University of Dhaka

Department of Computer Science and Engineering

CSE-4255 : Introduction to Data Mining and Warehousing Lab

Lab Assignment: Implementation and time-memory usage comparison of Apriori and FP-Growth Algorithms on different datasets

Submitted On:

April 30, 2025

Submitted To:

Dr. Chowdhury Farhan Ahmed

Professor
Department of Computer Science & Engineering
University of Dhaka

Md. Mahmudur Rahman

Assistant Professor
Department of Computer Science & Engineering
University of Dhaka

Submitted By:

Himel Chandra Roy

Roll No : 13

Registration No : 2020915623

Contents

1 Introduction

Frequent pattern mining is a fundamental task in data mining, aiming to discover recurring relationships among items in large transactional datasets. Two of the most widely used algorithms for this purpose are the Apriori algorithm and the FP-Growth (Frequent Pattern Growth) algorithm. **Apriori** relies on a breadth-first search strategy and generates candidate itemsets, which can become computationally expensive with large or dense datasets. In contrast, **FP-Growth** employs a depth-first strategy and uses a compact data structure known as the FP-tree to mine frequent patterns without candidate generation, often resulting in improved performance.

This report presents a comparative analysis of the Apriori and FP-Growth algorithms across various datasets. The primary focus is on evaluating their execution time and memory consumption under different data characteristics, such as transaction size, itemset density, and dataset scale. Through empirical experiments, this study aims to highlight the strengths and limitations of each algorithm and provide guidance on their suitability for different data mining scenarios.

2 Apriori Algorithm

1. **Load Dataset:**
   - Read transaction data from the file.
   - Store each transaction as a set of items.
   - Count the frequency of each individual item.
   - Calculate the minimum support count using the user-defined support percentage.
2. **Generate Frequent 1-Itemsets (L1):**
   - Filter items whose frequency $\geq$ minimum support.
   - Store them as frequent 1-itemsets.
3. **Iteratively Generate Frequent k-Itemsets (Lk):**
   - For $k \geq 2$, generate candidate k-itemsets from L_{k-1} using the `apriori_gen` method.
   - For each transaction, check which candidates are subsets.
   - Count and filter candidates with support $\geq$ threshold.
   - Repeat until no more frequent itemsets can be generated.
4. **Track Performance:**
   - Measure execution time per level.
   - Measure peak memory usage.
5. **Return Results:**
   - Return all frequent itemsets grouped by size.
   - Return time per level and memory usage.

3 FP-Growth Algorithm

1. **Load Dataset and Pre-process:**
   - Read transactions from file and store in list.
   - Count frequency of each item.
   - Calculate the minimum support count.
2. **Build Header Table:**
   - Store items that meet minimum support with their counts and node link.
   - Sort items in descending order of frequency.
3. **Build the FP-Tree:**
   - For each transaction:
     - Filter and sort items by header table order.
     - Insert items into the FP-tree.
     - Update counts and maintain node links via the header table.
4. **Mine Frequent Patterns:**
   - For each item in the header table (from least to most frequent):
     - Construct conditional pattern base by tracing paths to root.
     - Build conditional FP-tree using these paths.
     - Recursively mine conditional trees to extract patterns.
5. **Track Performance:**
   - Measure time for tree building and mining.
   - Measure peak memory usage.
6. **Return Results:**
   - Return all frequent patterns grouped by itemset size.
   - Return total time and memory usage.

4 Coding Environment

All code implementations and experiments were performed using the following software and hardware configuration:

- **Programming Language:** Python 3.10
- **Integrated Development Environment (IDE):** Visual Studio Code
- **Libraries Used:**
  - `os` – for interacting with the operating system
  - `psutil` – for monitoring system resource usage
  - `time` – for measuring execution time
  - `math` – for mathematical operations
  - `matplotlib` – for data visualization
  - `collections` – for efficient data structures like `defaultdict`
  - `itertools` – for creating iterators for combinatorial constructs
- **Operating System:** Windows 11 (64-bit)
- **Processor:** 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- **RAM:** 16 GB

This environment provided a stable and efficient setup for developing and testing the implemented algorithms.

