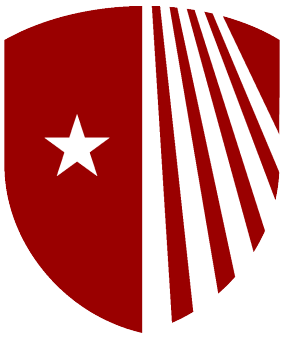
FP Growth Algorithm

ESE 589

Zahin Huq & Hualin Zheng

November 2019



Stony Brook University

Abstract

To extract the desired information and knowledge, we use data mining techniques to deal with large data stored in databases. There are various techniques to extract data. The association rule is one of the most efficient data mining techniques. We can use it to discover patterns from large datasets. The frequent pattern growth (FP-growth) algorithm is one of the most efficient algorithms to search and find the associative rules within a dataset. In this report, we build a program to implement the FP-growth algorithm. We also evaluate our program with several datasets from the UCI Machine-Learning benchmarks.

Contents

1 Introduction

Frequent pattern mining can be used with various types of datasets, such as relational tables or transactional databases, in order to determine the frequent itemsets and the associative rules that can be made using them. Frequent itemsets are based on a minimum support condition and measure data that appears simultaneously at higher frequency than other combinations. Once these frequent itemsets are found, associative rules may be derived from them. Associative rules are philosophical in nature; they contain an antecedent with one or more clauses and a consequent. For example, one such associative rule with a dataset regarding grocery sales may have an antecedent of buys apples and a consequent of buy oranges.

One example a frequent pattern mining method is using the Apriori algorithm. It is based on the Apriori condition where, in the case for frequent pattern mining, subsets of a frequent itemset must also be frequent. This allows for pruning after a database is scanned, eliminating the need to access every itemset and pass it through an algorithm. The database is scanned, where the data is pruned based on a minimum support condition. Then, the data that met the Apriori condition are joined in an itemset and the pruning process is applied again. This is a recursive algorithm that repeats until the frequent item sets with the highest specificity are found.

While the Apriori algorithm is an effective algorithm, another frequent pattern mining algorithm is the FP-growth algorithm. In this paper, we analyze the capabilities and performance of the FP-growth algorithm using several benchmark examples.

1.1 FP-Growth Algorithm

The diagrams below demonstrate the process of the FP-growth algorithm. First, the database is scanned and a FP tree is created. Then the tree is traversed and mined for frequent patterns.

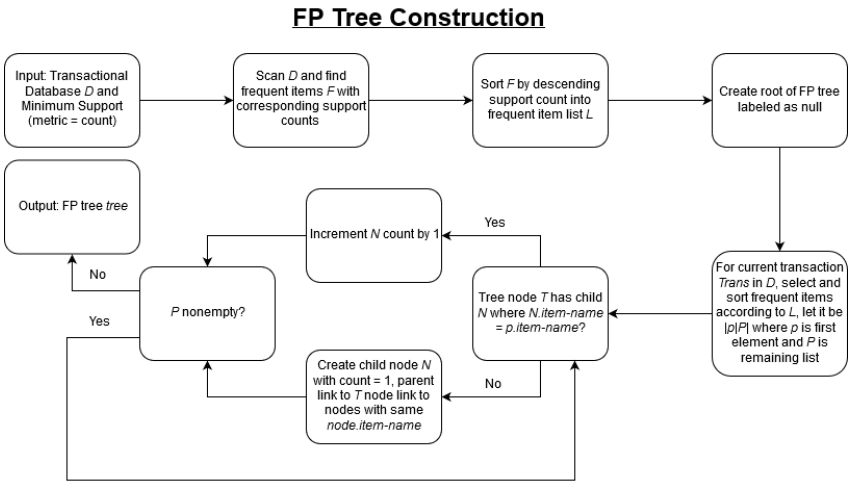


Figure 1: Process to generate the FP tree. It is a recursive process that repeats until the end of the list of frequent items gathered by scanning the dataset.

