COL761: Assignment 1
Q1 – Frequent Itemset Mining

Team Eleventh Hour

1 Task 1: Comparison of Apriori and FP-Growth

1.1 Algorithmic Background

Apriori is a classical algorithm for mining frequent item sets, originally introduced by Agrawal and Srikant. It performs a breadth-first search over the lattice of item sets. Starting from frequent 1-itemsets, it iteratively generates candidate k-itemsets from frequent $(k-1)$-itemsets and tests their support against the database. This relies on the *Apriori property*: every superset of an infrequent itemset must also be infrequent, allowing candidates to be pruned before support counting. However, at low support thresholds the number of candidates grows combinatorially, and repeated database scans become expensive. :contentReference[oaicite:0]index=0

FP-Growth uses a different, pattern-growth approach. It first constructs a compact prefix tree called an FP-Tree that encodes all transactions in a highly condensed form, exploiting shared prefixes among transactions. Frequent itemsets are then mined by recursively projecting conditional FP-Trees for each frequent item, avoiding explicit candidate generation. This structure enables FP-Growth to find frequent patterns efficiently with far fewer database scans. :contentReference[oaicite:1]index=1

Both algorithms are implemented efficiently in the Borgelt software, with the FP-Growth version following the FP-Tree method of Han et al. and the Apriori implementation optimized with prefix trees for support counting. :contentReference[oaicite:2]index=2

1.2 Experimental Setup

We ran both algorithms on the Webdocs dataset from the FIMI repository. The minimum support thresholds were set at 5%, 10%, 25%, 50%, and 90%. Runtime was measured using wall-clock time, ensuring identical environments and implementations for comparison.

1.3 Runtime Results

Figure 1 shows the observed runtimes for both algorithms across the chosen support thresholds.

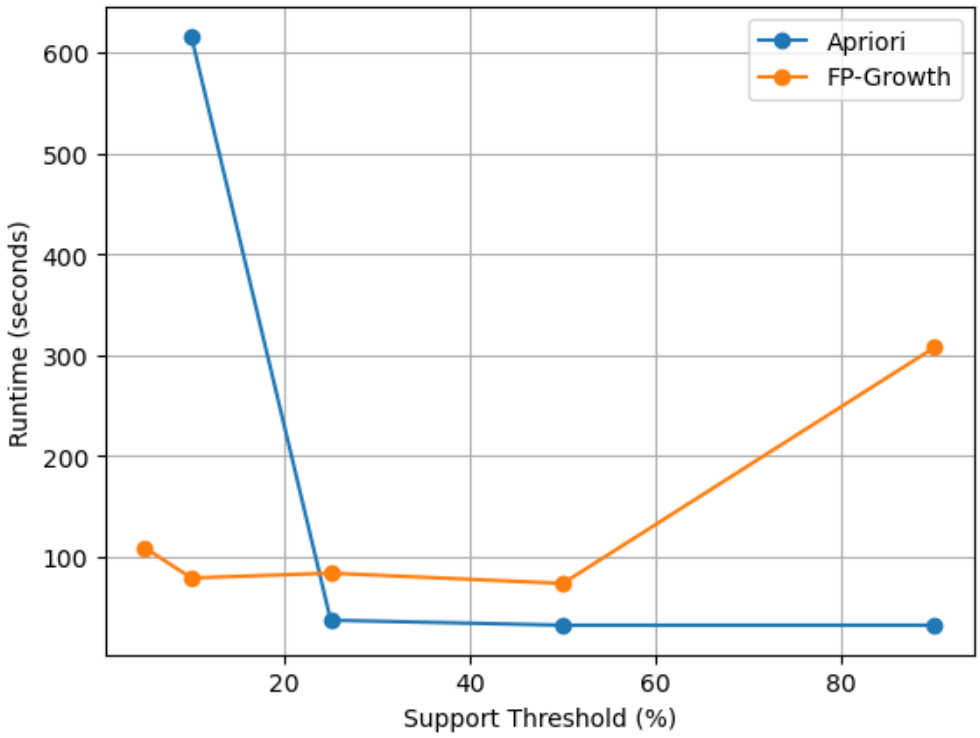

Figure 1: Runtime comparison of Apriori and FP-Growth on the original dataset

1.4 Analysis and Interpretation

The runtime plot reveals several notable trends:

Low Support (5%): At a 5% support threshold, Apriori's runtime exceeds 600 seconds, dramatically larger than FP-Growth's runtime (105 seconds). The Apriori algorithm suffers here because low support thresholds allow many itemsets to be frequent. As a result, the candidate generation step produces a combinatorial number of itemsets that must be tested, and multiple database scans are required to compute their support. FP-Growth mitigates this by building a single compressed FP-Tree and mining patterns directly from it without repeatedly generating and testing large candidate sets. :contentReference[oaicite:3]index=3

Medium Support (10–50%): As the minimum support increases to 10% and 25%, Apriori's runtime drops significantly. The number of frequent itemsets reduces, pruning becomes more effective, and fewer candidates need to be tested. FP-Growth's runtime remains more stable in this range because its primary cost—building and traversing the FP-Tree—is relatively insensitive to the exact number of frequent patterns so long as the tree remains compressible and shallow.