5 Results

To evaluate the performance of the Apriori and FP-Growth algorithms, experiments were conducted on multiple benchmark datasets with varying characteristics such as transaction volume, average transaction length, and itemset density. The key performance metrics analyzed were execution time and memory consumption, measured for each algorithm under identical support thresholds and computational environments.

5.1 Execution Time Comparison

Across all datasets, the FP-Growth algorithm consistently outperformed Apriori in terms of execution time. For smaller datasets with low itemset density, the difference was minimal; however, as dataset size and density increased, the performance gap widened significantly. In dense datasets, Apriori's execution time increased exponentially due to the costly candidate generation step, while FP-Growth maintained relatively stable performance by avoiding candidate generation through its FP-tree structure.

5.2 Memory Consumption Comparison

In terms of memory usage, the results were more nuanced. Apriori, despite its slower execution, showed lower memory usage in sparse datasets due to its simple data structures. However, as dataset complexity increased, Apriori's memory usage rose sharply due to the exponential growth in candidate itemsets. FP-Growth, on the other hand, required more initial memory to construct the FP-tree but used it efficiently during mining, resulting in better scalability.

6 Result Figures

6.1 Accidents

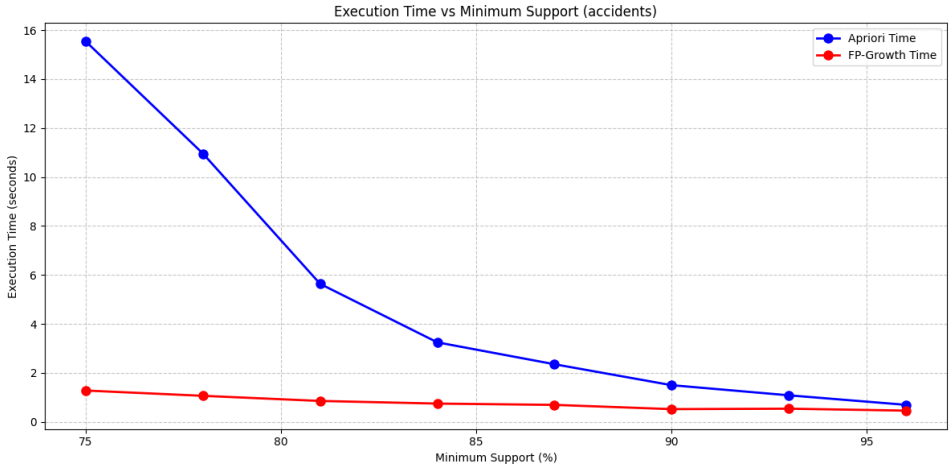

Figure 1: Execution time of Accidents dataset.

Figure 1 shows the execution time vs minimum support graph for accidents dataset for both Apriori algorithm and FP-Growth algorithm.

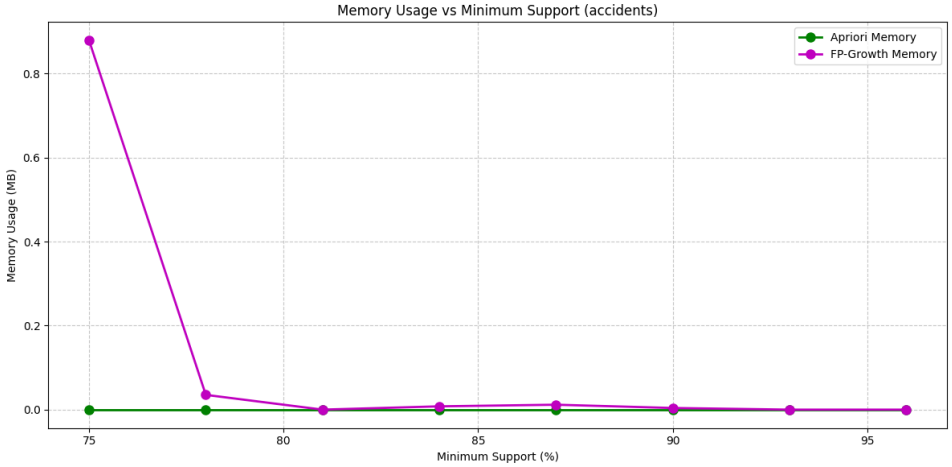

Figure 2: Memory usage of accidents dataset

Figure 2 shows the memory usage vs minimum support graph for accidents dataset for both Apriori algorithm and FP-Growth algorithm.