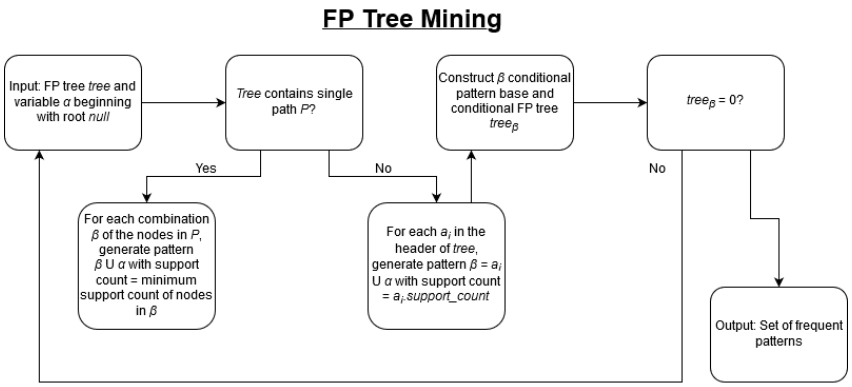


Figure 2: Process to mine the generate FP tree. Also recursively called until all frequent item sets with a given support condition are obtained.

Figures 1 and 2 show how the database needs to be accessed less than with the Apriori algorithm, which means less interactions with the external memory. The algorithm accesses the database to form the FP tree and then uses the generated tree to comb for the frequent pattern associations.

1.2 Preprocessing

Some datasets required some preprocessing in order to implement the FP-growth algorithm. Mainly a simple preprocessing technique was used, binning, in order to organize the data in certain small datasets such that they can be effectively used in conjunction with the FP-growth algorithm. Other data preprocessing techniques, such as correlative analysis, were not implemented in order to focus on the performance of the FP-growth algorithm with raw data.

2 Software Design

The algorithm was implemented in Java. We chose Java because of the differences in memory allocation and deallocation when compared to C++, making it particurly useful for data analytical purposes. Compared to the Apriori algorithm, the FP-growth algorithm only scans the dataset twice. The way we used to finding frequent itemset based on FP-growth is: First, build FP tree. Second, mine and search data based on the FP tree. The general steps to perform FP tree:

2.1 Data Structures

First, we build a structure called FpNode. It is used to construct the root nodes. It includes ID, children node, parent node, next point, and count. Next we build the Fptree structure, which is a kind of search tree. It is an ordered tree data structure that is used to store dynamic set or associative array where the keys are strings or numbers. All the children nodes have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are only associated with leaves and the inside nodes can correspond to keys of interest.

2.2 Software Organization

Structures and functions	Description	Inputs	Outputs
FpNode	The basic structure to record tree node.	nodes	leaves
FpTree	The higher structure to building Fp tree.	nodes	trees
growthFunc	Based on FP growth, recursively finding frequent items	Root, head node, id	Frequent items
addLeafToFrequent	Add leaf nodes to frequent set	Leaf, frequent set	Frequent set
discretePath	Determine if an fp-tree is a single path	Top head node, link	True or false
getConditionFpTree	Generate condition tree	path	Condition tree
combiPattern	Find all combinations on a single path and frequent items generate from ID	Path, idmark	Qualified frequent items
findFpTree	Build Fp-Tree structure	Tophaed, frequentlink	Fptree structure
orderedLink	Sort the ID in descending order based on the value of the frequentLink	Line, frequentLink	orderLink
getTop	Generate table. The key is equal to ID value. Descend ordering based on frequent value.	Table, frequntLink	topHead
Holder	Generate holder for conditional tree	Root, head	holder

Figure 3: Structures and Functions.

3 Implementation & Validation

For implementation, we use Visual Studio 2019 to run our program. We used two simple datasets to testing if our implementation works or not. The tests were run on a Lenovo Thinkpad W530 with 16 GB RAM and i7 Processor and a Lenovo Ideapad with 8 GB RAM and i7 Processor.

```
1   milk bread
2   bread diaper beer egg
3   milk diaper beer coke
4   bread milk diaper beer
5   bread milk diaper coke
```

Figure 4: Sample dataset 1.

```
C:\Users\Kyle Zheng\Documents\589-Project2\FPtree-master>java FpTree
Absolute support value: 3
Frequent items:
bread diaper    3
diaper  4
bread   4
diaper beer     3
diaper milk     3
```

Figure 5: Sample dataset 1 Result.

```
1   1   2   5
2   2   4
3   2   3
4   1   2   4
5   1   3
6   2   3
7   1   3
8   1   2   3   5
9   1   2   3
10
```

Figure 6: Sample dataset 2.

