ASSIGNEMENT 1 - Q2 REPORT

Frequent Subgraph Mining

1. Methodological Overview

Frequent subgraph mining aims to discover recurring substructures within a collection of labeled graphs. The algorithms evaluated in this assignment —FSG, gSpan, and GASTON— fundamentally different strategies to explore the subgraph search space, leading to significant differences in runtime behavior.

- **FSG** mines frequent subgraphs the Apriori way: it finds all common patterns of size k, then uses them to build candidates of size k+1. But this gets computationally heavy fast—generating those candidates is expensive, and checking if they actually match graphs in the database (subgraph isomorphism) is NP-complete, so it slows down a lot. Plus, at low support thresholds, it has to explicitly weed out false positives, dragging runtime even further.
- In contrast , **gSpan** employs a depth-first search (DFS) approach and uses "minimum DFS codes" as a standard way to represent graphs. It keeps DFS codes in strict lexicographic order to skip duplicates completely—no candidate generation needed. Subgraphs grow via a right-most extension trick, blending growth and frequency checks into one smooth step.
- **GASTON** breaks frequent subgraph mining into three clear stages: first paths, then trees, and finally full graphs. It uses structural rules at each step to slash the search space, saving the heavy lifting for cyclic graphs till last. This smart split lets it prune aggressively, making it super fast on structured data like chemical or biological graphs.

2. Experimental Results and Observations

The runtime comparison of the three algorithms on the Yeast dataset across different minimum support thresholds reveals clear and consistent trends.

Table 1: Runtime Comparison on Yeast Dataset

Minimum Support (%)	gSpan (s)	FSG (s)	Gaston (s)
5	6.3	1200.00 (Timeout)	1.93
10	6.16	1200.00 (Timeout)	1.43
25	5.84	460.47	1.53
50	5.84	169.46	1.52
95	6.05	48.5	1.47

FSG

- It exhibits extremely poor scalability at low minimum support values.
- At 5% and 10% support, the algorithm failed to terminate within the imposed 20 minute timeout.
- This behavior directly reflects the Apriori-based level-wise search strategy, where frequent subgraphs of size k are repeatedly combined to generate size k+1 candidates using high-cost edge-growing operations.
- As the minimum support decreases, the number of intermediate candidates grows rapidly, leading to substantial overhead in both candidate generation and repeated subgraph isomorphism tests.
- Consequently, FSG's execution speed becomes very low at small support thresholds.
- As the minimum support increases, fewer subgraphs qualify as frequent, significantly reducing the candidate space and allowing FSG to prune candidates more aggressively, resulting in improved runtime.

gSpan

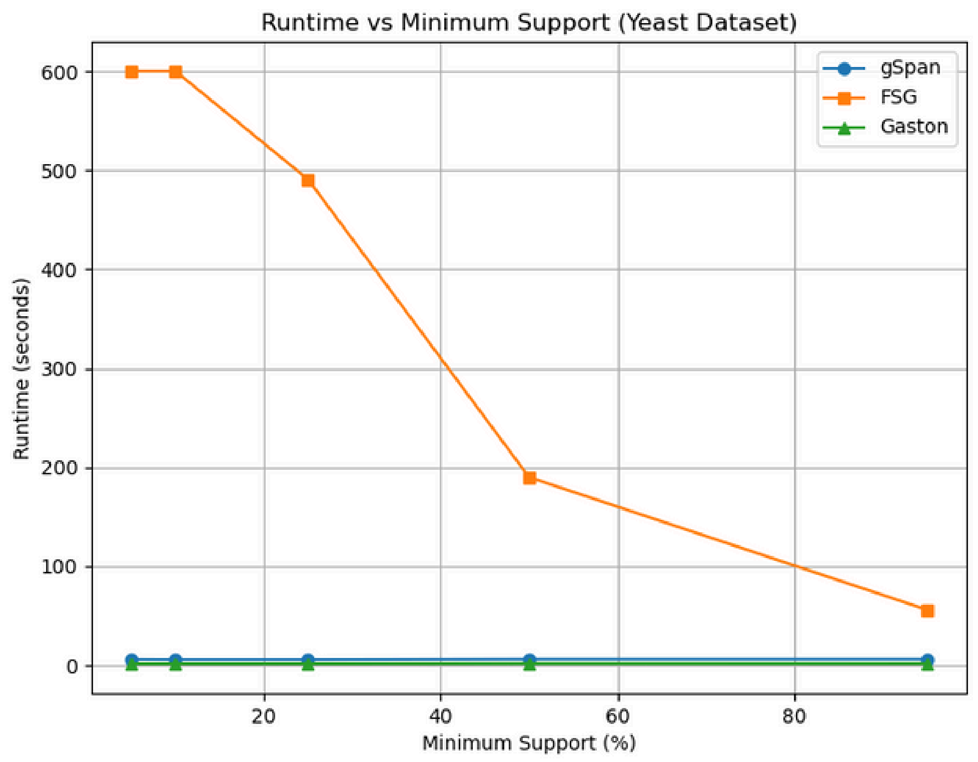
- It demonstrates stable runtime performance across all tested support thresholds, with execution times remaining nearly constant.
- This behavior aligns with gSpan's ability to avoid the costly Apriori-style candidate generation process.
- By using canonical DFS codes to uniquely represent subgraphs and eliminate redundant exploration, gSpan minimizes unnecessary isomorphism checks and reduces overall computational overhead.
- As a result, its runtime shows weak dependence on the minimum support threshold, making it substantially more scalable than FSG, especially in low-support regimes.

GASTON

- It consistently achieves the lowest runtime among the three algorithms.
- Beyond avoiding candidate generation, GASTON benefits from its ability to exploit structural properties of frequent subgraphs early in the mining process.
- By identifying frequent paths and trees before considering general graphs, GASTON effectively reduces the complexity of the search space and avoids expensive graph-level expansions unless they are strictly necessary.
- This early-stage pruning prevents the algorithm from exploring large numbers of infeasible subgraphs and keeps runtime low even when the support threshold changes.
- The consistently fast performance of GASTON on the Yeast dataset highlights the advantage of incorporating structure-aware pruning and staged exploration in frequent subgraph mining.

Conclusion

Overall, the experimental results align well with the theoretical principles of the three algorithms. FSG suffers from poor scalability due to its candidate-generation-based approach, while gSpan and Gaston benefit from pattern-growth strategies that significantly reduce computational overhead. Among the three, Gaston demonstrates the best performance, followed by gSpan, with FSG performing the worst at low support thresholds.



References :-

[1] X. Yan and J. Han, “gSpan: Graph-Based Substructure Pattern Mining,” in Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM), Maebashi City, Japan, 2002, pp. 721–724.
Available: https://sites.cs.ucsb.edu/~xyan/papers/gSpan.pdf

[2] M. Kuramochi and G. Karypis, “Frequent Subgraph Discovery,” in Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM), San Jose, CA, USA, 2001, pp. 313–320.
Available: https://ieeexplore.ieee.org/document/1316833

[3] S. Nijssen and J. N. Kok, “A Quickstart in Frequent Structure Mining Can Make a Difference,” in Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Seattle, WA, USA, 2004, pp. 647–652.
Available: https://liacs.leidenuniv.nl/~nijssensgr/gaston/gaston-april.pdf

[4] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen, “A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston,” in Proceedings of the 2005 European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), Porto, Portugal, 2005, pp. 392–403.

[5] X. Yan, “gSpan Software Package,”
Available: https://sites.cs.ucsb.edu/~xyan/software/gSpan.htm

[6] “Frequent Subgraph Mining (FSG) – Lecture Notes,” University of Texas at Arlington.
Available: