Analysis of Frequent Subgraph Mining Techniques

Yearning For The Mines

February 2025

1 Introduction

Frequent subgraph mining is an essential task in graph analytics, widely used in bioinformatics, cheminformatics, and social network analysis. In this study, we compare the runtime performance of three algorithms: gSpan, FSG, and Gaston, across different minimum support thresholds.

2 Methods

2.1 gSpan

gSpan (graph-based Substructure pattern mining) employs a depth-first search (DFS) approach to efficiently enumerate frequent subgraphs without candidate generation.

2.2 FSG

FSG (Frequent Subgraph Mining) is based on an Apriori-like approach that generates candidate subgraphs and filters infrequent ones using the downward closure property.

2.3 Gaston

Gaston integrates different search strategies, including path-based, tree-based, and graph-based searches, optimizing subgraph enumeration based on their structural characteristics.

3 Results

Figure 1 shows the runtime of the three algorithms at various minimum support thresholds.

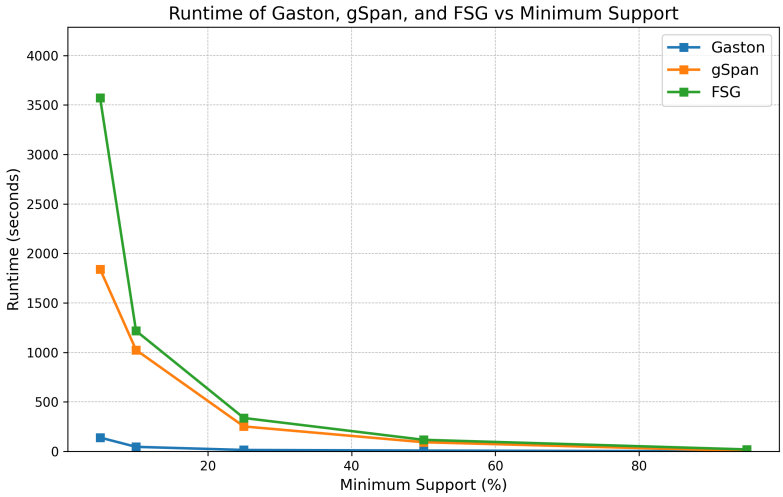

Figure 1: Runtime of Gaston, gSpan, and FSG vs. Minimum Support

4 Discussion

The plot illustrates the following trends:

- FSG exhibits a sharp increase in runtime at low support values, indicating high computational costs for candidate generation and pruning.
- gSpan shows a similar decreasing trend but remains faster than FSG, likely due to its DFS-based approach reducing candidate explosion.
- Gaston consistently outperforms both methods, maintaining the lowest runtime across different support values. Its hybrid strategy effectively optimizes search space traversal.

4.1 Why Gaston is Faster than gSpan

Gaston outperforms gSpan due to its hybrid search strategy, which optimizes the way frequent subgraphs are discovered:

- Multiple Search Strategies: Gaston employs three different search paradigms: path-based search, tree-based search, and graph-based search. This adaptive selection helps Gaston prune the search space efficiently, reducing unnecessary computations.
- Efficient Subgraph Enumeration: Gaston prioritizes structures that are more likely to be frequent, reducing redundant subgraph checks.

- Lower Candidate Explosion: Unlike gSpan, which relies entirely on depth-first search (DFS), Gaston intelligently switches strategies based on the subgraph type, preventing candidate explosion in highly connected graphs.
- Optimized Data Structures: Gaston uses a compact representation of graphs, leading to faster access and processing compared to gSpan's DFS-based tree expansion.

4.2 Why gSpan is Faster than FSG

gSpan outperforms FSG primarily because it avoids the inefficiencies of the Apriori-like candidate generation used in FSG:

- No Candidate Generation: FSG, like the Apriori algorithm in frequent itemset mining, generates and tests candidate subgraphs, leading to exponential growth in the number of candidate patterns. gSpan, on the other hand, directly extends patterns using DFS-based pattern growth, reducing redundant operations.
- Lexicographic Order and DFS Pruning: gSpan efficiently prunes the search space by using lexicographic order and DFS extensions, reducing the number of unnecessary pattern checks.
- Less Memory Overhead: FSG stores and processes many intermediate candidate graphs, leading to high memory usage and slow performance. gSpan, being DFS-based, maintains a more compact search tree, reducing memory overhead.
- Better Scalability: Because FSG relies on joining subgraphs in an iterative manner, it becomes much slower as graph sizes increase. gSpan, using DFS-based search, avoids unnecessary recomputations and is inherently more scalable.

5 Conclusion

Our analysis suggests that FSG is highly inefficient at low support values, whereas Gaston provides the most scalable performance. gSpan serves as a middle ground but does not outperform Gaston in any setting. The choice of algorithm should consider dataset characteristics and minimum support thresholds.