Empirical Comparison of Apriori and FP-Tree Algorithms

COL761 Assignment 1

1 Introduction

Frequent itemset mining is a fundamental task in data mining, with Apriori and FP-Tree (FP-Growth) being two widely used algorithms. In this assignment, we conduct an empirical comparison of these algorithms using the provided libraries on the `webdocs.dat` dataset and on a synthetically generated dataset. The goal is to analyze their runtime behavior under varying minimum support thresholds.

2 Experimental Setup

We utilize the implementations provided by Christian Borgelt for both Apriori and FP-Tree algorithms. The experiments were conducted on the `webdocs.dat` dataset for Task 1 and on a generated dataset of approximately 15,000 transactions for Task 2.

For both datasets, runtimes were measured at the following minimum support thresholds:

$$5\%,\ 10\%,\ 25\%,\ 50\%,\ 90\%$$

The runtime was measured using the `/usr/bin/time -v` command, and the elapsed wall-clock time was recorded. Each algorithm was executed independently for each support threshold.

2.1 Observations

- At **5% minimum support**, the Apriori algorithm exceeded the time limit of **3600 seconds** and was terminated. This indicates a severe scalability limitation of Apriori at low support thresholds.
- FP-Tree completed successfully at all support thresholds, including 5%, demonstrating its superior efficiency.
- As the support threshold increases, the runtime of both algorithms decreases significantly.
- At high support thresholds (50% and 90%), the runtime difference between Apriori and FP-Tree becomes negligible.

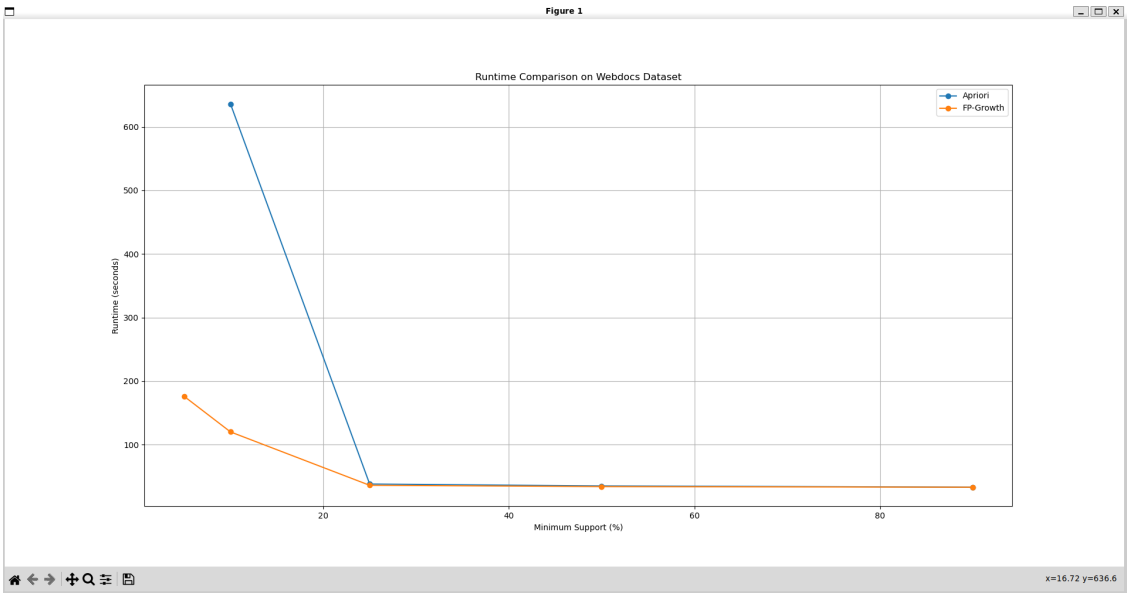

Figure 1: Runtime comparison of Apriori and FP-Tree on the webdocs.dat dataset across varying minimum support thresholds.

3 Results on Constructed Dataset (Task 2)

A synthetic transactional dataset with approximately 15,000 transactions was generated by sampling from a universal itemset. Items were drawn with a skewed frequency distribution consisting of frequent, medium-frequency, and rare items, ensuring realistic transaction diversity without duplicating identical transactions.

3.1 Observations

- Both algorithms complete quickly at higher support thresholds due to the reduced number of frequent itemsets.
- Apriori shows higher runtime than FP-Tree at lower supports, though the absolute runtime is smaller compared to the original dataset.
- The qualitative trend of Apriori being significantly slower at low supports is preserved, matching the behavior observed in the original dataset.

