# Comparison of 'Apriori Algorithm' vs 'IIMFI' for Frequent Itemset Mining and Optimization Strategies

Syed Kazim Haider
*Faculty of Computer Science & Engg.*
*GIK Institute of Engg. Sciences & Tech.*
Topi, Khyber Pakhtunkhwa, Pakistan.
u2021624@giki.ed.pk

Atesam Abdullah
*Faculty of Computer Science & Engg.*
*GIK Institute of Engg. Sciences & Tech.*
Topi, Khyber Pakhtunkhwa, Pakistan.
u2021114@giki.edu.pk

***Abstract*—Frequent Itemset Mining (FIM) is an important foundation in data mining it is necessary to derive patterns and learn association rules from transactional data. In this study, we evaluate performance of the classical Apriori algorithm, and specifically its computational limits. The running time of Apriori was optimized in order to make Apriori more effective in large-scale datasets. In addition, the efficiency and scalability of the contemporary algorithm, An Incremental Interesting Maximal Frequent Itemset Mining Based on FP-Growth is evaluated. The proposed optimized Apriori, which is an advantageous version of the most popular algorithm for discovering frequent patterns, was evaluated experimentally on datasets, comparing to the existing algorithm based on FP-Growth, in terms of execution time, memory usage and scalability. The results show how the performance of Apriori can be improved, how the contemporary method compares in terms of performance to the classical methods, and how classical and modern FIM techniques can trade off one another. Research directions that remove potential limitations of the presented FIM methodologies are discussed to foster advancements in the efficient FIM methodologies. .**

***Index Terms*—Frequent Itemset Mining (FIM), Apriori Algorithm, FP-Growth Algorithm, Maximal Frequent Itemsets**

## I. Introduction

Frequent Itemset Mining (FIM), is a basic type of data mining that extracts recurring item sets within the huge transactional datasets. This process forms the basis of numerous analytical endeavours to produce association rules that shed light on patterns inherent in data. For example, in retail, involves the identification of market basket by FIM for the study of customers purchasing habits so as to enhance the position of products. In healthcare, it can identify relations inside clinical data, contributing to more accurate diagnosis and treatment advising. Also, FIM is used in fraud detection, web usage mining and other areas where it is imperative to find associations between the elements.

Graphic is useful, yet the calculation is quite intensive, especially when the size of the data and its complexity increase. The main algorithm of FIM, the classical Aprior that most people associate with the algorithm, is an example of systematic methods of frequent itemsets, candidate generation. However, its performance reduces notably due to high computational cost of the continual generation and testing of candidates when working with sparse or dense data sets.

To mitigate such issues, researcher have proposed modern techniques that include Incremental Interesting Maximal Frequent Itemset Mining Based on FP-Growth. This modern approach of data mining uses tree based structures to facilitate the mining in particular on dynamic data sets where updates are frequent.

This work was motivated by the fact that while the fundamental analytical tools of classical approach have been considerably improved in terms of computational accuracy, their large-scale applicability remains a challenge. This project aims to: 1. Check out how the classical Apriori algorithm works and discover the shortcomings of that method. 2. Different optimization techniques should be employed to help reduce the running time of Apriori. 3. Compare these results against a more modern FP-Growth based methodology and then consider the merits and drawbacks of each as well as their applicability to real-world problems.

As such, this paper aims to present experimental results for analyzing the effectiveness of FIM techniques and their improvements as well as future development direction for making these algorithms more flexible and high performance.

## II. Literature Review

### A. Apriori

Apriori algorithm is candidate generation and pruning based algorithm used for frequent itemset identification using an iterative process. Candidate k-itemsets are formed by k-1 itemsets generated from the previous iteration by trimming 1 itemset if its subset is not frequent. Each candidate k-itemset is supported by scanning over the entire database to count the number of transactions having all items of the candidate, with one scan for each longest itemset. We compare the support of candidates to the minimum threshold, retaining only those exceeding it, where patterns are determined by high frequencies. The algorithm terminates when no new high frequency patterns are generated in the process, and the process repeats by increasing k. This structured approach shows the exploitation of pruning, database scanning and iterative refinement for efficient mining of frequent itemsets. [1]

