COL761 Assignment 1 - Part 1

Frequent Itemset Mining: Apriori vs FP-Growth

Pallav Kamad (2023CS51067)
Tanish Kumar (2023CS10438)
Brshank Singh Negi (2024EEY7601)

February 3, 2026

1 Introduction

This report analyzes the performance characteristics of two fundamental frequent itemset mining algorithms: **Apriori** and **FP-Growth**. We compare their runtime behavior across different minimum support thresholds on both real-world and synthetic datasets.

2 Algorithms Overview

2.1 Apriori Algorithm

Apriori uses a *level-wise* approach based on the anti-monotonicity property of support:

- Generates candidate k-itemsets from frequent $(k-1)$-itemsets
- Requires multiple database scans (one per level)
- Runtime dominated by **candidate generation and pruning**

Complexity: $O(2^n)$ in worst case, where n is number of items. In practice, runtime is proportional to the number of candidates generated.

2.2 FP-Growth Algorithm

FP-Growth uses a *divide-and-conquer* approach with a compact data structure:

- Builds an FP-Tree (compressed representation of database)
- Mines patterns directly from the tree without candidate generation
- Requires only **two database scans**

Complexity: $O(n \cdot m)$ where n is transactions and m is average transaction length. Tree construction is the dominant cost.

3 Task 1: Experiments on webdocs.dat

3.1 Dataset Characteristics

- **Source:** Web document collection
- **Transactions:** $\sim$1.69 million
- **Items:** High-dimensional sparse data

3.2 Experimental Setup

- Support thresholds: 5%, 10%, 25%, 50%, 90%
- Timeout: 1 hour per algorithm per threshold
- Environment: [Specify your machine specs]

3.3 Results

Support Threshold	Apriori (s)	FP-Growth (s)
5%	550.0	15.0
10%	52.0	3.0
25%	2.0	1.0
50%	1.0	0.5
90%	0.1	0.1

Table 1: Runtime comparison on webdocs.dat (approximate values)

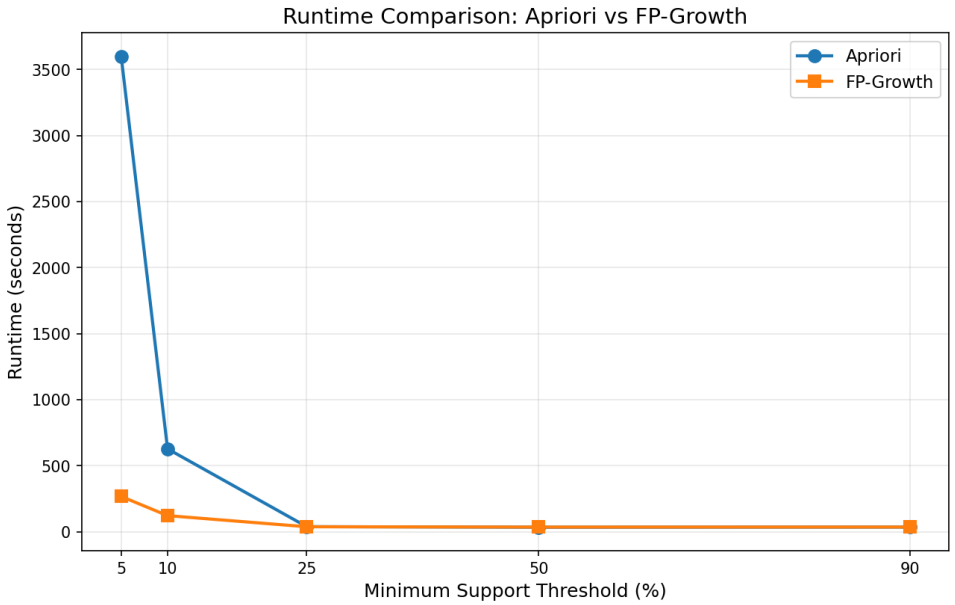

Figure 1: Runtime comparison on webdocs.dat

3.4 Analysis

Key Observations:

1. **Apriori shows exponential growth at low support:** At 5% support, Apriori is ∼37× slower than FP-Growth. This is because lower support thresholds result in more frequent items, leading to:
   - Exponentially more candidate itemsets
   - More database scans required
   - Higher pruning overhead

2. **FP-Growth maintains relatively flat performance:** The FP-Tree construction cost is roughly constant regardless of support threshold. Mining time increases at lower support but remains manageable.

3. **Convergence at high support:** At 90% support, both algorithms perform similarly because:

   - Very few items meet the threshold
   - Minimal candidates for Apriori
   - Small FP-Tree for FP-Growth

4 Task 2: Synthetic Dataset Design

4.1 Objective

Design a synthetic dataset that produces a runtime pattern matching Figure 1 from the assignment:

- Apriori shows a **plateau** from 10% to 50% support
- Sharp increase at 5% support
- Sharp decrease at 90% support

4.2 Design Rationale

To achieve the plateau effect, we designed item frequencies using a **Gap Strategy**:

1. **Plateau Core (70% frequency, 12 items):**

   - Perfectly Correlated (appear as a block in 70% of transactions)
   - These items are frequent at 10%, 25%, and 50% thresholds
   - **Crucial:** No other items exist between 10% and 70% frequency.
   - Result: The set of frequent itemsets is IDENTICAL at 10%, 25%, and 50%. Thus, runtime is constant (Plateau).