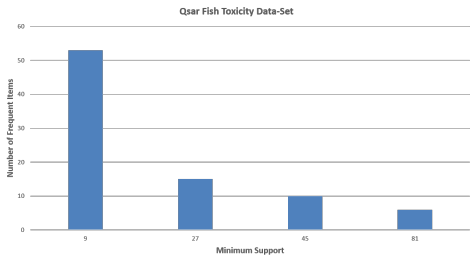
```
C:\Users\Kyle Zheng\Documents\589-Project2\FPtree-master>java FpTree
Absolute support value: 1
Frequent items:
Time Cost: 0
1       2       3       5       1
2       3       2
1       3       2
1       2       5       1
2       4       1
1       2       3       1
1       2       4       1
```

Figure 7: Sample dataset 2 Result.

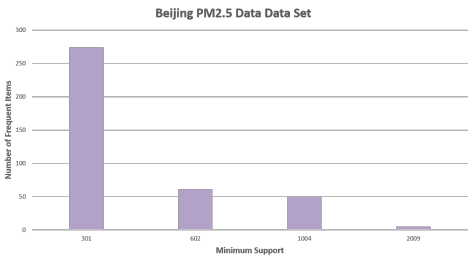
The results from these small sample datasets are meant to validate our code for the FP-growth algorithm. As seen in the figures above, the code accurately outputs the frequent itemsets based on the support threshold.

4 Experiments & Discussion

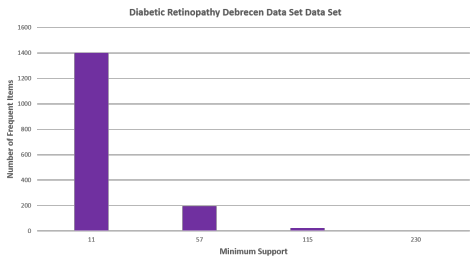
Several benchmarks were used in order to assess the accuracy of the code and analyze its FP-growth algorithm implementation performance.



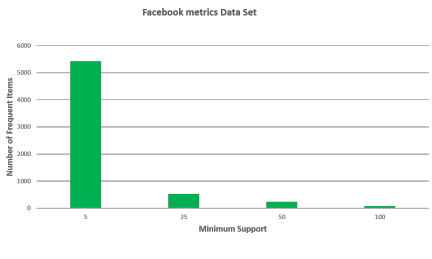
(a) QSAR fish toxicity dataset (medium).



(b) Beijing PM2.5 Data dataset (large).



(c) Diabetic Retinopathy Debrecen dataset (small).

