Task 1: Performance Analysis of Apriori and FP-Growth on the Original Dataset

Approach and Implementation

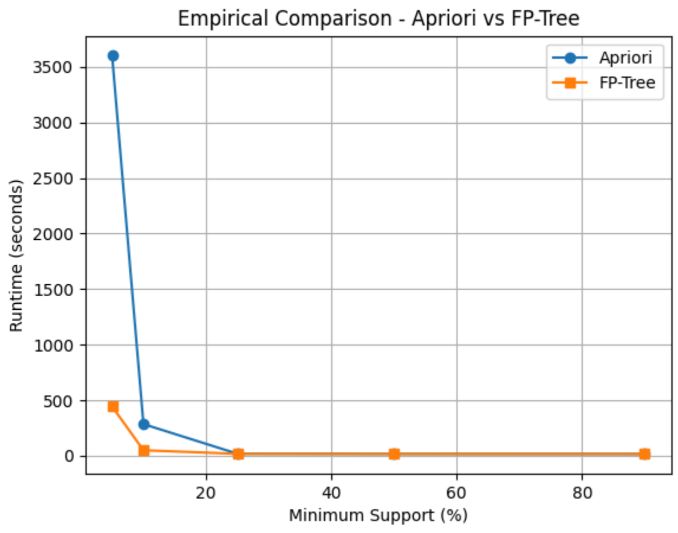
In Task 1, we used the original transaction dataset that came with the assignment to assess the runtime behaviour of the Apriori and FP-Growth algorithms. Examining how these strategies scale as the minimum support criterion changes was the goal.

The minimal support parameter was adjusted during the studies, while all other parameters remained the same. In particular, we tested the algorithms at 5%, 10%, 25%, 50%, and 90% minimum support thresholds (represented by the usage of the “-s#” flag). The wall-clock runtime that the mining program indicated for each run was noted. A timeout of 3600 seconds (1 hour) was enforced to comply with the submission guidelines. The runtimes DID NOT include the time needed to generate the plot.

To ensure a fair comparison, Apriori and FP-Growth both used the same dataset and support thresholds. Python 3.10 and matplotlib were used to plot the resultant runtimes.

Observed Results and Analysis

The resulting plot shows a **clear and dramatic performance difference** between Apriori and FP-Growth, particularly at low support values.



Low Support (5%)

Apriori shows a severe runtime explosion with a minimum support of 5%, taking several thousand seconds to finish. Because Apriori relies on explicit candidate creation, this behaviour is expected. There is a combinatorial explosion in the number of candidate (k)-itemsets at low support thresholds because many itemsets satisfy the frequency restriction. The transaction database must be repeatedly scanned at each stage of candidate generation, greatly increasing runtime.

FP-Growth, on the other hand, performs noticeably better at the same support level. By condensing the dataset into an FP-tree and repeatedly mining conditional trees, FP-Growth prevents candidate formation. It can handle dense, frequent patterns significantly more effectively than Apriori thanks to this design.

Moderate Support (10%–25%)

Both techniques exhibit a sharp drop in runtime as the minimal support threshold rises. Many itemsets are pruned early, which shrinks the conditional FP-trees constructed by FP-Growth and lowers the number of candidates produced by Apriori. The difference between the two algorithms becomes much less in this regime, even if Apriori is still slower than FP-Growth.

High Support (50%–90%)

At high support thresholds, the runtimes of both algorithms converge to very small values. Very few (or no) frequent itemsets of size greater than one survive, causing both algorithms to terminate quickly.

Task 2: Analysis on the Constructed Dataset and Comparison with Task 1

Dataset Construction Approach

In Task 2, we constructed a synthetic transaction dataset to analyze how controlled item distributions influence the runtime behavior of Apriori and FP-Growth.

Inputs:

- the size of the universal itemset (between 25 and 40, as mentioned by the TA), and
- the total number of transactions to be generated.

All transactions were generated **exclusively by sampling from this item universe**, and each transaction was generated independently using randomized probabilistic sampling.

No identical transactions were intentionally duplicated, and no shortcuts were used to artificially enforce runtime behavior.

Dataset Generation Logic

Each transaction was generated using the following controlled probabilistic strategy:

Dense Cores That Overlap

Overlapping groupings within the item universe were specified as many dense item subsets. One or two of these dense cores were chosen at random for each transaction, and items from the chosen cores were included with a high probability (about 0.8). As a result, Apriori is stressed by frequent itemsets and associated item occurrences.