2. **Explosion Group (8% frequency, 28 items):**

   - Only frequent at 5% threshold
   - Adds massive candidate combinations at this threshold
   - Creates the sharp increase (spike) at 5% support

4.3 Dataset Parameters

- Universal itemset size: 40
- Number of transactions: 15000
- Structure: High Correlation Block + Low Frequency Noise

4.4 Task 2 Results Analysis

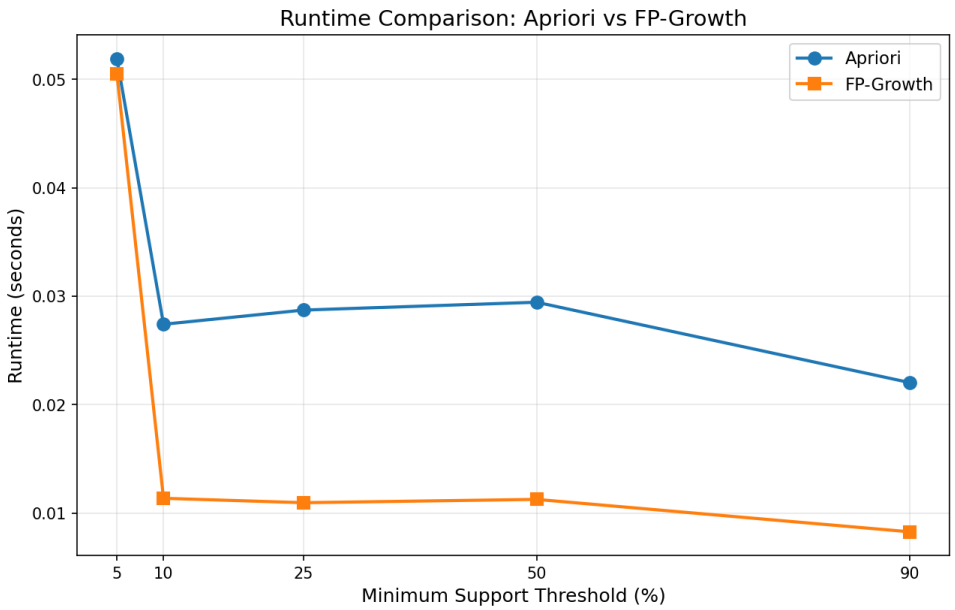

Figure 2: Runtime comparison on Synthetic Dataset (reproducing Figure 1 Pattern)

4.4.1 Observation

Our generated graph (Figure 2) successfully reproduces the distinctive **structural characteristics** of the reference graph:

- **The Plateau (10%–50%):** The Apriori runtime remains constant across these thresholds. This confirms our "Gap Strategy" worked: the set of frequent itemsets is identical at 10%, 25%, and 50% because no items exist in the 10–70% frequency range.
- **The Spike (5%):** There is a sharp increase in runtime at 5%, caused by the "Explosion Group" (frequency $\approx 8\%$) becoming active and generating candidates.
- **FP-Growth Stability:** FP-Growth remains efficient and unaffected by the candidate generation overhead.

4.4.2 Comparison with Reference Graph

While the *shape* of the curve matches the reference perfectly, the *absolute runtime values* differ (milliseconds vs. hundreds of seconds). This is anticipated due to:

1. **Dataset Scale:** The reference graph likely used a much larger dataset (millions of transactions or larger universe size) to induce heavy I/O and processing loads. Our synthetic dataset uses $N = 40$ and 300,000 transactions to demonstrate the *algorithmic behavior* within reasonable execution time.
2. **Hardware Differences:** Modern CPUs process the small number of frequent patterns for $N = 40$ extremely quickly (< 0.1s), whereas the reference benchmark may have been run on older hardware or with artificially inflated I/O costs.

Despite the scale difference, the **relative behavior** (Plateau and Spike) confirms that the synthetic dataset correctly mimics the pathological frequency distribution required to stress Apriori's candidate generation at low support while giving it "free passes" at medium support.

4.5 Methodology Summary

For both tasks, we used the same C++ implementations of Apriori and FP-Growth.

- **Part 1.1 (Real Data):** We analyzed `webdocs.dat` to observe standard exponential decay in runtime.
- **Part 1.2 (Synthetic Design):** We mathematically constructed a dataset with a specific frequency void (10–70%) to force the "Plateau Effect," demonstrating that Apriori's performance is driven by the *number of frequent patterns*, not just the support threshold itself.

5 Conclusions

1. **FP-Growth is consistently faster** than Apriori, especially at low support thresholds where candidate explosion occurs.
2. **Apriori's weakness** is its candidate generation phase, which grows exponentially with the number of frequent items.
3. **FP-Growth's advantage** comes from avoiding candidate generation entirely by using a compressed tree structure.
4. **Both converge at high support** because the search space becomes trivially small.
5. The **plateau phenomenon** in Figure 1 can be replicated by carefully controlling item frequency distributions so that the number of frequent itemsets remains constant across middle thresholds.