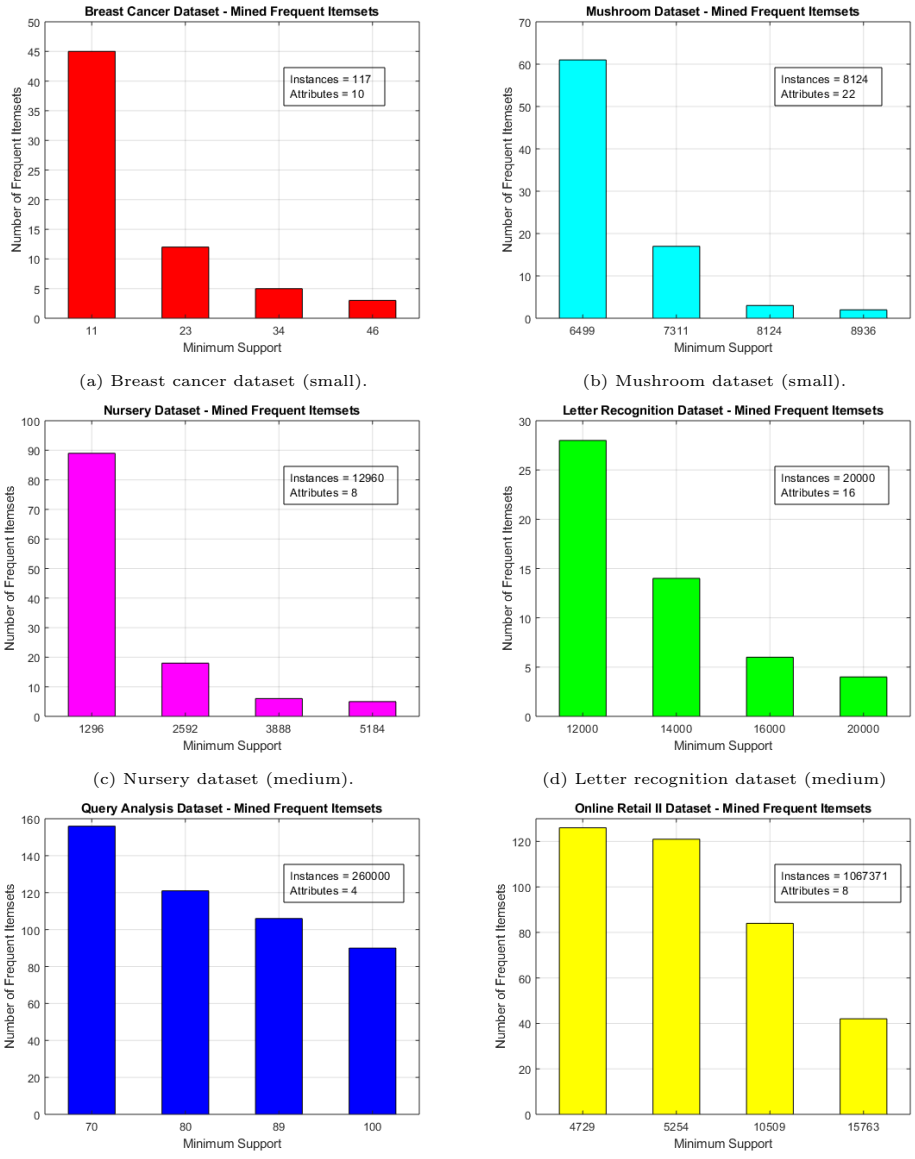


(d) Facebook metrics dataset (medium)

Figure 8: Mined frequent itemsets from several benchmark examples in order to further validate the code for the FP-growth aglorithm.

These benchmarks further validate the code implementation for our FP-growth algorithm. The consistency in the trends observed ensure that the code is precise in its output.

4.1 Support Threshold Dependence



(a) Breast cancer dataset (small).

(b) Mushroom dataset (small).

(c) Nursery dataset (medium).

(d) Letter recognition dataset (medium)

(e) Query analysis dataset (large).

(f) Online retail II dataset (large).

Figure 9: Mined frequent itemsets from various benchmark examples. Size of datasets vary to demonstrate the algorithms difference's in behavior and the nature of large data.

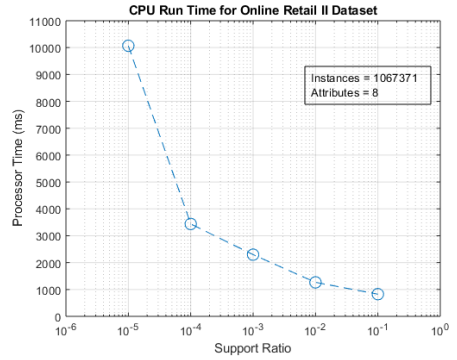
The algorithm is dependent upon the data size and the minimum support threshold. Adjusting these according to whatever the desired analytic metric highlights the customizability of the FP-growth algorithm. However, certain trends exist. The figure above shows how increase the support threshold decreases the amount of unique frequent itemsets that are mined. This is consistent intuitively, as a greater support means a higher threshold for the itemset count to surpass.

The minimum support variability arises from the dataset and problem variability. Depending on the parameters of a specific problem, the minimum support can be fine tuned to obtain unique

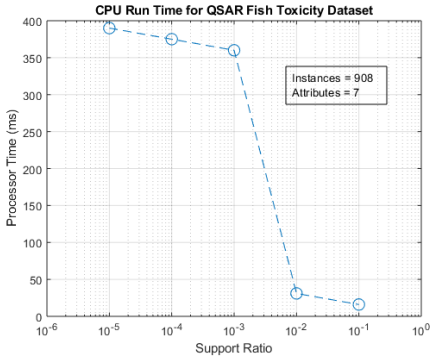
frequent itemsets. For large datasets this minimum support count has a much wider range of possible values as compared to small datasets. This presents a limitation with the algorithm, since the process of selecting a minimum support threshold cannot really be automated for a general purpose algorithm.