Apriori is a strong algorithm to the extent of having the capacity to separate helpful designs and findings from preparations and is consequently a valuable instrument in information mining. That said, it highly excels in analyzing complex relationships like analyzing the connections between multiple traits for example in the contexts of basketball penalties and techincal actions. The advantage of the capability of the algorithm to uncover hidden relationships based on itemsets and association rules enables the algorithm to derive association rules by examining itemsets. On the top of that Apriori algorithm further enhanced versions make it more efficient and effective which allows us to find stronger connections and technical moves as is evident in the basketball game analysis. With these improvements, the algorithm becomes practical and impactful and yields insights that serve coaches, players, and analysts to make informed technical and tactical strategies. [2].

Although Apriori is a landmark frequent itemset mining algorithm by its simplicity and clarity, it has a number of serious limitations. Repeated scanning of the transaction database for each candidate set causes high I/O cost and its exponential growth of candidate set size brings out computational and memory problems. Setting a minimum support threshold entirely on a minimum support threshold can lead to elimination of patterns that have real meaning and creation of too many irrelevant rules. The algorithm also fails to adopt to multi-dimensional or multi-level applications and thus the only way to approach this algorithm becomes major modifications or different approaches for real world applicability. In the presence of these challenges, there is a need for better and scalable frequent itemset mining techniques. [3]

## B. Hash-based Apriori

An enhanced version of the Apriori algorithm, this proposed approach is optimized by hash based techniques, and enhanced data structures to achieve better performance over the traditional Apriori algorithm. In contrast to the normal technique where each candidate itemset is scanned again and again in the database transaction space, the approach used here exploits an inverted index that leads to minimal redundant database scans. Support counting is accelerated by scanning the precomputed transactions, and not all transactions, that correspond to each item. Hash maps are used in the candidate generation step to efficiently prune excessive itemsets with probability such that the candidates to be generated are more likely to be frequent. The approach generated only candidate itemsets when necessary, and only itemsets whose subsets are frequent are considered, thus reducing the overall search space.

The algorithm is made orders of magnitude faster by these optimizations, especially for bigger datasets and higher order itemsets. The approach reduces computational and memory over head present in typical Apriori implementations through minimizing the number of database scans while focusing the candidate generation on promising sets. As a result, regarding I/O operations, they are a major bottleneck in the original Apriori algorithm. By limiting the maximum size of itemsets to 4 (a user defined parameter), the maximum amount by which itemsets can grow exponentially without increasing the computational burden of the algorithm is reduced as well. All in all, these optimizations make the approach more scalable in a way that allows to run faster when there are many transactions and candidate itemsets.

A hashing technique for generating efficient 2-itemsets and reducing the number of candidate itemsets is used to make the hash based Apriori algorithm more efficient. The candidate 2-itemsets are mapped from one hash table with fixed size buckets to another hash table with fixed size buckets during the first pass through the transaction database, and the count for each bucket is increased whenever itemsets are hashed on the same bucket. When the count for a bucket meets the minimum support threshold, the corresponding itemset is kept; otherwise, it is discarded. In this method we limit the number of candidate itemset and so it speeds up the algoirthm to find frequent itemset specially with big dataset. [4].

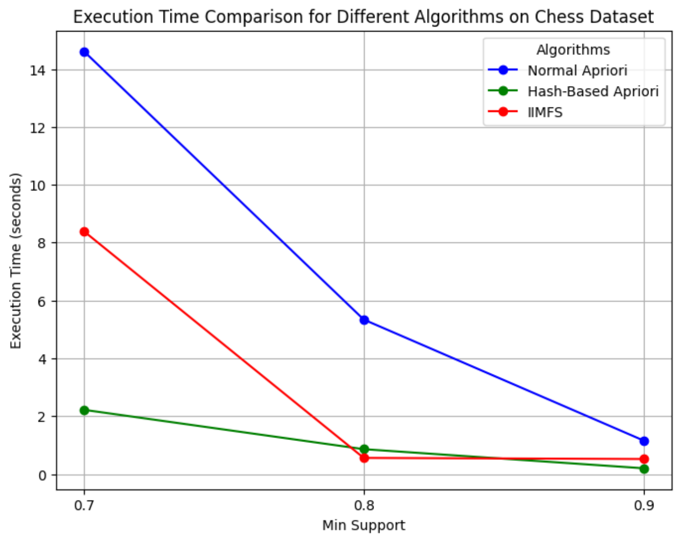

Fig. 1. Line Graph comparing the execution times of the Apriori method across different datasets with varying support thresholds. The execution times are plotted on the y-axis, while the x-axis Min support .

