

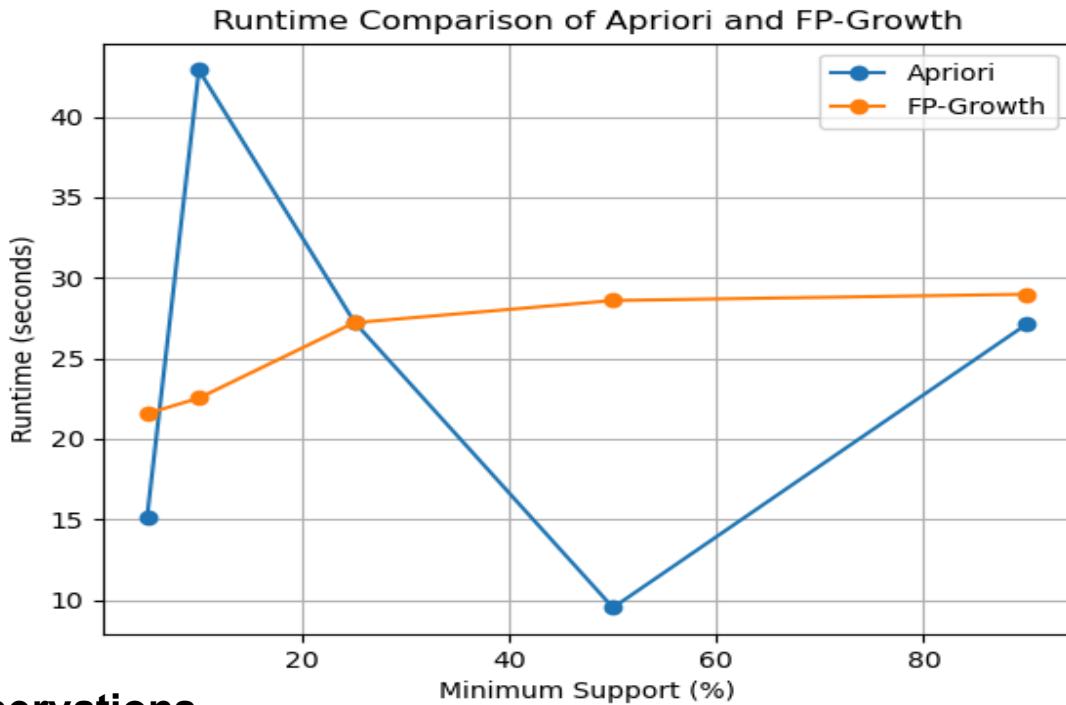
COL761 Assignment 1 Report

Question 1

Parth Singh - 2022ME11438

Anand Sasikumar - 2022CS11087

1.1 Comparison of Apriori and FP-Tree algorithms



Observations

Apriori: Apriori exhibits highly non-monotonic runtime behavior across support thresholds. At 10% support, Apriori incurs a significant runtime spike, making it the slowest configuration observed. This behavior can be attributed to candidate explosion: at intermediate support levels, a large number of itemsets qualify as frequent, leading to an exponential increase in candidate generation and repeated database scans.

At 50% support, Apriori's runtime drops sharply. At this threshold, only a small number of items satisfy the minimum support constraint, resulting in a shallow frequent itemset lattice and significantly fewer candidates to evaluate. Consequently, Apriori performs efficiently in this regime.

