. Shimony, “Computing Frequent Graph Patterns from Semistructured Data,” *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, 2002.

b. Libraries

i. gSpan Official Implementation

ii. FSG “PAFI: Frequent Subgraph Mining”

iii. Gaston Official Implementation

c. AI Tools

i. OpenAI, *ChatGPT (GPT-5.2)* [Large language model]