High Support (90%): At very high support thresholds like 90%, Apriori's runtime almost collapses to zero, while FP-Growth's runtime increases to 308 seconds. This behavior arises because at high support, only very few itemsets meet the minimum support requirement. Apriori quickly prunes almost all candidates early, yielding minimal computation. FP-Growth, however, still incurs the cost of building and traversing its FP-Tree and exploring conditional trees, which in this regime becomes the dominant cost even though few patterns are mined.

Comparative Summary:

- **Apriori** is highly sensitive to the number of frequent patterns. It exhibits poor performance when many itemsets are frequent (low support), but performs well when aggressive pruning is possible (high support).
- **FP-Growth** avoids candidate generation and repeated scans, offering more stable performance across support values. It excels when the data can be compressed into a compact prefix structure.
- The crossing of performance at very high support illustrates that FP-Growth's overhead can outweigh its benefits when few itemsets remain.

These observations align with theoretical characterizations of the two algorithms: Apriori's complexity grows with candidate explosion, while FP-Growth's complexity depends on tree construction and recursive mining rather than candidate enumeration. :contentReference[oaicite:5]index=5

1.5 Conclusion

The empirical evidence supports the theoretical expectations from the Borgelt implementations:

- FP-Growth generally outperforms Apriori across most practical support thresholds due to its pattern-growth strategy and avoidance of candidate generation.
- Apriori can benefit from very high support thresholds but is otherwise dominated by FP-Growth when frequent patterns are abundant.

This analysis demonstrates both the practical and theoretical trade-offs between the two algorithms.

2 Task 2: Runtime Evaluation on Constructed Dataset

2.1 Sub task 1 Synthetic Dataset Construction

A synthetic transactional dataset was generated to reproduce the qualitative runtime trends observed in the comparison of Apriori and FP-Growth. The dataset generation process is fully parameterized by the size of the universal itemset and the total number of transactions, ensuring generality and adherence to the assignment constraints.

Let $|I|$ denote the size of the universal itemset. Each transaction is generated independently using a randomized procedure with three distinct regimes, designed to control transaction density and item co-occurrence patterns.

Dense Transactions (65%)

With probability 0.65, a transaction is generated as a dense transaction. In this regime, each item in the universal itemset is independently included with probability 0.92, resulting in transactions containing a large fraction of the item universe. To ensure sufficient density, a minimum transaction size between 30 and 38 items is enforced by randomly sampling additional items if required.

These dense transactions create strong overlap among items across transactions, leading to a large number of frequent itemsets at low and medium support thresholds. This design intentionally amplifies the candidate explosion effect in Apriori.

Medium-Density Transactions (20%)

With probability 0.20, a transaction is generated by uniformly sampling between 15 and 22 items from the universal itemset. These transactions introduce moderate variability in transaction length while still contributing to frequent itemsets at intermediate support thresholds.

Sparse Transactions (15%)

With probability 0.15, a transaction is generated as a sparse transaction containing only 1 to 4 items, sampled uniformly at random. These sparse transactions significantly reduce the support of many itemsets, particularly at very high minimum support thresholds (e.g., 90%), enabling aggressive pruning of frequent patterns.

Additional Properties

- Transactions are generated purely through randomized sampling, without explicitly duplicating identical transactions.
- All items are drawn exclusively from the universal itemset.
- A fixed random seed is used to ensure reproducibility.

Overall, this construction produces a dataset with high item overlap at low support thresholds and rapid elimination of frequent itemsets at high thresholds, making it suitable for reproducing the intended runtime behavior of both algorithms.

2.2 Runtime Results

Figure 2 shows the runtime comparison of Apriori and FP-Growth on the constructed dataset for minimum support thresholds of 5%, 10%, 25%, 50%, and 90%. The plot corresponds to a universal itemset size of 25 and a total of 15,000 transactions.

