DATA ANALYTICS

ARM

Association Rule Mining

2019111019 - 2019111026

Question 1: FP Growth

Optimization Strategy

● **Bottom Up technique (with removal of redundant paths)**

○ We traverse through the header linked list of a particular item and then for each node we get its path to the root.

○ We have to traverse the itemsets in the opposite order of what the itemsets are ordered.

○ We have deleted the part of the path that becomes redundant after the traversal and rather push into the earlier node's conditional pattern base. That is, we have pushed right the branches that have been mined for a particular item to the remaining branch. This is expected to somewhat (minor) speed up the algorithm and the deletion of the redundant path will save up some space.

● **Top Down technique**

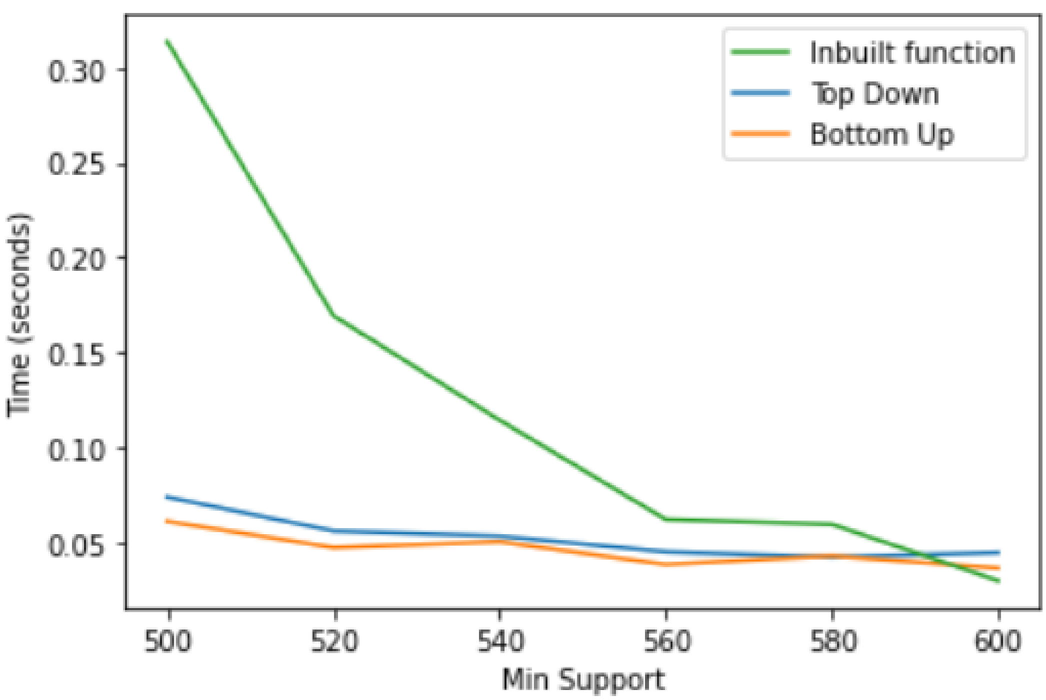
○ We traverse through the header linked list of a particular item and then for each node apply depth first search for each node to generate the conditional pattern base.

○ We have to traverse the itemsets in the same order in which the itemsets were ordered during the building strategy.

● **Since only the bottom up approach has been optimized, we expect it to perform better.**