## C. Incremental Interesting Maximal Frequent Itemset Mining (IIMFI)

The Incremental Interesting Maximal Frequent Itemset (IIMFI) mining algorithm enhances the FP-Max approach to handle dynamic and incremental data efficiently. This framework incorporates subjective measures, such as novelty metrics (NM), during the mining process to focus on user-relevant patterns and reduce search complexity.

**Key Features**

1) **Incremental Nature**: IIMFI supports mining from dynamic databases by maintaining and updating previously discovered itemsets without requiring complete re-

| Dataset Name | Method Used | Min Support | 1st Size | 2nd Size | 3rd Size | 4th Size | Exec Time (sec) | Total Frequent Itemsets |
|---|---|---|---|---|---|---|---|---|
| Connect.txt | Apriori | 0.7 | 34 | 513 | 4678 | 29284 | 38.33 | 34509 |
| Connect.txt | Apriori | 0.8 | 30 | 394 | 3136 | 17253 | 24.87 | 20813 |
| Connect.txt | Apriori | 0.9 | 26 | 294 | 1970 | 8866 | 14.59 | 11156 |
| Connect.txt | Hash-based | 0.7 | 34 | 513 | 4678 | 29284 | 214.89 | 34509 |
| Connect.txt | Hash-based | 0.8 | 30 | 394 | 3136 | 17253 | 124.74 | 20813 |
| Connect.txt | Hash-based | 0.9 | 26 | 294 | 1970 | 8866 | 70.77 | 11156 |
| Connect.txt | IIMFI | 0.7 | 34 | 513 | 4678 | 29284 | 3170 | 224 |
| Connect.txt | IIMFI | 0.8 | 30 | 394 | 3136 | 17253 | 1456 | 27 |
| Connect.txt | IIMFI | 0.9 | 26 | 294 | 1970 | 8866 | 572 | 27 |
| Chess.txt | Apriori | 0.7 | 24 | 238 | 1237 | 3857 | 2.22 | 5356 |
| Chess.txt | Apriori | 0.8 | 19 | 141 | 566 | 1383 | 0.86 | 2109 |
| Chess.txt | Apriori | 0.9 | 13 | 68 | 167 | 203 | 0.20 | 451 |
| Chess.txt | Hash-based | 0.7 | 24 | 238 | 1237 | 3857 | 14.61 | 5356 |
| Chess.txt | Hash-based | 0.8 | 19 | 141 | 566 | 1383 | 5.34 | 2109 |
| Chess.txt | Hash-based | 0.9 | 13 | 68 | 167 | 203 | 1.16 | 451 |
| Chess.txt | IIMFI | 0.7 | 24 | 238 | 1237 | 3857 | 8.39 | 11463 |
| Chess.txt | IIMFI | 0.8 | 19 | 141 | 566 | 1383 | 0.56 | 891 |
| Chess.txt | IIMFI | 0.9 | 13 | 68 | 167 | 203 | 0.52 | 891 |
| Accident.txt | Apriori | 0.7 | 16 | 74 | 165 | 202 | 0.64 | 457 |
| Accident.txt | Apriori | 0.8 | 10 | 37 | 62 | 55 | 0.32 | 164 |
| Accident.txt | Apriori | 0.9 | 6 | 13 | 13 | 6 | 0.19 | 38 |
| Accident.txt | Hash-based | 0.7 | 16 | 74 | 165 | 202 | 4.38 | 457 |
| Accident.txt | Hash-based | 0.8 | 10 | 37 | 62 | 55 | 2.85 | 164 |
| Accident.txt | Hash-based | 0.9 | 6 | 13 | 13 | 6 | 1.17 | 38 |
| Accident.txt | IIMFI | 0.7 | 16 | 74 | 165 | 202 | 0.64 | 457 |
| Accident.txt | IIMFI | 0.8 | 10 | 37 | 62 | 55 | 0.32 | 164 |
| Accident.txt | IIMFI | 0.9 | 6 | 13 | 13 | 6 | 0.19 | 38 |

TABLE I
FREQUENT ITEMSET MINING RESULTS ACROSS DATASETS WITH IIMFI.

