East West University

Course title: Data Mining

Course code: CSE477

Summer 2023

Section: 1 & 2

“Mini Project Report”

" Implementation and Performance Analysis of FP-Growth Algorithm."

Submitted To:

Jesan Ahmed Ovi

Senior Lecturer,

Department of Computer Science and Engineering

East West University, Dhaka

Submitted By:

Name	ID	Section
K. M. Safin Kamal	2020-1-60-235	1
Mysha Maliha Priyanka	2020-1-60-230	1
Nahida Hoque	2020-1-60-147	2
Md. Adnan Morshed	2020-1-60-155	1

Department of Computer Science and Engineering

East West University, Dhaka

Date of Submission: 14/ 09 / 2023

Title: Implementation and Performance Analysis of FP-Growth Algorithm

Project Description:

This mini project focuses on implementing the FP-Growth algorithm in Python for frequent pattern mining. FP-growth is a fast way to find frequent patterns in big datasets. This method doesn't require candidate generation and can be much faster than the Apriori algorithm. The goal of this project is to develop a Python program that takes a transactional dataset as input and mines the frequent patterns using the FP-Growth algorithm.

This project has a few important steps:

1. Data Loading and Preprocessing: The ‘read data’ function reads transaction data from a file. It then prepares the data by splitting each file line into separate parts, called tokens. A dictionary is made where each entry represents a transaction and its frequency and then it is used for further processing.

2. FP-tree Construction: The function ‘create_HTandFPTree’ makes a Header Table and an FP-Tree. The Header Table has information about how often each item appears. The FP tree is a type of tree where each node represents an item. The nodes are connected to show the order and frequency of items in transactions.

3. FP tree growth: The 'updateTree' function updates the FP tree with a transaction in a repetitive manner. The 'update_NodeLink' function changes the links between nodes for items in the HeaderTable. These actions help create the FP-Tree in a more efficient way.

4. Mining Frequent Patterns: The 'uptraverse' and 'find_branches' functions are used to find conditional pattern bases for a given item. The function 'Mine_Tree' mines frequent patterns by creating special trees for each item. The frequent itemsets are kept in the ‘frequent_itemset’ list.

5. Result Generation and Analysis: Finally, the program shows how long the process took and shows the frequent patterns with their support values. It also calculates the support in percentages (relative support).

The main goal of this project is to efficiently mine frequent itemsets from transaction data using the FP-growth algorithm and provide a clear analysis of the discovered patterns. After finding these patterns, the project will analyze them, and we can gain a deeper understanding of frequent pattern mining techniques and their applications in real-world scenarios.

Dataset Description:

Two datasets, namely chess and mushroom, were utilized in the study. The chess dataset comprises 3196 transactions, each involving 75 items. In contrast, the mushroom dataset contains 8124 transactions with 119 items. The sizes of the chess and mushroom datasets are approximately 335 KB and 558 KB, respectively.

Dataset pre-processing:

In both of our datasets, we have meticulously maintained a unique item occurrence policy, ensuring that each item appears only once. Consequently, we have effectively eliminated any duplications of items within a single transaction.

Implementation:

The FP Growth algorithm is implemented in python language. Pycharm software is used for this. The configuration of the computer is Intel i7 13th 13700k (5.40 GHz) CPU, 32GB DDR5 (5200MHz) RAM, RTX 3060ti GDDR6 GPU and Windows 11 pro–operating system.

Result Analysis:

In this study, we'll explore the outcomes of the FP Growth algorithm on two different sets of data. Initially, we'll delve into the Chess dataset, followed by the Mushroom dataset. We'll do this by considering different thresholds, specifically at 60%, 65%, 70%, 80%, and 90%. Additionally, we'll conduct a comparison between the FP Growth algorithm and the Apriori algorithm to gain a deeper understanding of their performance.

Dataset: Chess

We applied our custom FP-Growth algorithm to the Chess dataset. When we set different thresholds, we found that we got varying amounts of frequent patterns. You can see the details in Table 1 and Figure 1, which display the number of frequent patterns we found at different threshold levels.