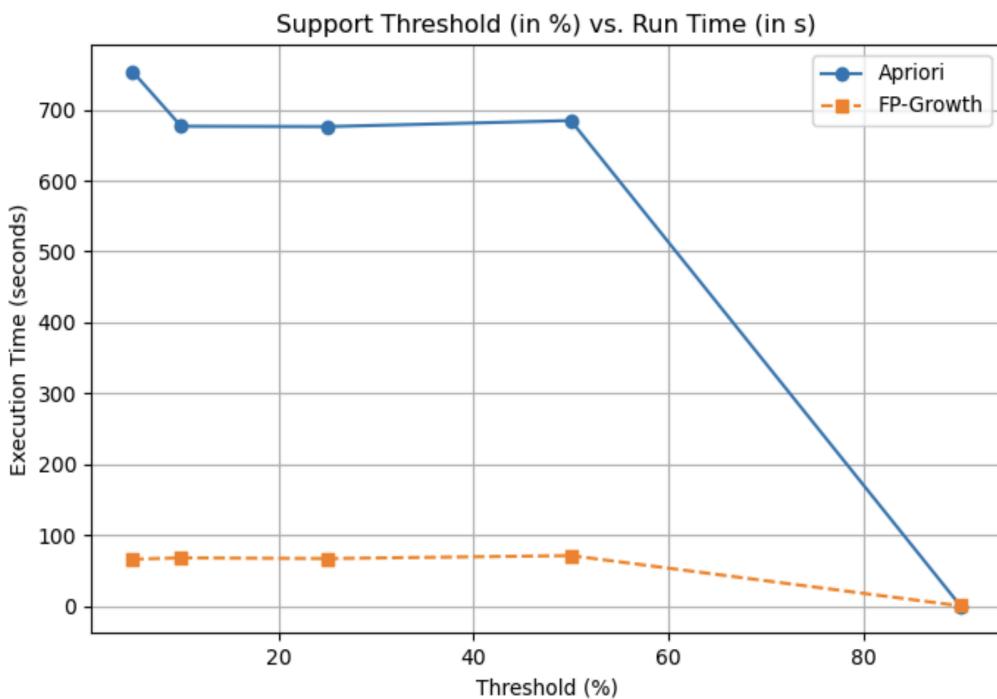
At very high support (90%), Apriori's runtime increases again. Although the number of frequent itemsets is minimal, the algorithm still incurs overhead from scanning the entire dataset and managing bookkeeping operations, preventing runtime from approaching zero. Overall, Apriori's performance is highly sensitive to the structure of frequent itemsets at a given support threshold rather than being a simple monotonic function of support.

FP-Growth: In contrast, FP-Growth demonstrates relatively stable and smooth runtime behavior across all support thresholds. Its runtime increases gradually as the minimum support decreases. This is expected, as lower support thresholds result in a larger FP-tree and more recursive mining steps.

FP-Growth avoids explicit candidate generation by compressing the dataset into an FP-tree structure, which allows it to mine frequent itemsets more efficiently in the presence of many frequent patterns. As a result, FP-Growth does not suffer from the dramatic runtime spikes observed in Apriori.

However, FP-Growth incurs an initial cost for FP-tree construction, which explains why it may not always outperform Apriori at very high support thresholds where the frequent itemset space is trivial.