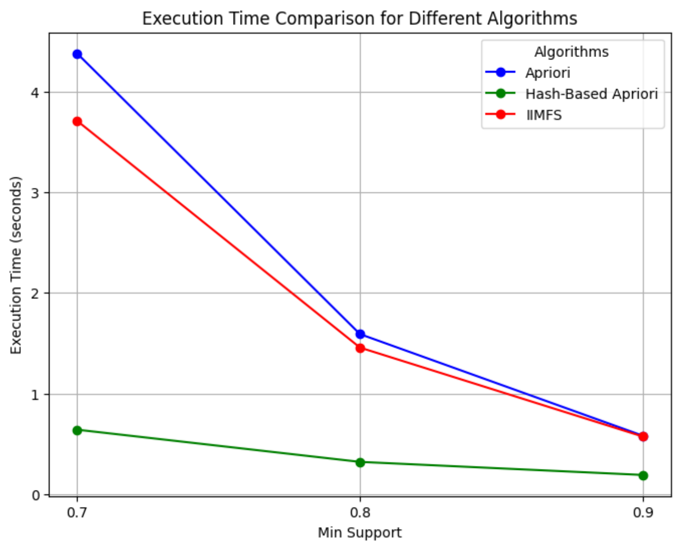

Fig. 2. Line Graph comparing the execution times of the Apriori method across different datasets with varying support thresholds. The execution times are plotted on the y-axis, while the x-axis Min support .

mining. It dynamically adjusts thresholds and supports based on newly added data.

2) **Tree-Based Structure**: The algorithm builds upon FP-Growth and FP-Max by constructing and updating an FP-tree-like structure, optimized to include previously discovered patterns (PD-IMFIs) for rapid access.
3) **Dynamic Pruning**: Using the NM measure, IIMFI eliminates uninteresting patterns, reducing both the search space and the number of association rules generated.

**Phased Execution**

1) **Phase 1**: Updates incremental support values and constructs PD-IMFIs.
2) **Phase 2**: Constructs the tree structure from both the incremental dataset and previous itemsets.
3) **Phase 3**: Mines new maximal frequent itemsets using FP-Max.
4) **Phase 4**: Applies dynamic pruning based on NM thresholds to retain only interesting patterns.
5) **Phase 5**: Generates association rules meeting user-defined confidence thresholds.

This structured approach makes IIMFI particularly suited for evolving datasets, enabling it to maintain efficiency and relevance in real-time analysis [5].

The code for this algorithm is available at: https://github.com/alhussein1977/IIMFIs-Algorithm

## III. DATASETS

The experimental evaluation of all three algorithms was conducted using publicly available datasets, specifically the Accidents, Chess, and Connect datasets from the FIMI repository http://fimi.uantwerpen.be/data/.

- **Accidents**: This dataset contains anonymized traffic accident data, providing a real-world scenario for frequent itemset mining. Detailed information about this dataset can be found in the associated paper [6].
- **Chess**: A dense dataset representing chess endgame positions, commonly used to evaluate the performance of pattern mining algorithms.

- **Connect**: Also known as the Connect-4 dataset, it comprises all legal 8-ply positions in the game of Connect-4, offering a large and complex dataset for analysis.

