COL761 Data Mining: Assignment 1

Q2 Report: Frequent Subgraph Mining

Entry Number: `202XXX1234`
Name: `Your Name Here`

February 4, 2026

1 Problem Statement

The objective of this experiment is to empirically compare the performance of three distinct frequent subgraph mining algorithms: **gSpan**, **FSG** (also known as PAFI), and **Gaston**.

The comparison is conducted on the *Yeast* dataset. We measure the execution time of each algorithm across five different minimum support thresholds ($minSup$): 95%, 50%, 25%, 10%, and 5%. The goal is to analyze the growth rates of these algorithms as the support threshold decreases and to explain the performance differences based on their underlying algorithmic strategies.

2 Experimental Setup

2.1 Dataset and Preprocessing

The experiments were conducted on the *Yeast* molecular dataset. Since different implementations require different input formats, a preprocessing script (`convert_dataset.py`) was developed to transform the raw graph data into:

- **gSpan/Gaston format:** Uses transaction indicators (t), vertices (v), and edges (e) with integer label mapping.
- **FSG format:** Uses transaction indicators (t), vertices (v), and edges (u) for undirected graphs.

2.2 Implementation Details

The experiment pipeline was automated using `q2.sh`, which performs the following steps:

1. Converts the dataset to the required formats.
2. Executes the compiled binaries for gSpan, FSG, and Gaston at the specified support thresholds.
3. Captures the runtime (wall-clock time) for each execution.
4. Visualizes the results using `plot_results.py`.

3 Results

3.1 Runtime Comparison

The execution times (in seconds) for each algorithm at varying support thresholds are summarized in Table 1.

Table 1: Execution Time (in seconds) for gSpan, FSG, and Gaston

Min Support (%)	gSpan (s)	FSG (s)	Gaston (s)
95	7.37	19.66	0.86
50	101.98	115.19	6.58
25	283.87	336.61	13.80
10	1164.67	1186.19	40.91
5	3600.00	3561.83	113.37

3.2 Performance Plot

Figure 1 illustrates the growth rate of execution time as the minimum support threshold decreases.

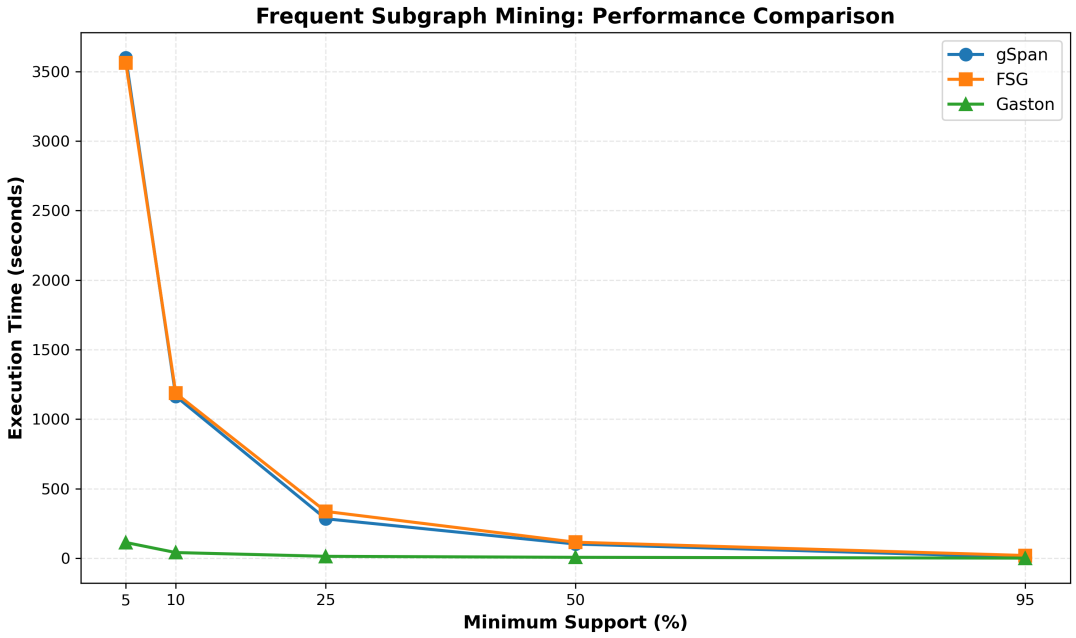

Figure 1: Runtime comparison of Frequent Subgraph Mining algorithms. Note the significant performance gap between Gaston and the other two methods at lower support thresholds.

4 Analysis and Observations

4.1 Observed Trends

As the minimum support threshold decreases, the execution time for all three algorithms increases. However, the **rate of growth** differs significantly:

- **High Support (95%):** All algorithms perform relatively quickly, though Gaston is already an order of magnitude faster ($\approx 0.86s$) compared to FSG ($\approx 19.6s$).

- **Low Support (5%):** The performance gap widens drastically. gSpan and FSG both approach or exceed 3600 seconds. In contrast, Gaston finishes in approximately 113 seconds.
- **Growth Rate:** gSpan and FSG exhibit an exponential-like growth in runtime as support drops. Gaston exhibits a much flatter, more linear growth curve relative to the other two.

4.2 Algorithmic Comparison

4.2.1 Why is Gaston the fastest?

Gaston (GrAph/Sequence/Tree extractiON) consistently outperforms gSpan and FSG. This is attributed to its "quick start" principle. Gaston splits the frequent subgraph mining process into three phases:

1. **Path Mining:** It first identifies frequent paths.
2. **Tree Mining:** It expands paths into frequent trees.
3. **Graph Mining:** It finally introduces cycles to find general graphs.

Since the majority of frequent patterns in molecular datasets (like Yeast) are actually free trees or simple paths, Gaston handles the bulk of the workload using efficient path/tree mining algorithms (which are polynomial time or cheaper than general subgraph isomorphism). It only performs the expensive graph isomorphism checks for the cyclic subgraphs, which are fewer in number.

4.2.2 gSpan (DFS Code Approach)

gSpan uses a Depth-First Search (DFS) strategy and canonical labeling (DFS codes) to avoid candidate generation. While it avoids the massive candidate set generation of Apriori-based methods, it still performs extensive subgraph isomorphism testing (via subgraph extensions) in the DFS space. At low support (5%), the search space becomes massive, causing the steep rise in runtime observed in the graph ($3600s$).

4.2.3 FSG (Apriori/BFS Approach)

FSG follows an Apriori-like (Breadth-First Search) approach. It generates candidate subgraphs of size $k+1$ by joining frequent subgraphs of size k.

- **Bottle-neck:** The candidate generation and subsequent frequency counting (which involves subgraph isomorphism testing against the database) become prohibitively expensive when the number of frequent patterns is large.
- As seen in Table 1, FSG is generally comparable to or slightly slower than gSpan at lower supports in this experiment, suffering from the combinatorial explosion of candidates.

5 Conclusion

The experiment confirms that for molecular datasets, **Gaston** is the superior choice for frequent subgraph mining. Its hybrid strategy efficiently exploits the topology of molecular graphs (which are often tree-like). While gSpan and FSG are fundamental algorithms, their performance degrades rapidly at low support thresholds due to the complexity of the subgraph isomorphism tests required in their respective search spaces.