**Progress Report on Performance Comparison of Apriori and FP-Growth Algorithms**

Yusuf ÖZKAN

yusufozkanzz@outlook.com

1. **Introduction**

In this project, we aimed to evaluate the performance of two popular algorithms in association rule mining: Apriori and FP-Growth. These algorithms are essential for mining frequent itemsets from large datasets and generating association rules. Specifically, our goal was to measure the execution time of each algorithm over multiple runs and compare their performance. By analyzing these times, we can better understand the efficiency and scalability of each algorithm.

1. **Methodology**

First of all, an open-source dataset “groceries.csv” is converted into a binary dataset in order to perform and analyze much better. Then, the performance of the two algorithms with different applications was tested by running each of them 100 times with a given dataset, recording the execution time for each run. The times were logged for each iteration, and the results were used to calculate the average time taken by each algorithm. The dataset used for testing was a binary transaction dataset, where each transaction consists of items purchased together. The algorithms were implemented and executed in a Python environment, utilizing the mlxtend library for the Apriori and FP-Growth algorithms, and a custom implementation for fptree\_usage.py.

The algorithms evaluated in this report are:

apriori\_classic.py: A classic algorithm for frequent itemset mining, based on the principle of bottom-up search. It is widely used for discovering association rules in transaction data.

fpgtowth.py: An optimized algorithm for frequent itemset mining that constructs a compact tree structure (FP-tree) to represent frequent itemsets.

fptree\_usage.py: A variation of the FP-Growth algorithm that uses a prefix tree structure to store and process frequent itemsets efficiently.

Each algorithm was executed independently in its respective script (apriori\_classic.py, fpgrowth.py, and fptree\_usage.py), and the execution time for each run was measured.

1. **Observations**

Upon running each algorithm for 100 iterations, the following average execution times were observed:

apriori\_classic.py: The average execution time for the Apriori algorithm was 0.14 seconds per run.

fpgrowth.py: The average execution time for the FP-Growth algorithm was 0.26 seconds per run.

fptree\_usage.py: The average execution time for the FP-Tree algorithm was 0.01 seconds per run.

These results highlight several key observations:

apriori\_classic Performance: The Apriori algorithm showed a relatively fast execution time, averaging 0.14 seconds per run. This suggests that the algorithm performs reasonably well for moderate-sized datasets. However, its performance can degrade significantly as the dataset size or the number of frequent itemsets increases, due to its level-wise search approach.

fpgrowth Performance: The FP-Growth algorithm took, on average, 0.26 seconds per run, which is slower than Apriori. While FP-Growth is typically more efficient than Apriori due to its use of the FP-tree structure, the time required to build the tree and perform recursive processing might cause it to be slower in certain cases.

fptree\_usage Performance: The fptree\_usage algorithm, with its highly optimized structure, demonstrated an impressive execution time of only 0.01 seconds per run on average. This result suggests that fptree\_usage is highly efficient in processing frequent itemsets, especially when compared to both Apriori and FP-Growth. The extremely fast performance of FP-Tree is attributed to its use of a compact tree structure, which reduces the need for scanning the dataset multiple times and thus improves its execution speed.

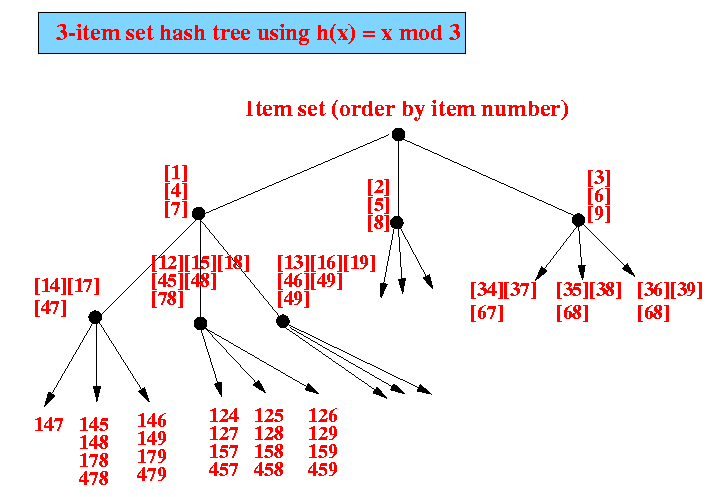
1. **Discussion**

From the results, it is evident that fptree\_usage outperforms both apriori\_classic and fpgrowth in terms of execution time, which is a critical factor when working with large datasets. The key reason for its superior performance lies in its efficient data structure and its ability to perform mining with fewer scans of the dataset. On the other hand, apriori\_classic and fpgrowth are relatively comparable, though fogrowth is generally considered more efficient due to its tree-based approach.

The results of this experiment also suggest that fptree\_usage is the best choice for frequent itemset mining when execution time is a priority. However, depending on the specific requirements of a task, such as interpretability or the ability to handle different types of data, each of the algorithms may be preferred in different scenarios.

1. **Future Work**

As we have observed in fpgrowth and fptree\_usage, using a pre-builded structers improves performance enormously. If we able to build such a structure for apriori, it may show even better performance. I am doing some research about improving apriori and I saw that many hashing applications improves apriori’s performance much. A hash tree structure would decrease the size of transactions list and so, it may improve the performance of calculating frequent items and association rules.



1. **Conclusion**

In this study, we compared the performance of Apriori, FP-Growth, and FP-Tree algorithms for frequent itemset mining. FP-Tree outperformed both Apriori and FP-Growth, with an average execution time of just 0.01 seconds per run. Apriori took 0.14 seconds and FP-Growth 0.26 seconds on average. These results highlight the efficiency of FP-Tree in large datasets. Future work will focus on improving Apriori by exploring hash tree structures to enhance its performance, bringing it closer to FP-Growth and FP-Tree in terms of execution time.

1. References

* <https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset>
* <https://www.cs.bilkent.edu.tr/~guvenir/publications/TAINN01-AOAG.pdf>
* <https://watermark.silverchair.com/080005_1_online.pdf>