## IV. Experimental Setup

In this paper, we study how the algorithms behaves in different datasets and with varying minimum support thresholds and define an experimental framework for evaluating the algorithm's performance. The framework includes three primary datasets: Real-world data on which the algorithm is tested, represented by 'Connect.txt', 'Chess.txt' and 'Accident.txt'. From text files containing transactions with items in each line in spaces, the datasets are read and processed. The dataset size is limited to a maximum of 10,000 rows for each dataset to be manageable for performance analysis.We have also reduce the itemset size to only 4

In this experiment, execution time recorded on each dataset under different minimum support values (0.7, 0.8, and 0.9) are used as performance metrics. It also shows the effect of varying support thresholds on the time required for the algorithm to identify frequent itemsets. Also tracked is the total number of frequent itemsets discovered and their distribution across different itemset sizes (1 itemsets, 2 itemsets, 3 itemsets, and 4 itemsets). Through these metrics we can evaluate what is the effectiveness and efficiency of the algorithm and check it for scalability and computational resource usage with the different support values.

### A. Apriori Implementation

It uses Python built-in libraries like 'defaultdict' for fast storage and retrieval of data and 'frozenset' for fast operations on candidate itemsets and support count. The algorithm then proceeds by calculating the support for 1-itemsets and iteratively generating candidate itemsets of size 2, 3, and 4. The dictionary stores the number of frequent itemsets found for each size and their corresponding support. It also measures the total execution time required to process each dataset under varying conditions to find out how the algorithm performs as data sizes grow and support levels change.

### B. Hash Based Apriori Implementation

Hash based structures are used for the implementation of the optimized Apriori algorithm, fram which frozenset is used where appropriate for set operations. However it's in the candidate generation and support counting side that the optimized one makes main differences. The generate candidates function relies on a hash based technique of generating candidates faster than the traditional Apriori by mapping itemsets to hash buckets. Furthermore, with an inverted index that stores occurrences of items, support counting is accelerated and candidate itemset checking takes much less time.

### C. IIMFI Algorithm Implementation

The Incremental Interesting Maximal Frequent Itemset (IIMFI) algorithm is implemented in Java, building upon the functionalities provided by the SPMF open-source data mining library [7]. SPMF specializes in pattern mining and offers a wide range of algorithms for tasks such as association rule mining, itemset mining, and sequential pattern mining. By leveraging SPMF's efficient data structures and algorithms, IIMFI efficiently manages dynamic datasets and integrates subjective interestingness measures during the mining process. The implementation is available on GitHub for further reference and experimentation [**?**].

The tests measured execution time and memory usage as the dataset size increased and new data increments were introduced. Support thresholds [0.7, 0.8, 0.9] were applied to observe the algorithm's adaptability to different levels of data frequency. The results demonstrated that IIMFI effectively manages incremental data, maintaining high performance even as the dataset evolves [5].

## V. Results

For the three datasets Connect.txt, Chess.txt and Accident.txt — both the traditional Apriori algorithm and hash based Apriori algorithm result in large differences in execution time and the number of frequent itemsets it discover, which provides hash a better performance on the efficiency front.

The hash based approach is significantly faster than a traditional approach in Apriori based on the Connect.txt dataset. The traditional Apriori takes 38.33 seconds at a minimum support of 0.7, while the hash version takes 214.89 seconds. The execution time of the traditional method decreases dramatically with increasing the support range, which decreases to 24.87 s for 0.8 support and 14.59 s for 0.9 support. It is worth noting that the hash based method has more stable (but long) execution times at 124.74 seconds for 0.8 support and 70.77 seconds for 0.9 support. Hashing pattern shows that although it may not always lead to minimum total execution time, it does reduce the total processing of large datasets and focus on the procurable itemsets.

The same trend is observed in the number of frequent itemsets discovered by the two methods. On Connect.txt for at least 0.7 of support, the traditional Apriori algorithm finds 34 itemsets of size 1, 513 itemsets of size 2, 4678 itemsets of size 3, as well as 29284 itemsets of size 4, with a great number of these itemsets. It is expected that as the support threshold increases, the number of frequent itemsets decreases. With the decrease at higher supports, the hash based approach discovers 34 itemsets of size 1, 513 itemsets of size 2, and 4678 itemsets of size 3. The fact that both methods manage to find almost the same number of itemsets but hashing does it in much less time by efficiently pruning non promising candidates during the processing is what is being highlighted here.