6.2 Connect

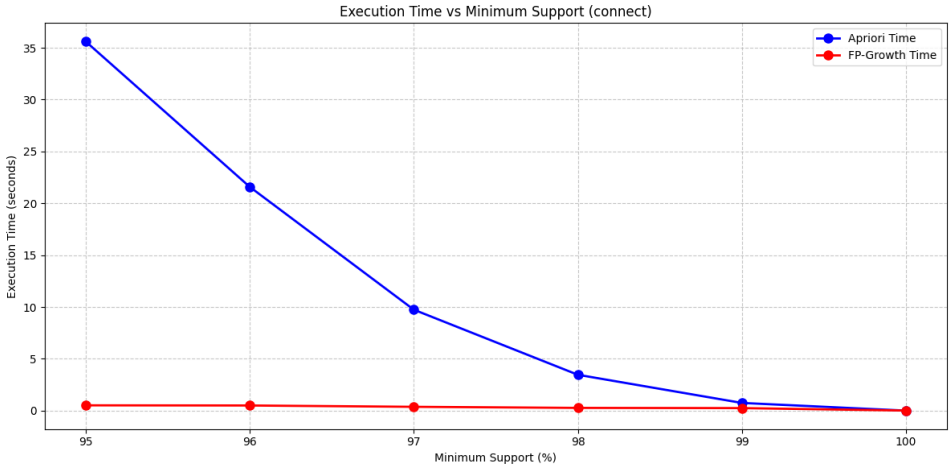

Figure 3: Execution time of Connect dataset.

Figure 3 shows the execution time vs minimum support graph for the Connect dataset for both Apriori algorithm and FP-Growth algorithm.

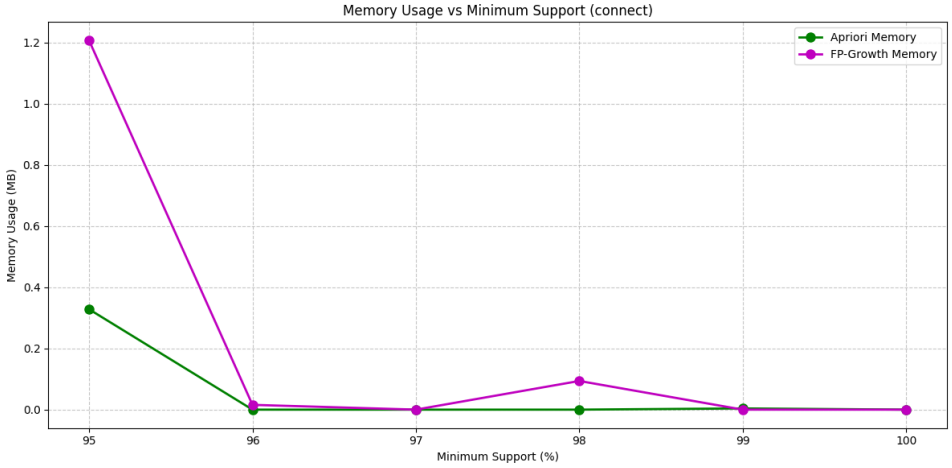

Figure 4: Memory usage of Connect dataset.

Figure 4 shows the memory usage vs minimum support graph for the Connect dataset for both Apriori algorithm and FP-Growth algorithm.

