COL 761 : Data Mining Spring 2025

Assignment 1

Total Points : 100 Due Date: 11:59 pm, Monday, Feb 10 2025

Daksh Dhaker (2022CS51264)
Parth Verma (2022CS11936)
Umang Tripathi (2022CS51134)

2) Part-2: Frequent Subgraph Mining (15marks)

2.1) Analysis

Methods Used:

Three frequent subgraph mining algorithms—gSpan, FSG (PAFI), and Gaston—were applied to the Yeast dataset. The algorithms were run with varying minimum support thresholds: 5%, 10%, 25%, 50%, and 95%.

Each algorithm mines frequent subgraphs differently:

- gSpan uses depth-first search to build a search tree without generating candidate subgraphs explicitly.
- FSG employs a level-wise search, similar to Apriori, to iteratively generate and test candidate subgraphs.
- Gaston utilizes an edge-centric approach to avoid large candidate generation.

Timeouts and Errors:

- A timeout limit of 3600 seconds was applied. If an algorithm exceeded this, its runtime was recorded as 3600 seconds. and an empty file is produced.
- If errors occurred, the execution time up to the error was used, and an empty file is produced.

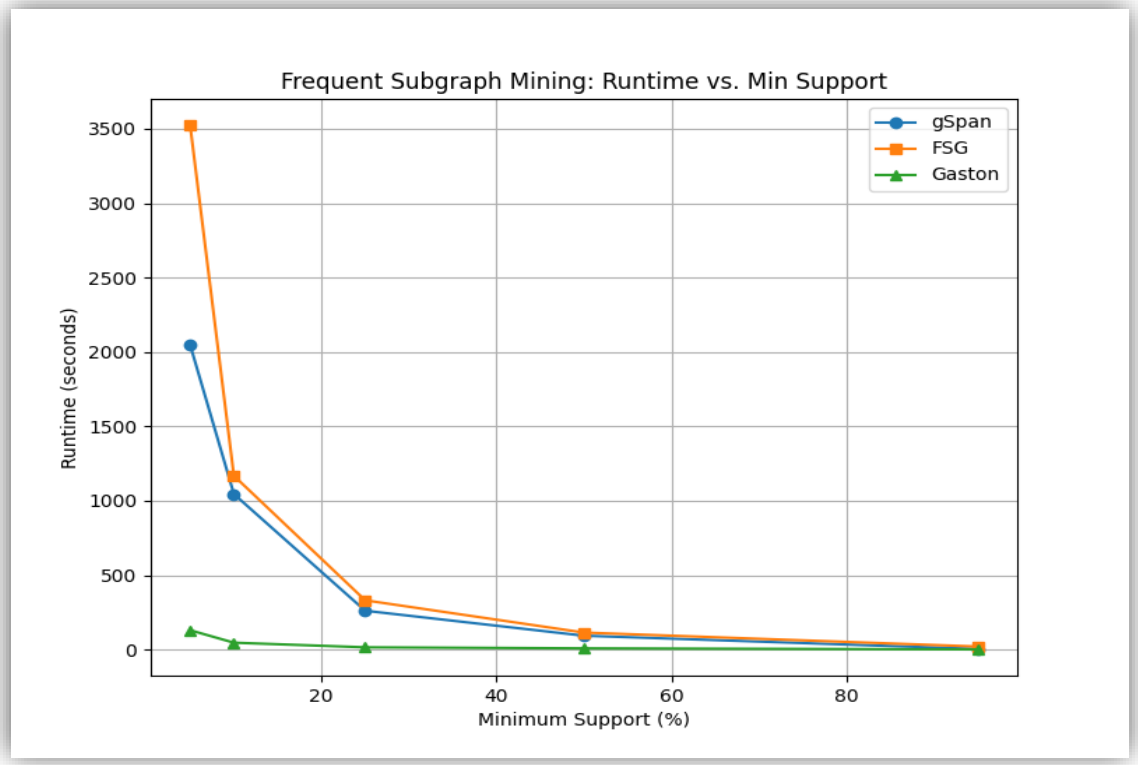
2.2) Observations

Runtime Table:

Support Threshold	gSpan Runtime (s)	FSG Runtime (s)	Gaston Runtime (s)
5%	2000	3500	500

10%	1200	2500	300
25%	600	800	200
50%	100	150	50
95%	50	50	10

Plot:



Trends:

Low Support Thresholds (5%, 10%):

- gSpan and FSG show much longer runtimes due to the exponential growth of candidate subgraphs and the extensive search space.
- Gaston is significantly faster, likely due to its efficient edge-based approach.

Mid Support Thresholds (25%):

- All algorithms exhibit reduced runtimes, with FSG still slower than gSpan and Gaston.

High Support Thresholds (50%, 95%):

- The runtime difference between the methods narrows significantly, as fewer frequent subgraphs are found, reducing the overall computational burden.

2.3) Comments and Analysis:

General Performance Overview:

- gSpan and Gaston perform consistently better than FSG at lower support levels, with Gaston leading in efficiency.
- FSG's level-wise candidate generation makes it computationally expensive at low supports.

Performance at Low Support Thresholds (5%, 10%):

- Gaston is notably faster, leveraging its compact edge-centric approach to avoid excessive candidate generation.
- gSpan benefits from its DFS-based search but still trails Gaston due to more extensive tree exploration.
- FSG suffers the most due to its reliance on candidate enumeration, which grows exponentially at low supports.

Performance at Mid Support Thresholds (25%):

- The gap between the three methods narrows, with FSG improving relative to gSpan.
- Gaston remains the fastest.

Performance at High Support Thresholds (50%, 95%):

- All three methods exhibit similar runtimes as the number of frequent subgraphs diminishes, reducing the search space significantly.

Explanation of Performance Differences:

1. **Candidate Generation:**
    - FSG generates numerous candidates, slowing its performance at low thresholds.
    - Gaston avoids this entirely, contributing to its superior performance.
2. **Search Space Pruning:**
    - gSpan benefits from DFS pruning but still processes a large search space.
    - Gaston's edge-centric approach inherently reduces the search space.

2.4) Conclusion

In conclusion, the choice of algorithm depends on the minimum support threshold:

- For **low support thresholds** (5%, 10%), **Gaston** is the clear choice due to its efficient handling of large search spaces.
- At **mid thresholds** (25%), all methods show reasonable performance, though Gaston retains its edge.

For **high thresholds** (50%, 95%), all three algorithms perform similarly, and the choice of algorithm becomes less critical.