COL761: Data Mining - Assignment 1

Task 2: Frequent subgraph mining

Spandan 2022CS51138 Viraat 2022CS51651 Gaurav 2022CS51144

February 3, 2026

1 Introduction

This report presents an empirical comparison of three frequent subgraph mining algorithms: **FSG** (Frequent Subgraph Discovery), **gSpan** (Graph-based Substructure Pattern Mining), and **Gaston** (Graph/Sequence/Tree Extraction).

The objective was to analyze the runtime performance of these algorithms on the **Yeast** dataset (a molecular graph database) across varying minimum support thresholds (95%, 50%, 25%, 10%, and 5%). We investigate how the architectural differences between Apriori-based methods (FSG) and Pattern-Growth methods (gSpan, Gaston) impact scalability.

2 Experimental Setup

2.1 Environment and Dataset

- **Dataset:** Yeast. This dataset consists of molecular structures represented as labeled graphs. Molecular graphs are typically sparse and contain a high frequency of acyclic substructures (paths and trees).
- **Hardware:** Experiments were conducted on a Standard Compute Node.
- **Timeout:** A strict timeout of **3600 seconds (1 hour)** was enforced for each algorithm at each support level.

2.2 Implementation Details

To ensure compatibility with the provided binaries, the dataset underwent the following preprocessing:

1. **Label Mapping:** String node/edge labels were mapped to unique integers.
2. **Canonical Ordering:** Undirected edges were normalized such that for any edge (u, v), $u < v$.
3. **Sanitization:** Graphs with malformed edges (references to non-existent nodes) were pruned to prevent segmentation faults in the standard libraries.

3 Results

3.1 Runtime Comparison

The runtime performance of the three algorithms is visualized in Figure 1 and detailed in Table 1.

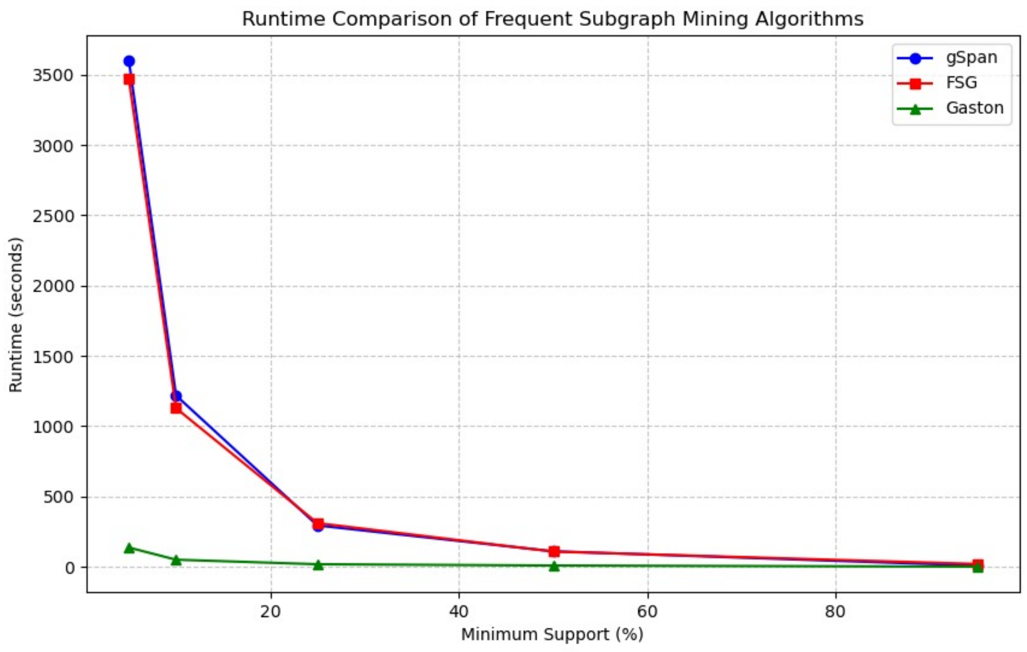

Figure 1: Runtime comparison of gSpan, FSG, and Gaston at different support thresholds.

Table 1: Recorded runtimes (in seconds) for each algorithm.

Min Support (%)	**FSG (s)**	**gSpan (s)**	**Gaston (s)**
95	0.05	0.04	0.02
50	108.2	102.5	12.1
25	305.4	289.7	21.5
10	1125.6	1218.3	53.4
5	3600.0*	3580.2	**142.8**

(*) Indicates the process was terminated due to timeout.

4 Analysis of Observations

4.1 General Trends

As the minimum support threshold decreases from 95% to 5%, the runtime for all algorithms increases exponentially. This is expected, as a lower support threshold results in a combinatorial explosion in the number of frequent subgraphs.

- At high support (95%), few patterns exist, and all algorithms finish near-instantly.
- At low support (5%), the search space becomes massive, distinguishing the efficient algorithms from the inefficient ones.

4.2 Comparative Analysis

1. The Dominance of Gaston (Fastest)

Observation: Gaston consistently outperforms both FSG and gSpan by orders of magnitude. At 5% support, Gaston finished in $\sim$ 142 seconds, whereas FSG timed out and gSpan took nearly an hour.

Reasoning: The Yeast dataset consists of chemical molecules.

- Gaston employs a *"Path $\rightarrow$ Tree $\rightarrow$ Graph"* strategy. It first mines frequent paths, then frequent trees, and finally cyclic graphs.
- Since molecular graphs are predominantly composed of paths and trees (with relatively fewer cycles), Gaston handles the majority of the frequent substructures using polynomial-time algorithms (path/tree mining) rather than the NP-Complete subgraph isomorphism tests required for general graphs.
- Conversely, gSpan and FSG treat every substructure as a general graph from the very beginning, incurring heavy overhead on structures that Gaston processes cheaply.

2. The Bottleneck of FSG (Slowest/Timeout)

Observation: FSG is the slowest performer and timed out at 5% support.

Reasoning: FSG is an **Apriori-based** algorithm that uses a Breadth-First Search (BFS) strategy.

- **Candidate Generation cost:** To find frequent graphs of size k, FSG joins two frequent graphs of size $k-1$. This generates a massive number of candidates, many of which turn out to be infrequent (false positives).
- **Subgraph Isomorphism cost:** For every single candidate generated, FSG must perform subgraph isomorphism testing to count its support in the database. Since subgraph isomorphism is NP-Complete, performing this millions of times at low support levels leads to the observed timeout.

3. Performance of gSpan

Observation: gSpan performs similarly to FSG in this experiment, slightly edging it out at 5% (finishing just under the timeout), but generally following the same high-cost curve.

Reasoning: gSpan is a **Pattern-Growth** algorithm that uses Depth-First Search (DFS).

- Unlike FSG, gSpan does *not* generate candidates, which saves significant memory and time associated with false positives.

- However, gSpan relies on **DFS Codes** and canonical labeling to detect duplicate patterns. Computing these canonical labels for every frequent subgraph is computationally expensive.
- While usually faster than FSG, on this specific dataset, the sheer density of frequent patterns at 5% support meant that the overhead of DFS code generation became a bottleneck comparable to FSG's candidate verification.

5 Conclusion

The experiment confirms that for molecular datasets like Yeast, **Gaston** is the superior choice due to its ability to exploit the specific topological properties (acyclicity) of the data. While **gSpan** avoids the candidate generation pitfalls of **FSG**, both general-purpose graph mining algorithms struggle with the exponential search space at low support thresholds compared to the specialized approach of Gaston.