6.3 Chess

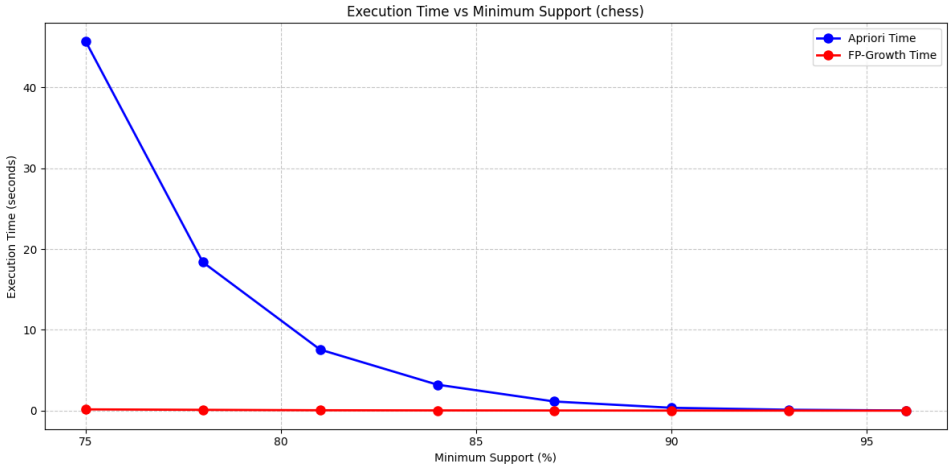

Figure 5: Execution time of Chess dataset.

Figure 5 shows the execution time vs minimum support graph for the Chess dataset for both Apriori algorithm and FP-Growth algorithm.

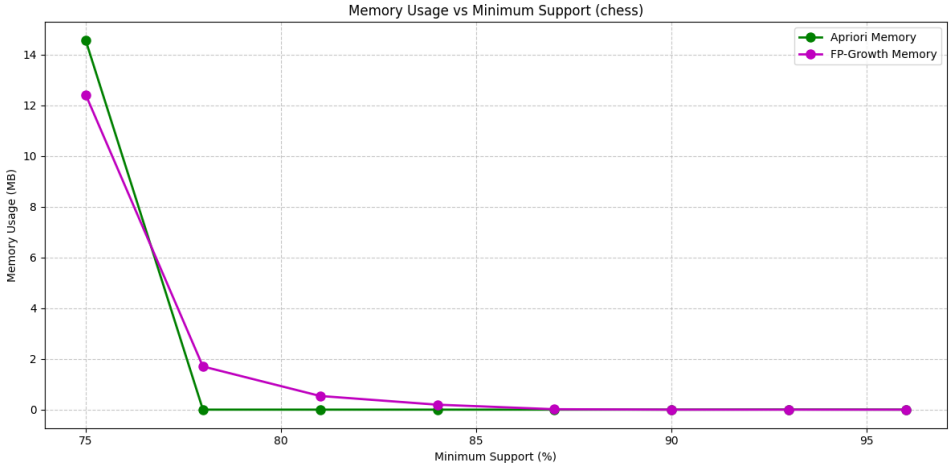

Figure 6: Memory usage of Chess dataset.

Figure 6 shows the memory usage vs minimum support graph for the Chess dataset for both Apriori algorithm and FP-Growth algorithm.

6.4 Mushroom

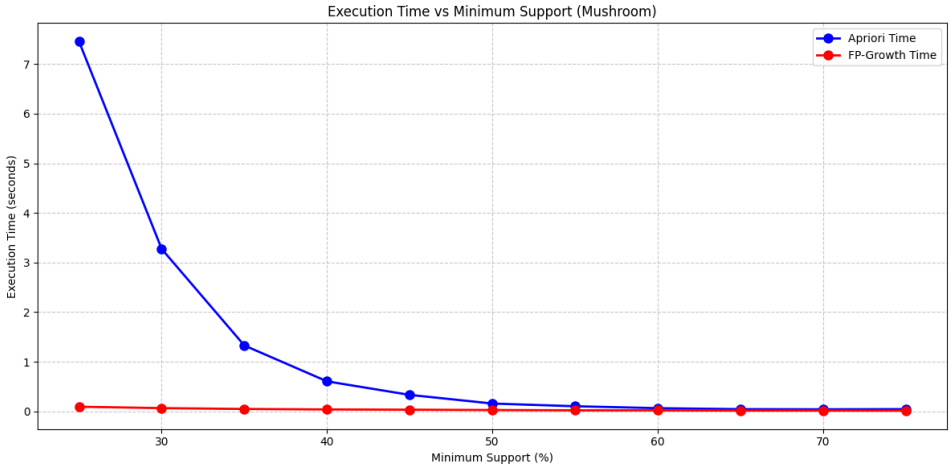

Figure 7: Execution time of Mushroom dataset.