Table 1: number of frequent patterns for each threshold for chess data

Threshold	**Number of frequent patterns**
60%	228175
65%	102033
70%	48152
80%	8166
90%	609

In our observations, we set the threshold at 60%, we find the largest number of patterns. However, as we increase the threshold, the number of patterns decreases. This is a common pattern: when the threshold is lower, we tend to have more patterns that meet the criteria, while with a higher threshold, fewer patterns qualify.

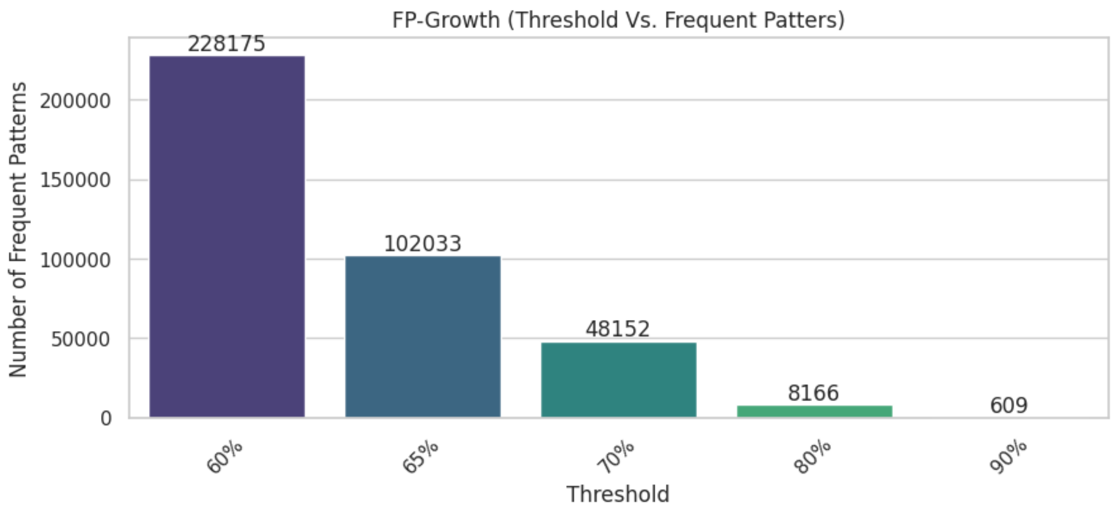

Figure 1: a graph showing number of frequent patterns for each threshold

When we compare the results with Apriori, we see that they produce almost the same number of frequent patterns when the threshold is set exactly the same. In simpler terms, the choice of algorithm doesn't seem to make much of a difference in terms of finding these frequent patterns.

To better understand which algorithm is more efficient, we are comparing the time it takes for two different algorithms. In Figure 2, you can see a graphical representation of the time it takes for both the FP Growth and Apriori algorithms. This will help us determine which one performs better in terms of speed and efficiency.