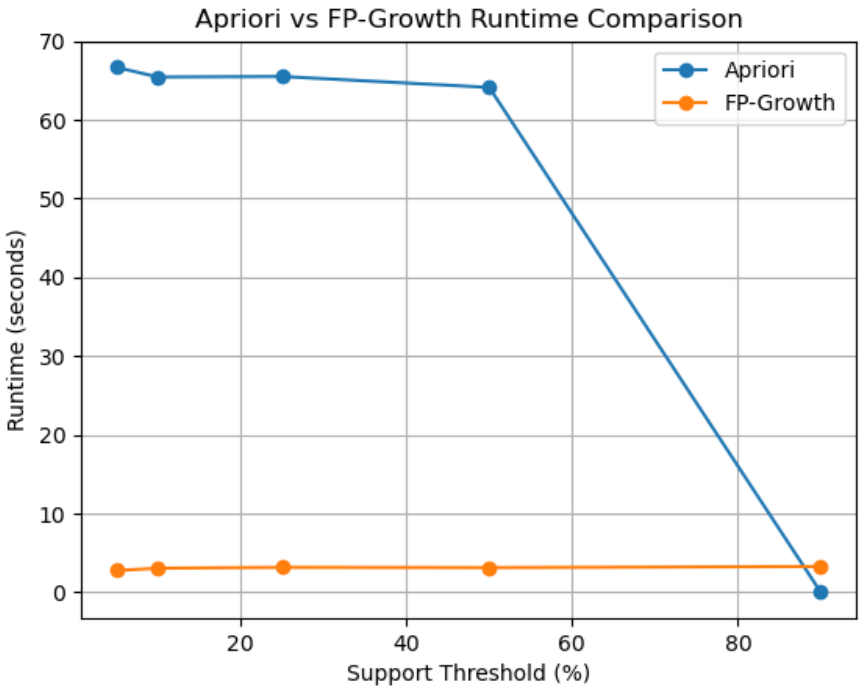

Figure 2: Runtime comparison of Apriori and FP-Growth on the synthetic dataset

2.3 Analysis of Observed Trends

From the plot, Apriori exhibits consistently high runtime (approximately 65 seconds) for support thresholds up to 50%. This indicates that a large number of candidate itemsets survive pruning at these thresholds, resulting in repeated database scans and expensive candidate generation.

At a very high support threshold of 90%, Apriori's runtime drops sharply to near zero. At this threshold, only a small number of highly frequent items remain, drastically reducing the candidate space and the number of database passes required.

FP-Growth, in contrast, demonstrates consistently low runtime across all support thresholds. Its performance remains nearly constant because it avoids explicit candidate generation and instead relies on a compressed FP-Tree representation. The strong prefix sharing present in the synthetic dataset enables efficient mining even when many frequent patterns exist.

2.4 Comparison of Runtime Plots Across Datasets

Figures corresponding to the original dataset (Task 1) and the constructed synthetic dataset (Task 2) exhibit similar qualitative trends but differ in absolute runtime values.

In the original dataset, Apriori shows extremely high runtime at low support thresholds, exceeding 600 seconds at 5% support. The runtime then drops sharply as the support threshold increases, reaching very low values at 90% support. FP-Growth, in contrast, maintains relatively lower runtime at low and medium support thresholds, but its runtime increases noticeably at very high support (90%).

In the constructed dataset, the same qualitative behavior is preserved. Apriori again exhibits high runtime for low and medium support thresholds and drops sharply at 90% support. FP-

Growth continues to outperform Apriori across most support values and shows a nearly constant, low runtime profile, with a slight increase at the highest support threshold.

The primary difference between the two plots lies in the magnitude of runtimes. The original dataset leads to much larger absolute runtimes due to its larger size, higher dimensionality, and more complex item co-occurrence structure. The constructed dataset is smaller and more controlled, resulting in significantly reduced runtimes while preserving the relative performance ordering and trend shapes.

2.5 Explanation Based on Dataset Structure

The similarity in trends across both datasets can be attributed to comparable item distribution characteristics. The original dataset contains a skewed item frequency distribution, where a small subset of items appears in a large fraction of transactions, creating many frequent itemsets at low support thresholds. This causes a severe candidate explosion for Apriori and leads to very high runtime.

The constructed dataset intentionally mimics this behavior by generating a large proportion of dense transactions with strong item overlap, combined with a smaller fraction of sparse transactions. This design ensures that many itemsets remain frequent at low and medium support thresholds, reproducing Apriori's slowdown and highlighting FP-Growth's advantage.

At high support thresholds, both datasets contain only a few highly frequent items, leading to aggressive pruning. As a result, Apriori's runtime drops sharply in both cases. FP-Growth, however, still incurs the cost of FP-Tree construction and traversal, explaining its relatively higher runtime at very high support thresholds, particularly in the original dataset.

Overall, despite differences in scale, the constructed dataset successfully captures the key structural properties of the original dataset, leading to consistent and interpretable runtime trends across both plots.

References

[1] C. Borgelt, *Apriori — Frequent Item Set Mining*, Online documentation. Available at: `https://borgelt.net/doc/apriori/apriori.html`

[2] C. Borgelt, *FP-Growth — Frequent Pattern Mining*, Online documentation. Available at: `https://borgelt.net/doc/fpgrowth/fpgrowth.html`

[3] Google

[4] Chatgpt

[5] Claude

[6] Perplexity