Figure 7 shows the execution time vs minimum support graph for the Mushroom dataset for both Apriori algorithm and FP-Growth algorithm.

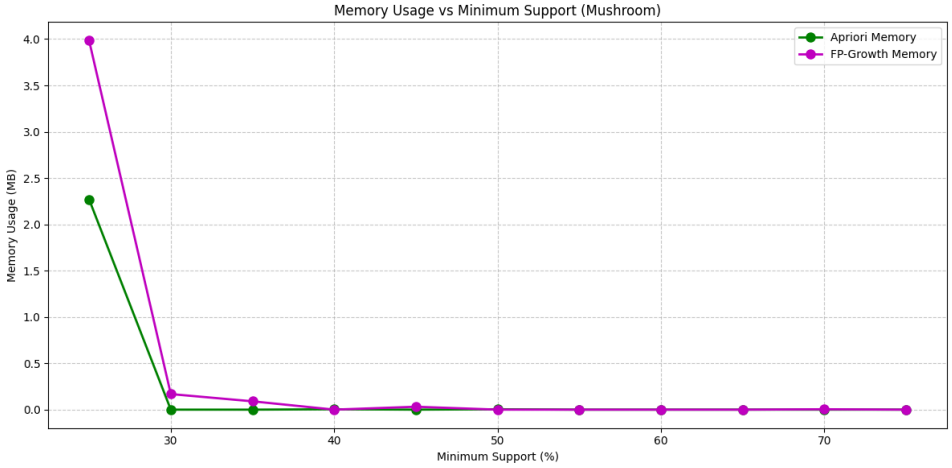

Figure 8: Memory usage of Mushroom dataset.

Figure 8 shows the memory usage vs minimum support graph for the Mushroom dataset for both Apriori algorithm and FP-Growth algorithm.

6.5 Retail

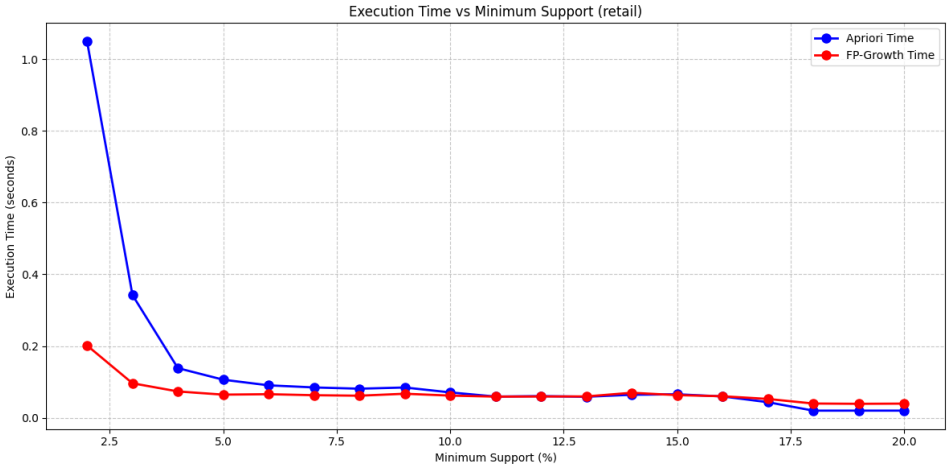

Figure 9: Execution time of Retail dataset.

Figure 9 shows the execution time vs minimum support graph for the Retail dataset for both Apriori algorithm and FP-Growth algorithm.