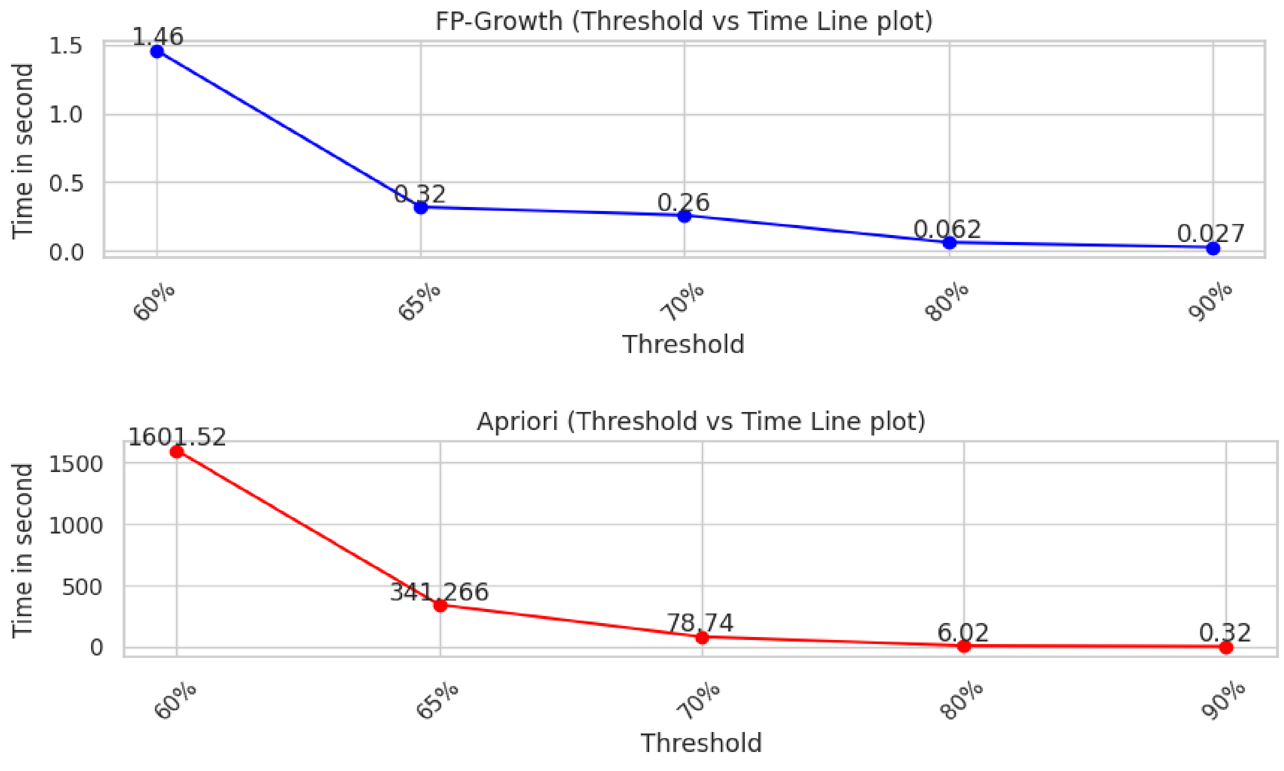

Figure 2: Threshold vs Time Line plot for FP-Growth (Top) and Apriori (bottom)

From the time plot, we can see that FP growth is taking less execution time in every threshold than Apriori algorithm. The FP growth is almost 1000 times faster in 60% threshold. We can also see that for less threshold, both algorithms are taking more time because it generates more patterns which causes extra times.

Dataset: Mushroom

We have run our implemented FP-Growth and Apriori algorithm on the Mushroom dataset. For different threshold, we got different number of frequent patterns. The table 2 and figure 3 showing number of frequent patterns for different thresholds of FP Growth.