4.2 Processing Time

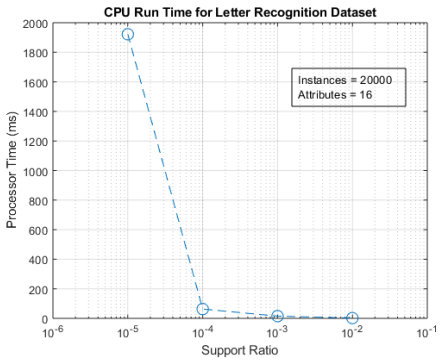
We hypothesize that with a smaller support, the larger the CPU time would be to execute the algorithm no matter the data size. For this experiment we use four datasets of varying sizes and compare the processor time across a logarithmic range of support rates. A logarithmic range was selected to demonstrate the CPU time over a wide range of support thresholds that are comparable no matter the data size, where the rate in our code is used to compute a support threshold depending on the size of the data.



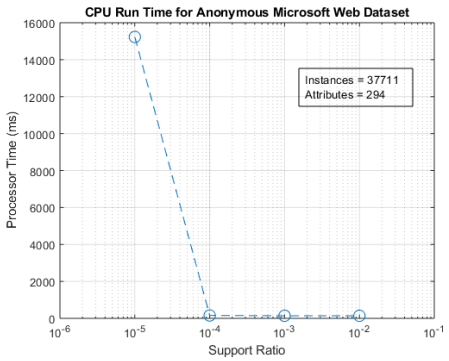
(a) Online retail II dataset processing time vs support threshold rate.



(b) QSAR fish toxicity dataset processing time vs support threshold rate.



(c) Letter recognition dataset processing time vs support threshold rate.



(d) Anonymous Microsoft web data dataset processing time vs support threshold rate.

Figure 10: CPU time for select datasets.

The figures show the nonlinear relationship between the processor time and the support threshold. However, a general trend of a significant decrease in time as the support threshold increases is present at certain regions. This observation is related to the trend about the number of unique frequent itemsets as the support threshold increases. With a lower support threshold, more data is stored as a frequent item and thus results in a longer execution time. However, further testing with much larger data sets and improved support threshold conditions as well as more test points are needed to determine whether or not a mathematical relationship exists between data size and FP-growth algorithm processing time. Results of an expansion of this test would be

useful in real world applications of the FP-growth algorithm, particularly in deciding a support threshold.

4.3 Limitations

Although the FP-growth algorithm is a powerful tool in mining frequent patterns, there are some limitations. Its efficiency decreases with larger datasets, where computing time compared to smaller datasets, as seen in figure 10, increases significantly. The FP-growth algorithm could be optimized further to incorporate a better method of handling larger data quantities, but our implementation shows it is best suited for smaller datasets. Another limitation of the FP-growth algorithm is the support threshold. It is highly variable and data dependent, and requires an expert on the data of interest in order to set. Our code cannot include an optimal method of selecting a support threshold depending on the data, where processing time, number of frequent patters, and frequent itemset size are important factors.

5 Conclusion

In this paper, we present our implementation of the FP-growth algorithm. The performance of the FP-growth algorithm shows that it is effective for mining both long and short frequent patterns' data, but more efficient for smaller datasets. Based on the experiments, the FP-growth algorithm has been proven that it is consistent. It can save both time and space when comparing to other frequent pattern algorithms such as the Apriori aglorithm. However, FP-growth algorithm also has problems. Once the dataset is too large or the support threshold is too small, the recursion depth will getting too high, and the efficiency will decrease significantly. In addition, the selection of a support threshold is too reliant on the dataset to be able to be automated. Overall the FP-growth algorithm presents a useful way to obtain frequent itemsets, in which the next step would be developing associative rules and possibly creating classifiers.