COL761/COL7361/AIL7026 Data Mining

Assignment 1: Frequent Itemset Mining

Team T123

Kislaya Shrestha 2025SIY7573
Palaash Goel 2025AIZ8825
Anoop Parashar 2025MCS2118

February 2, 2026

Task 2: Comparision of gSpan, fSG, and Gaston

This tasks required us to run gSpan (Yan and Han, 2002), fSG (Kuramochi and Karypis, 2004), and Gaston (Nijssen and Kok, 2005) on the given Yeast dataset to mine frequent subgraphs from it, analysing the run-time for each algorithm and the reasons behind the same.

Experimental Setup

We utilized binaries provided by the authors of the respective methods to mine frequent subgraphs from the Yeast dataset and utilized simple Bash programming to record the execution time for each algorithm. This data was then passed to a Python script for plotting (`plot.py`). Note that the dataset had to be formatted different for each method. This was achieved using another python script (`q2.py`). We liberally used the Python [1] and Bash [2] documentation and tutorials [3] for the same.

Runtime Results

Table 1 presents the observed execution times.

Table 1: Execution Time (s) for fSG, gSpan, and Gaston

Support (%)	fSG	gSpan	Gaston
5	3577	4301	104
10	1182	1172	38
25	335	286	12
50	115	102	6
95	19	7	0.80

[1]`https://docs.python.org/3/`
[2]`https://devdocs.io/bash/`
[3]`https://www.w3schools.com/bash/`

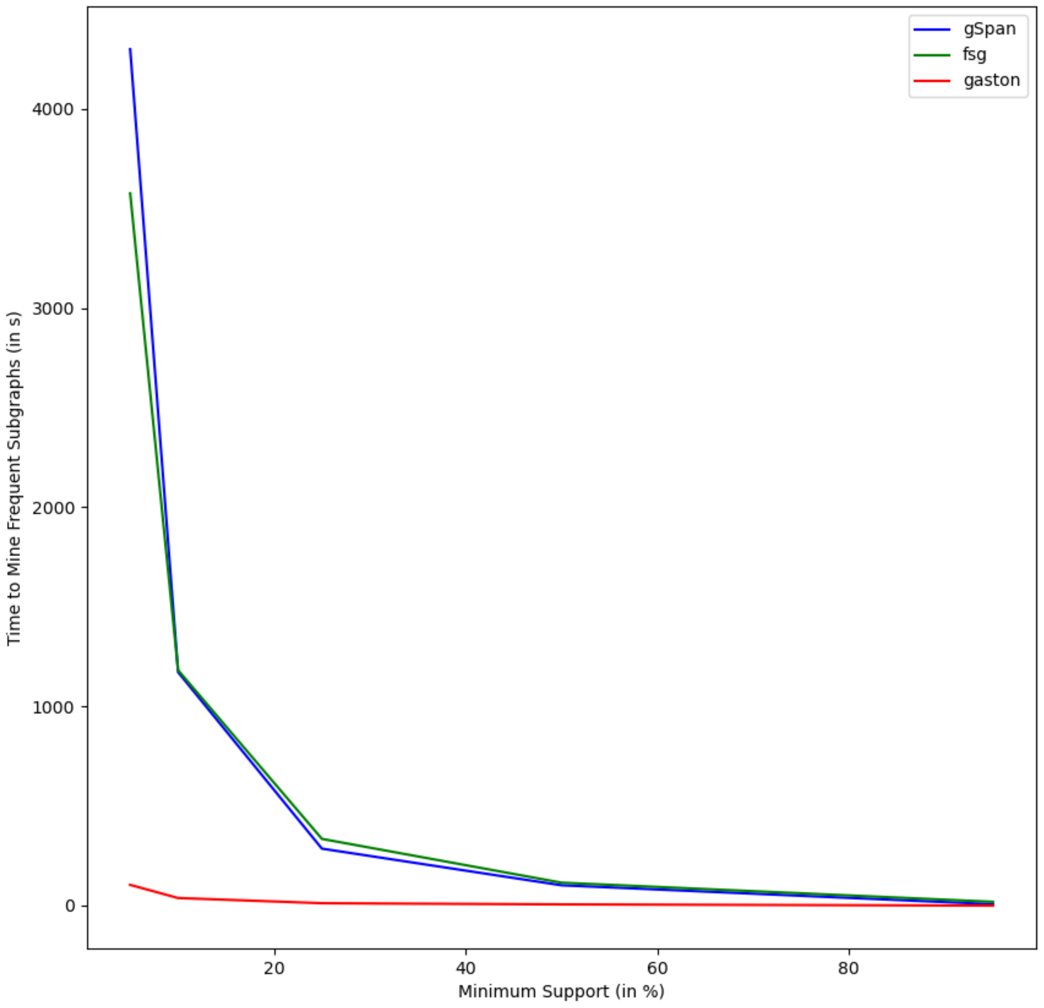

Figure 1: A comparision of the runtimes of fSG, gSpan, and Gaston at varying support thresholds (5%, 10%, 25%, 50% and 95%).