Table 2: number of frequent patterns for each threshold for chess data

Threshold	Number of frequent patterns
60%	51
65%	39
70%	31
80%	23
90%	9

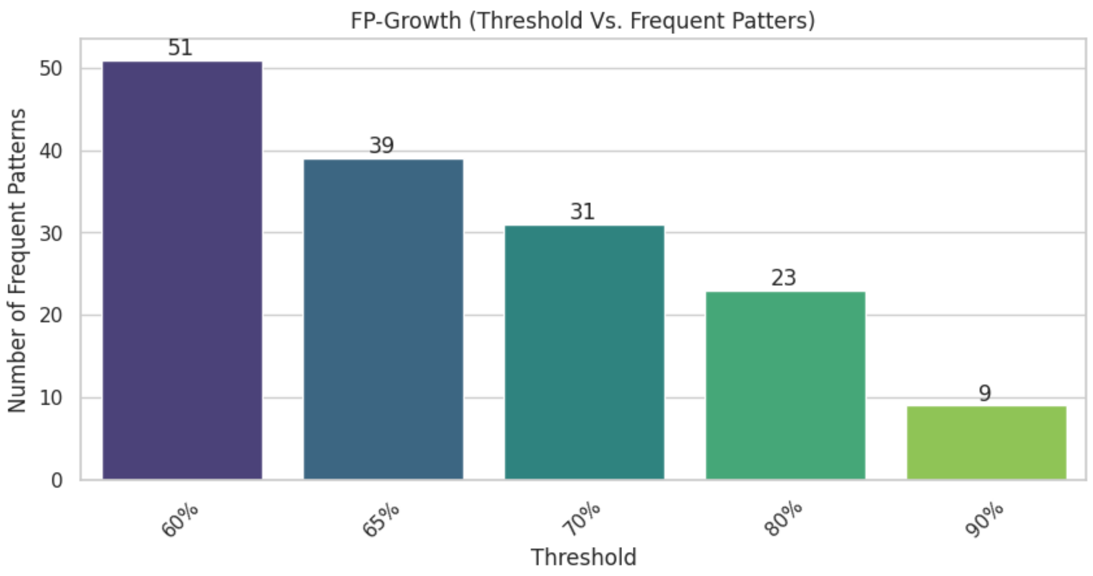

Figure 3: a graph showing number of frequent patterns for each threshold

We can also see the number of patterns is decreasing as the threshold is increasing. As we know, when there is lower threshold, patters become more frequent than the higher threshold. For Apriori algorithm, the numbers of frequent patters are almost same for exact same threshold. So, there is no difference in the frequent patterns between the two algorithms.

For this reason, we are showing the time comparison between two algorithms to see which one is better. The figure 4 showing the Time plot of the FP growth and Apriori algorithm.

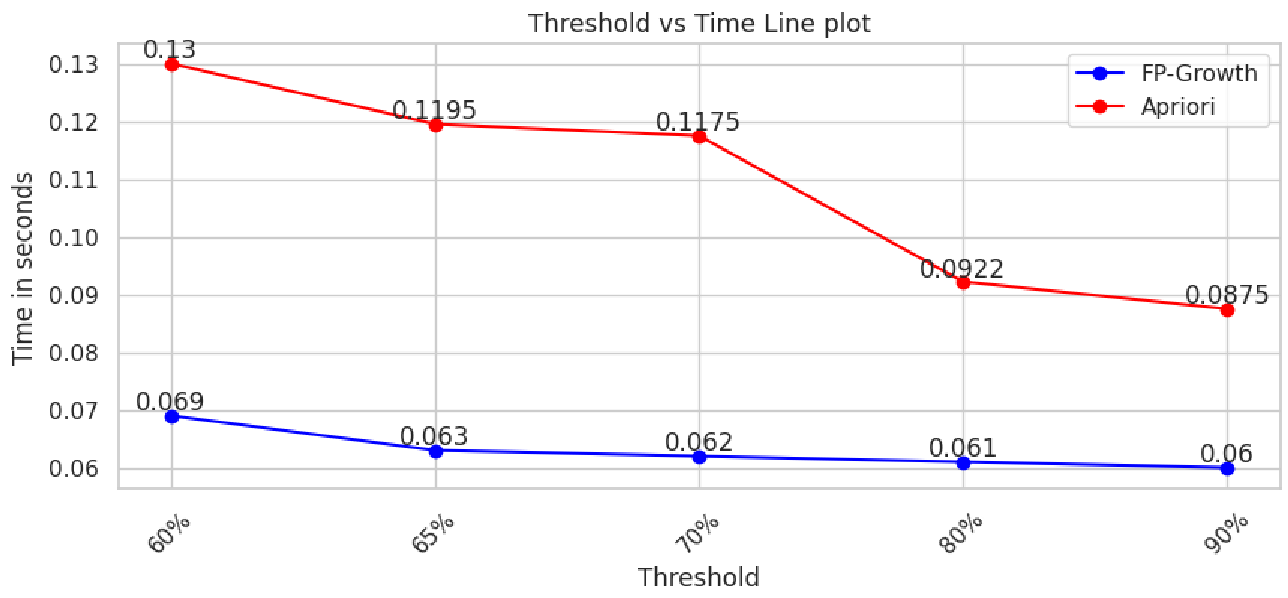

Figure 4: Threshold vs Time Line plot for FP-Growth and Apriori

From the time plot, we can see that FP growth is also taking less execution time in every threshold than Apriori algorithm.