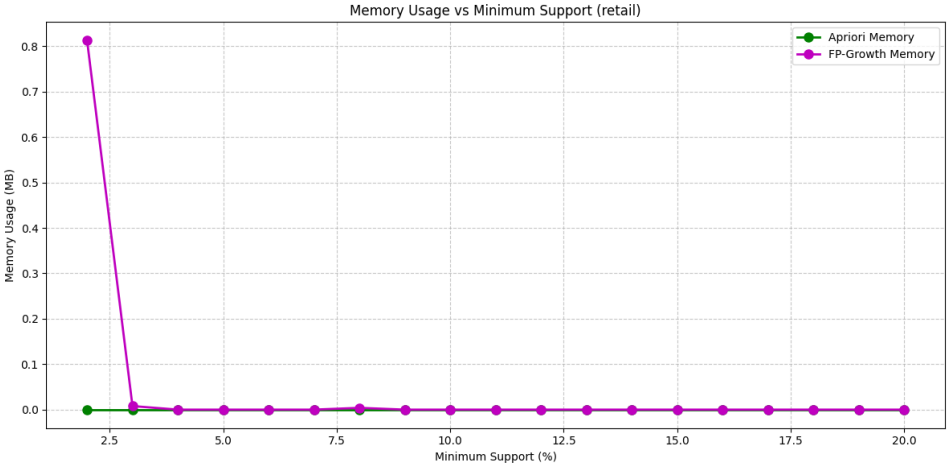

Figure 10: Memory usage of Retail dataset.

Figure 10 shows the memory usage vs minimum support graph for the Retail dataset for both Apriori algorithm and FP-Growth algorithm.

6.6 Kosarak

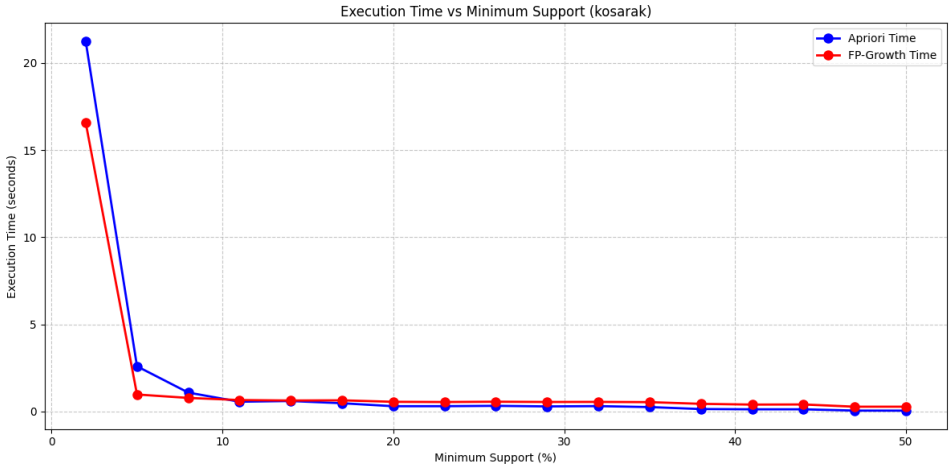

Figure 11: Execution time of Kosarak dataset.

Figure 11 shows the execution time vs minimum support graph for the Kosarak dataset for both Apriori algorithm and FP-Growth algorithm.

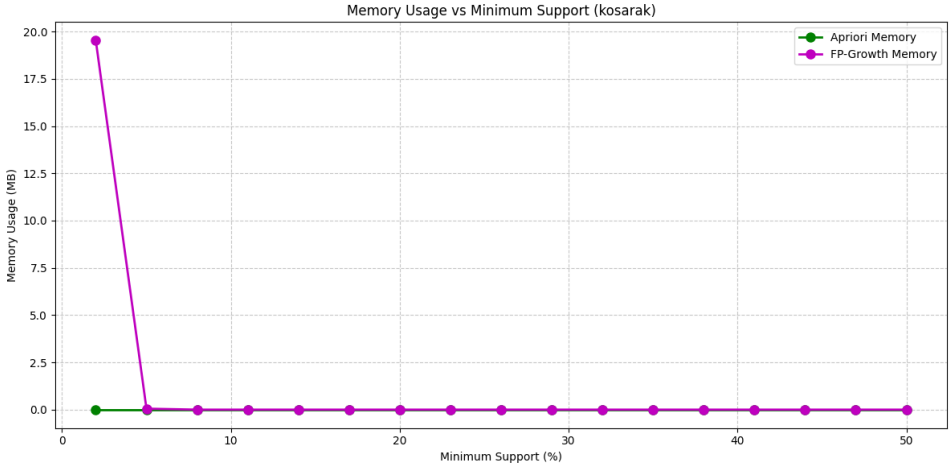

Figure 12: Memory usage of Kosarak dataset.

Figure 12 shows the memory usage vs minimum support graph for the Kosarak dataset for both Apriori algorithm and FP-Growth algorithm.