Experimental Results

We provide the quantitative experimental results in Table 1 and Figure 1.

Low Support - 5%, 10%

At the low support threshold of 5%, we observe that fSG is able to outperform gSpan while the reverse is true at a threshold of 10%. However in both cases, Gaston outperforms them by mining all frequent subgraphs in a fraction of their respective times.

Intermediate Support - 25%, 50%

Here, we observe that gSpan outperforms over fSG on both thresholds while Gaston is the superior method among the three.

High Support

We observe a similar trend in the case of a high support threshold. Due to how high the support threshold is, most subgraphs (and their derivatives) fail to meet this threshold, causing them to get pruned at earlier stages of mining. This leads to conservatively-sized candidate sets as the algorithm proceeds, thereby leading to significant speedups. Since all three methods benefit from this, they maintain the relative ordering of their execution times with Gaston being the fastest, followed by gSpan and fSG, respectively.

Analysis

We analyze the algorithms and observe strong correlations between their mining approach and their run-times. We provide a thorough analysis for each method below.

fSG. This is a breadth-first Apriori-like approach for mining subgraphs which generates candidates of size $k + 1$ by joining candidates of size k. While this approach guarantees correctness, it suffers from a significant overhead, *i.e.*, it is prone to generating a large number of candidates, including "false" candidates that require expensive computations such as exponential-time isomorphism tests to identify and discard (*e.g.*, duplicate candidates). Additionally, it needs to perform these subgraph isomorphism tests between each pair of candidate and database graphs in order to compute the support for the former. While it utilizes heuristics and techniques such as canonical labeling to lower overall cost, its breadth-first nature and join-based generation strategy invariably leads to large candidate sets and unavoidably significant computational costs.

gSpan. gSpan makes a significant improvement over fSG by opting for a depth-first approach instead of a breadth-first approach. In other words, it explores all subgraphs patterns exhaustively, thereby eliminating redundant paths early on. Additionally, unlike fSG, it does not rely on an expensive candidate generation stage and instead directly grows frequent patterns of size $k + 1$ from patterns of size k. This enables significant efficiency boosts as compared to fSG since it completely bypasses the expensive tests for identifying duplicate or wrongly-sized candidates.

We note that despite its performance benefits over fSG, it fails to beat fSG at a support threshold of 5% (Table 1 and Figure 1). We posit that at such a low support threshold, the space of subgraphs meeting this threshold is notably large since it is extremely likely for any arbitrary subgraph to be present in at least 5% of the graphs in the database. This diminishes the relative advantages attached to gSpan's depth-first approach, since its frequent pattern space reaches close to the size of fSG's breadth-first candidate space in the presence of such a small set of infrequent subgraphs. Furthermore, gSpan incurs an overhead for the various techniques it utilizes to maximize efficiency (*e.g.*, DFS codes). While this overhead this usually insignificant due to gSpan's efficient processing of frequent patterns, its effect becomes much more pronounced in the presence of abnormally large pattern sets (as in the case of a support threshold of 5%), leading to further degradation in performance. Despite this, we note gSpan's undeniable superiority over fSG on the remaining support thresholds, attributing its worse performance to an abnormally low support threshold.

Gaston. We observe that Gaston consistently improves over its competitors by impressive margins. This is due to its level-wise approach to mining frequent subgraphs. Specifically, Gaston operates on a simple yet effective observation, *i.e.*, quite often, frequent subgraphs are not highly complex, but rather consist of rudimentary structures that repeat in numerous graphs. By exploiting this observation, Gaston is able to mine frequent subgraphs at different levels of complexity. It first mines simple frequent paths in the database, followed by complex trees and lastly, cyclic graphs. Such a hierarchical search space grants significant speedups to Gaston, as evident by corresponding empirical evidence. Additionally, it utilizes various techniques such as normalizing graphs using Nauty and hashing them into hash tables for fast look-up, offering further speedups.

Conclusion

The experimental results confirm that Gaston performs the best out of the three methods at all support thresholds, followed by gSpan and fSG. While gSpan might suffer from significant overhead-related costs at extremely low support thresholds, this does not hinder its practical applicability over fSG since such low support thresholds are rarely explored in practical settings. From a broader perspective, we also observe that depth-first approaches (pattern-growth approaches) outperform breadth-first ones by pruning their search space early and avoiding the generation of redundant and "false" subgraphs as candidates.

References

M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004. doi: 10.1109/TKDE.2004.33.

S. Nijssen and J. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127:77–87, 03 2005. doi: 10.1016/j.entcs.2004.12.039.

X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, page 721, USA, 2002. IEEE Computer Society. ISBN 0769517544.