Discussion:

The comparison of the bar charts and line charts reveals the following differences between the two datasets considering both the FP-Growth and Apriori algorithms:

Number of Frequent Patterns: The bar charts illustrate the disparity in the number of frequent patterns between the Chess and Mushroom datasets. The Chess dataset generally yields a higher number of frequent patterns across various thresholds compared to the Mushroom dataset. This suggests that the Chess dataset contains more potential associations among items.

Threshold Impact: Both bar charts of both algorithms demonstrate that increasing the threshold leads to a decrease in the number of frequent patterns. However, the rate of decline differs between the two datasets. The mushroom dataset exhibits a more gradual reduction, indicating a higher number of potential associations. On the other hand, the chess dataset experiences a sharper decline, suggesting a more selective association among items.

Computational Time: The line charts depict the variation in computational time for frequent pattern mining. The chess dataset generally requires more time to mine frequent patterns compared to the mushroom dataset in both cases of Apriori and FP-growth. This disparity may be influenced by factors such as dataset size, complexity, or the nature of the data. Again, for Apriori for both chess and mushroom dataset it is taking much more time to time than the FP-Growth.

Algorithm Efficiency: Despite the differences in the number of frequent patterns and computational time, both line charts demonstrate the efficiency of the FP-Growth algorithm and Apriori. The FP-Growth algorithm consistently exhibits reduced execution time as the threshold increases, indicating its scalability and effectiveness for frequent pattern mining. It is clearly shown in the line charts that FP-Growth is taking much less time than the Apriori, So, FP-Growth is much more efficient than the Apriori.

Overall, the analysis of the bar charts and line charts highlights the distinctions between the mushroom and chess datasets for both Apriori and FP-Growth in terms of the number of frequent patterns, threshold impact, and computational time. These insights emphasize the significance of dataset characteristics and threshold selection in mining frequent patterns and showcase the efficiency of the FP-Growth algorithm in handling diverse datasets. In a nutshell we can say that as the size of the chess dataset is large, chess dataset has more frequent itemset in this case regrading.

Limitations:

Memory Usage: FP-Growth can consume a significant amount of memory for large datasets, especially when constructing the FP-tree and header table. This can limit its applicability to datasets that cannot fit into memory.

Fixed Minimum support Threshold: The code uses a fixed minimum support threshold (minSupport) throughout the entire process. In practice, users may require the flexibility to adjust this threshold during or after the mining process.

Limited Data Format Support: The code assumes a specific format for input data (space-separated items). It may not handle other common data formats or data preprocessing steps.

Suggestions:

Lack of User-Friendly Interface: This code is more of a script than a user-friendly application. In practice, users may prefer tools with graphical interfaces or command-line options for better usability.

Performance Optimization: While the code works correctly, there may be opportunities to optimize it further for better performance, especially when dealing with large datasets.

Efficiency and Scalability: Although the FP-Growth algorithm is generally efficient for frequent pattern mining, the code may not be optimized for large or complex datasets. The code does not include any optimization techniques like pruning strategies or parallel processing, which could improve its efficiency and scalability for handling larger datasets.

Conclusion:

In summary, FP-growth is usually seen as being better than Apriori for finding frequent groups of items. This is because it uses a more efficient way of organizing data (the FP-tree) to make the calculations faster and easier. It usually works better than Apriori in terms of how fast it is and how much memory it uses. This makes it the top choice for association rule mining in most situations. In the chess dataset, FP-Growth is usually considered superior to Apriori. FP-Growth is a method that helps find frequent patterns and connections in chess data, using less memory and being faster. Because it can handle big sets of information and make it easier to work with, it is a popular choice for analyzing chess information and finding strategic ideas. Both the FP-growth and Apriori algorithms are very important for the mushroom dataset because they can effectively find commonly occurring patterns and connections between different characteristics of mushrooms. However, FP-Growth is often thought to be a better option. FP-Growth is a method that finds common patterns in mushrooms. It is faster and uses less memory than another method called Apriori. It helps people tell which mushrooms are safe to eat and which are not by finding common features and promoting safe food practices. These algorithms are extremely important tools for analyzing data, categorizing information, and making well-informed decisions when working with the mushroom dataset.