Items with Medium-Support

The probability of items outside the dense cores being included was modest. This guarantees that at intermediate support criteria (10%–25%), a non-trivial proportion of itemsets continue to be common.

Uncommon Noise Products

To offer variation and prevent an excessively regular transaction structure, a tiny percentage of items were introduced with low probability.

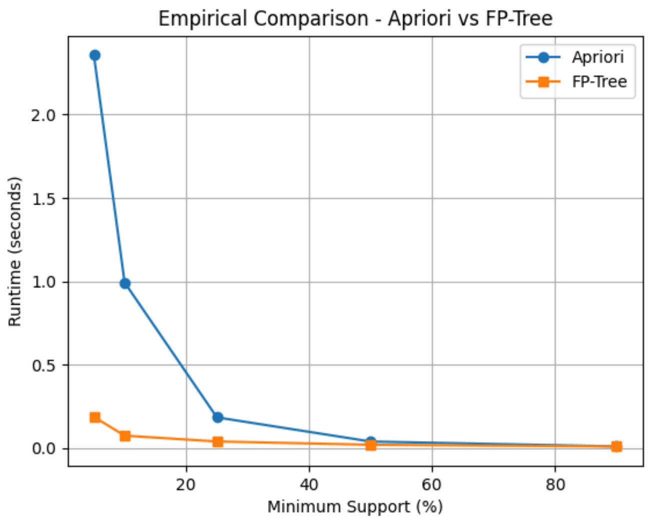
Controlled Transaction Length

A goal length range of 15–20 items was applied to each transaction. While staying within reasonable limitations, longer transactions boost Apriori's processing cost and increase the number of potential combinations.

All transactions were generated independently using random sampling from the universal itemset. The size of the universal itemset was set to **25** for our experiment.

Observed Results on the Constructed Dataset

The runtime plot for the constructed dataset shows **smoother and more controlled behavior** compared to Task 1.



Minimal Support (5%)

At low support, Apriori continues to lag behind FP-Growth, indicating the ongoing candidate explosion brought on by lengthy and dense transactions. But compared to Task 1, the runtime increase is much more modest, staying in the range of seconds rather than hours. This suggests that the created dataset emphasises Apriori while restricting extreme co-occurrence patterns.

By utilising FP-tree compression and eliminating explicit candidate enumeration, FP-Growth continues to outperform Apriori.

10%–25% Moderate Support

Both techniques show a progressive reduction in runtime as the support threshold rises. While FP-Growth builds smaller conditional FP-trees, Apriori gains from lower candidate numbers. Although it is still noticeable, the performance difference between the two algorithms is far less than it was in Task 1.

High Support (50%–90%)

Both techniques converge to a tiny constant runtime at high support thresholds. The majority of itemsets are trimmed early, making regular itemset mining insignificant. Instead of mining complexity, fixed overhead dominates runtime in this area.

Comparison Between Task 1 and Task 2

The comparison between the two tasks highlights the significant impact of dataset structure on algorithm performance:

- Extreme candidate explosion in Apriori is caused by the highly irregular and dense co-occurrence patterns in the original dataset in Task 1.
- Task 2's built dataset greatly minimises worst-case behaviour by enforcing controlled correlations and constrained item frequencies.
- In both tasks, FP-Growth consistently performs better than Apriori, but the difference is much more noticeable in Task 1.
- In both tasks, aggressive pruning and overhead-dominated execution are reflected in runtime convergence at high support.

References

1. **OpenAI.** *ChatGPT: Large Language Model for Conversational AI.*
   Used for assistance with code syntax and debugging, and summarizing explanations of algorithmic behavior.
   Available at: https://chat.openai.com
2. **Python Software Foundation.** *Python Language Reference, Version 3.10.*
   Used for implementing dataset generation scripts, experiment automation, and result processing.
   Available at: https://www.python.org
3. **Hunter, J. D.** *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering, 9(3), 90–95, 2007.
   Used for plotting and visualizing runtime comparisons between Apriori and FP-Growth algorithms.
4. **Borgelt, C.** *Efficient Implementations of Apriori and FP-Growth Algorithms.*
   Used for frequent itemset mining experiments in Tasks 1 and 2.
   (Algorithms executed via the provided mining binaries and scripts.)
5. All LLM generated code has been mentioned in the scripts, starting with the comment "#### LLM-assisted code (ChatGPT); all logic and correctness verified by the authors." and ending with the comment design "####".