For Chess.txt a smaller dataset the difference is even more stark. A comparison of the traditional Apriori, and the hash based version, shows that the latter requires 14.61 seconds to finish the task, which is significantly more time than 2.22 seconds needed by the traditional Apriori for 0.7 support. Both methods become more efficient as the support grows, with the hash based version 5.34 secs at 0.8 and 1.16 secs at 0.9 remaining relatively slow. However, the number of frequent

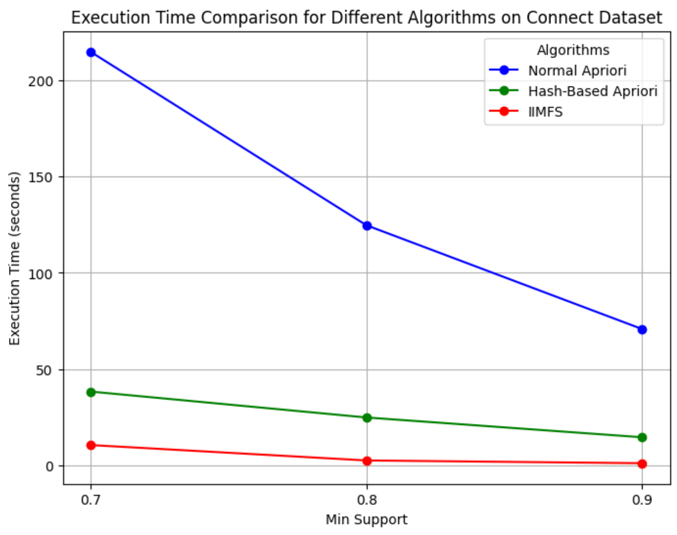

Fig. 3. Line Graph comparing the execution times of the Apriori methods Hash based apriori and IIMFS across Connect datasets with varying support thresholds. The execution times are plotted on the y-axis, while the x-axis min support .

itemsets generated by both methods remains in agreement with expectation, decreasing with increasing support values.

In the end, even more so for the algorithms in the Accident.txt dataset. The traditional Apriori method completes in 0.64 seconds at a minimum support of 0.7, finding 457 frequent itemsets. Compared to the hash based method, hash based one took 4.38 seconds to complete the task. We confirm the expected behavior by showing that both methods decrease the total number of frequent itemsets as the support grows, as 164 itemsets were found for 0.8 support and 38 itemsets for 0.9 support. The table is shown for the comparision in table 1.

These results show that the hash based approach is better in cases when the dataset is big and can quickly be reduced by means of hash buckets. The performance gain is less pronounced for smaller datasets like Chess.txt and the traditional Apriori method may be still trying for smaller datasets because of less computational overhead. The main strength of the hash-based Apriori is that it is able to accelerate the generation and pruning process of candidate itemsets in these large datasets, while traditional Apriori still performs competitively in small and manageable ones.

## References

[1] S. Panjaitan, Sulindawaty, M. Amin, S. Lindawati, R. Watrianthos, H. T. Sihotang, and B. Sinaga, "Implementation of apriori algorithm for analysis of consumer purchase patterns," *Journal of Physics: Conference Series*, vol. 1255, no. 1, p. 012057, aug 2019. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1255/1/012057

[2] Y. Zakur and L. Flaih, "Apriori algorithm and hybrid apriori algorithm in the data mining: A comprehensive review," in *E3S Web of Conferences*, vol. 448. EDP Sciences, 2023, p. 02021.

[3] Y. Liu, "Study on application of apriori algorithm in data mining," in *2010 Second International Conference on Computer Modeling and Simulation*, vol. 3, 2010, pp. 111–114.

[4] A. J. Doshi and B. Joshi, "Comparative analysis of apriori and apriori with hashing algorithm," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, no. 1, pp. 976–979, 2018.

[5] H. A. Alsaeedi and A. S. Alhegami, "An incremental interesting maximal frequent itemset mining based on fp-growth algorithm," *Complexity*, vol. 2022, no. 1, p. 1942517, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1942517

[6] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof, "'profiling high frequency accident locations using association rules," 2003, p. 18pp.

[7] P. Fournier Viger, J. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z.-H. Deng, and H. Lam, "The spmf open-source data mining library version 2," vol. 9853, 09 2016, pp. 36–40.