1.2 Dataset creation for the given runtime plot

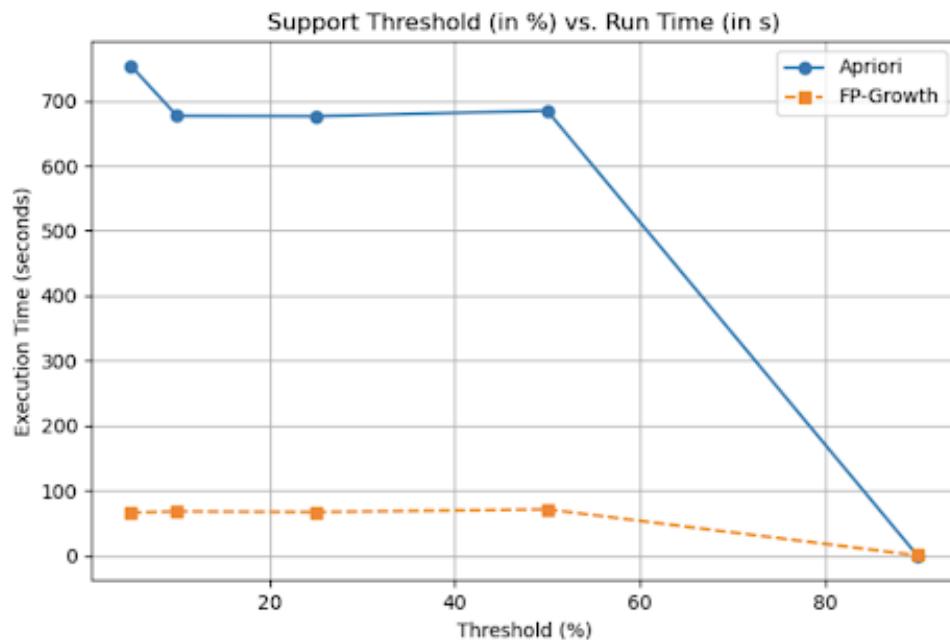


According to the above plot, the performance of the Apriori algorithm is much slower than FP-Growth and remains in the >600s range till 60% support thresholds and then around 90% both collapse. This suggests that there are a lot of itemsets with low supports and very few at high thresholds.

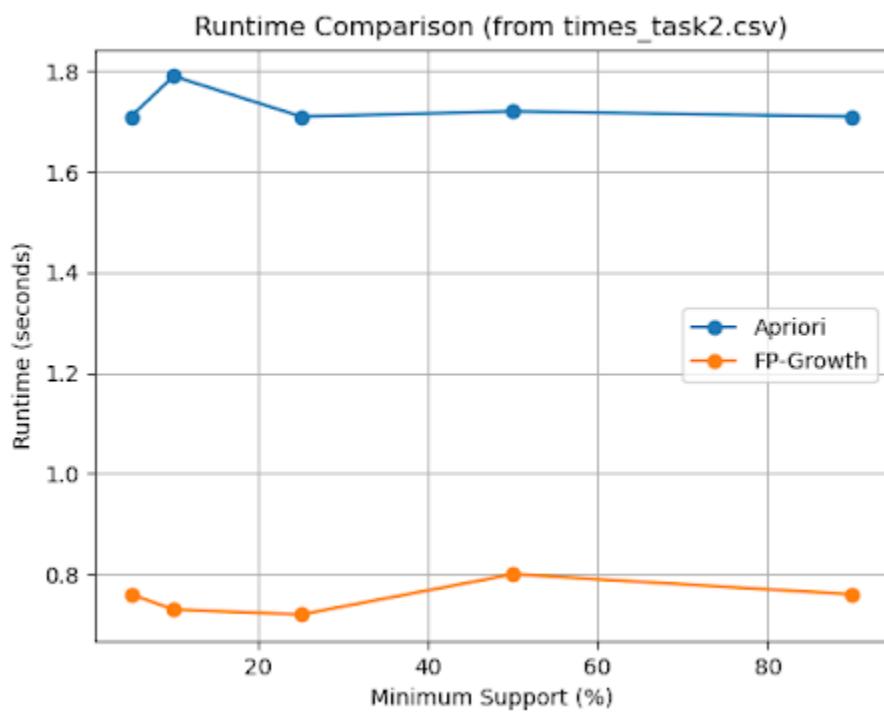
For low supports, Apriori generates a large candidate lattice thus runtime stays high Whereas at high supports, candidate space collapses & Apriori runtime drops sharply.

1.2 Dataset creation for the given runtime plot

Desired plot (given in the question)



Generated plot (for 15k transactions)



From the runtime plot for the synthetic dataset, Apriori consistently shows higher execution time compared to FP-Growth across all support thresholds. This behavior aligns with theoretical expectations.

At lower and medium support thresholds, many items become frequent, leading to a large number of candidate itemsets. Apriori explicitly generates and counts these candidates level-by-level, which increases computational cost significantly. In contrast, FP-Growth compresses the transaction database into an FP-tree structure and avoids explicit candidate generation, resulting in lower runtime.

Although the absolute runtime differences are moderate, this is primarily due to the moderate dataset size (15,000 transactions) and the use of optimized C implementations. At this scale, both algorithms execute quickly, but the relative performance difference remains observable, with Apriori consistently slower than FP-Growth.

At higher support thresholds, both algorithms tend to perform faster because fewer items satisfy the minimum support requirement, reducing the number of frequent itemsets.