6.7 Pumsb

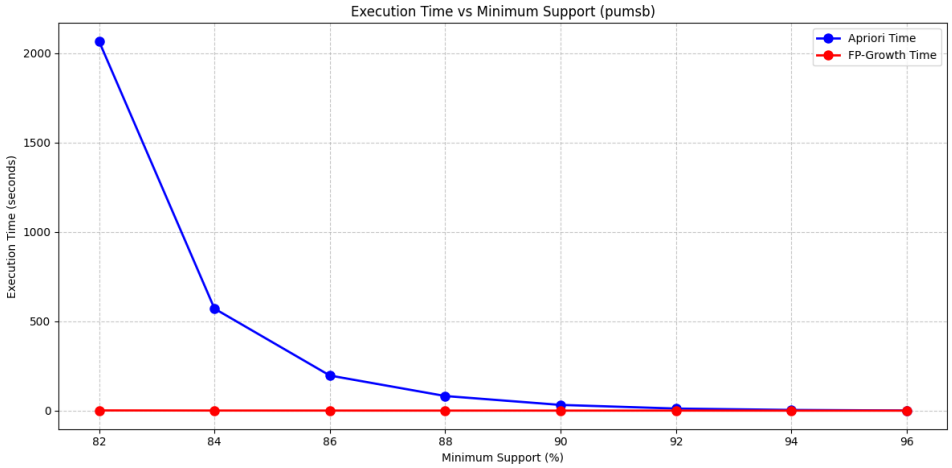

Figure 13: Execution time of PUMSB dataset.

Figure 13 shows the execution time vs minimum support graph for the PUMSB dataset for both Apriori algorithm and FP-Growth algorithm.

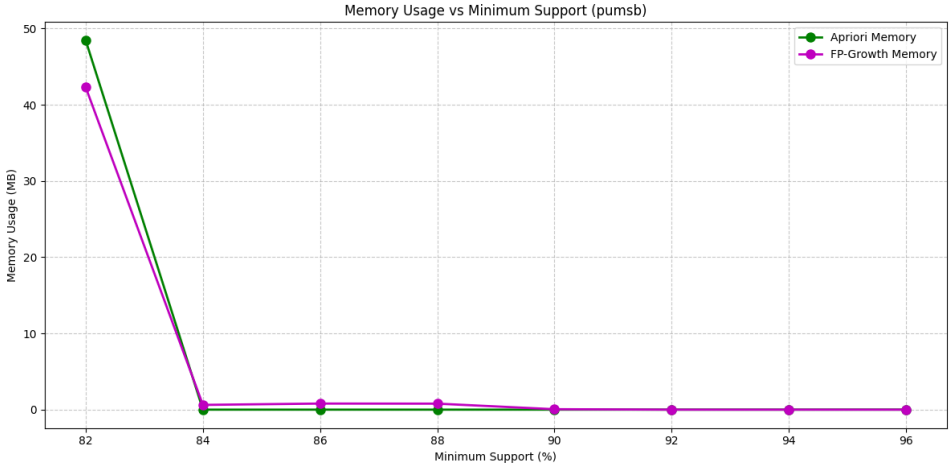

Figure 14: Memory usage of PUMSB dataset.

Figure 14 shows the memory usage vs minimum support graph for the PUMSB dataset for both Apriori algorithm and FP-Growth algorithm.

7 Result analysis

The experimental results clearly demonstrate the trade-offs between the Apriori and FP-Growth algorithms under varying data conditions. Apriori's performance deteriorates significantly with the increase in dataset size and itemset density, primarily due to its reliance on candidate generation and repeated scans of the database. This behavior leads to both longer execution times and higher memory consumption in complex datasets. Conversely, FP-Growth maintains a consistent advantage in performance by eliminating candidate generation and using a compact FP-tree structure, which allows for faster pattern mining and better scalability. While FP-Growth incurs a higher initial memory overhead due to tree construction, this is offset by its reduced processing time and lower memory growth rate as dataset complexity increases. These observations highlight that FP-Growth is better suited for large-scale or dense transactional datasets, whereas Apriori may remain a viable option for simpler, sparse datasets with fewer frequent patterns.