3.2 Observations

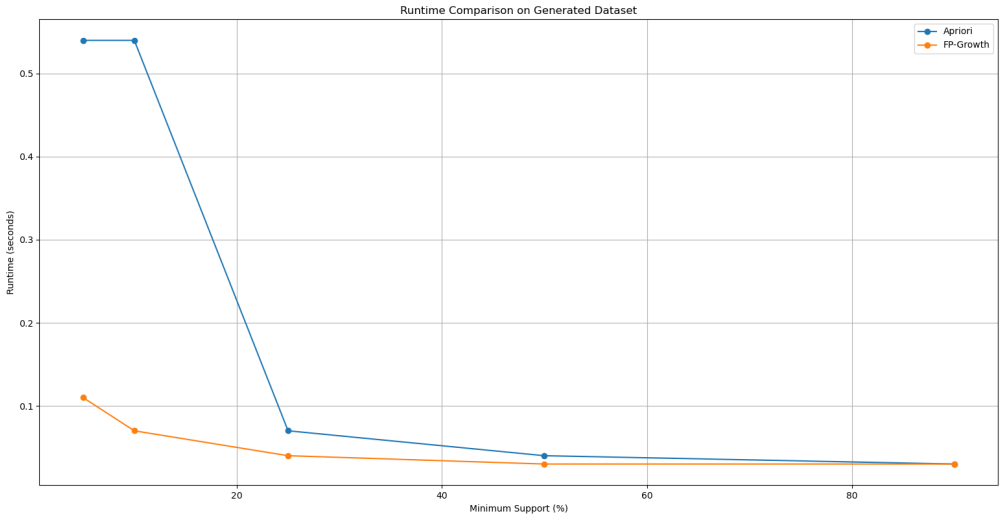

Figure 2: Runtime comparison of Apriori and FP-Tree on the constructed synthetic dataset.

4 Comparative Analysis

The key difference in runtime behavior between the two datasets arises from the **distribution of items across transactions**.

- The original `webdocs.dat` dataset exhibits a highly skewed item distribution with many infrequent items, leading to an exponential explosion in candidate generation for Apriori at low supports.
- FP-Tree avoids explicit candidate generation and instead compresses transactions into an FP-tree, making it more robust to such skewed distributions.
- In the constructed dataset, controlled item frequencies limit the number of candidate itemsets, resulting in lower runtimes while still preserving the qualitative performance gap between the algorithms.

4.1 Comparison Between Original and Constructed Datasets

A clear difference in the runtime behavior of Apriori and FP-Tree is observed when comparing the results on the original `webdocs.dat` dataset (Task 1) with those on the constructed synthetic dataset (Task 2). This difference can be attributed primarily to the distribution of items across transactions.

The original `webdocs.dat` dataset is characterized by a highly skewed item frequency distribution. A small subset of items appears very frequently across transactions, while a large number of items occur rarely. At low minimum support thresholds (notably 5%), this skew results in an extremely large number of candidate itemsets. Since Apriori explicitly generates and tests candidates at each level, the number of candidate combinations grows exponentially. As a result, Apriori exceeded the imposed time limit of 3600 seconds at 5% minimum support and was terminated.

In contrast, the FP-Tree algorithm handles this skewed distribution more efficiently by compressing transactions into a prefix tree and avoiding explicit candidate generation. This allows FP-Tree to complete execution even at low support thresholds, with significantly lower runtime compared to Apriori.

For the constructed dataset used in Task 2, the item distribution was deliberately controlled during transaction generation by sampling from frequent, medium-frequency, and rare item subsets. This design limits the growth of candidate itemsets while still maintaining realistic transaction diversity. Consequently, Apriori does not time out at 5% support on the constructed dataset, although it remains slower than FP-Tree.

At higher support thresholds (50% and 90%), both datasets show similar behavior: the number of frequent itemsets decreases sharply, leading to reduced runtimes and a narrowing performance gap between the two algorithms. This confirms that Apriori's inefficiency is most pronounced in dense, low-support regimes, while FP-Tree scales more gracefully across varying data distributions.

5 Conclusion

This empirical study demonstrates that FP-Tree consistently outperforms Apriori, particularly at low minimum support thresholds. Apriori's candidate generation strategy leads to prohibitively high runtimes, as evidenced by the timeout at 5% support on the original dataset. FP-Tree scales more effectively due to its compact data representation and recursive mining strategy. These results highlight the importance of algorithm selection based on dataset characteristics and support thresholds.

6 References

1. R. Agrawal and R. Srikant, *Fast Algorithms for Mining Association Rules*, Proc. VLDB, 1994.
2. J. Han, J. Pei, and Y. Yin, *Mining Frequent Patterns without Candidate Generation*, Proc. SIGMOD, 2000.
3. C. Borgelt, *Apriori — Frequent Item Set Mining Documentation*, `https://borgelt.net/doc/apriori/apriori.html`
4. C. Borgelt, *FP-Growth — Frequent Pattern Mining Documentation*, `https://borgelt.net/doc/fpgrowth/fpgrowth.html`
5. C. Borgelt, *Apriori Source Code*, `https://borgelt.net/src/apriori.zip`
6. C. Borgelt, *FP-Growth Source Code*, `https://borgelt.net/src/fpgrowth.zip`
7. FIMI Repository, *Webdocs Transactional Dataset*, `http://fimi.uantwerpen.be/data/webdocs.dat.gz`
8. FIMI Repository, *Webdocs Dataset Description*, `http://fimi.uantwerpen.be/data/webdocs.pdf`
9. GNU Project, *GNU Time Utility*, `https://www.gnu.org/software/time/`
10. Python Software Foundation, *Matplotlib Visualization Library*, `https://matplotlib.org/`