OUTPUT AND PLOTS:

```
/mnt/c/Users/Tushar Choudhary/Desktop/DA/Data-Analytics/Project-2 main *2 !5 ?2 > py 2019111019\ 2019111026\ fpg.py
RUNNING FOR TOP DOWN
MINSUP: 600    TIME: 0.044722557067871094 seconds
MINSUP: 580    TIME: 0.04230809211730957 seconds
MINSUP: 560    TIME: 0.04513859748840332 seconds
MINSUP: 540    TIME: 0.05336594581604004 seconds
MINSUP: 520    TIME: 0.05614161491394043 seconds
MINSUP: 500    TIME: 0.07393717765808105 seconds
RUNNING FOR BOTTOM UP
MINSUP: 600    TIME: 0.03654813766479492 seconds
MINSUP: 580    TIME: 0.04297065734863281 seconds
MINSUP: 560    TIME: 0.03847551345825195 seconds
MINSUP: 540    TIME: 0.05039858818054199 seconds
MINSUP: 520    TIME: 0.047370195388793945 seconds
MINSUP: 500    TIME: 0.06116986274719238 seconds
RUNNING USING INBUILT LIBRARY
MINSUP: 600    TIME: 0.029991626739501953 seconds
MINSUP: 580    TIME: 0.05961751937866211 seconds
MINSUP: 560    TIME: 0.06219840049743652 seconds
MINSUP: 540    TIME: 0.11445021629333496 seconds
MINSUP: 520    TIME: 0.16938447952270508 seconds
MINSUP: 500    TIME: 0.31352853775024414 seconds
/mnt/c/Users/Tushar Choudhary/Desktop/DA/Data-Analytics/Project-2 main *2 !5 ?2 >
```



Observations and Inferences

Bottom Up FP Growth **runs faster** than Top Down strategy for the given dataset. This might be due to the optimization strategy or could be because of implementational differences that such a thing is happening as the run time difference is very small. Top Down FP Growth runs faster than the inbuilt library for the given dataset.

Question 2: Apriori

Optimization Strategy

● **Hash Mapping**

○ Instead of using apriorGen for generating 2-candidate we use a counting procedure and a 2D array.

○ We iterate through all the transitions and remove the non frequent 1-itemsets from the transaction . For each transaction we then form all the 2 candidates and increment their count in a 2d map.

○ The ones whose bucket has a count >=minSupport is a 2-candidate

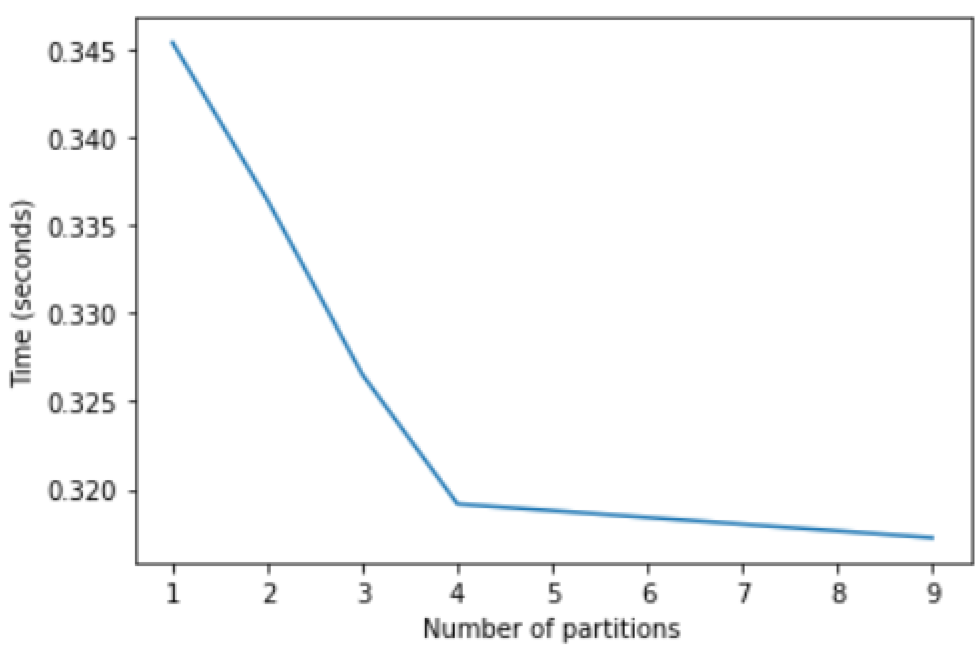
● **Partitioning**

○ Here, for calculating the frequent item sets, we have first partitioned the original dataset and checked if a set is frequent in any of the partitions. If an item set is not frequent in any of the partitions, then it can’t be frequent in the complete dataset either. Hence, such item sets have been scrapped.

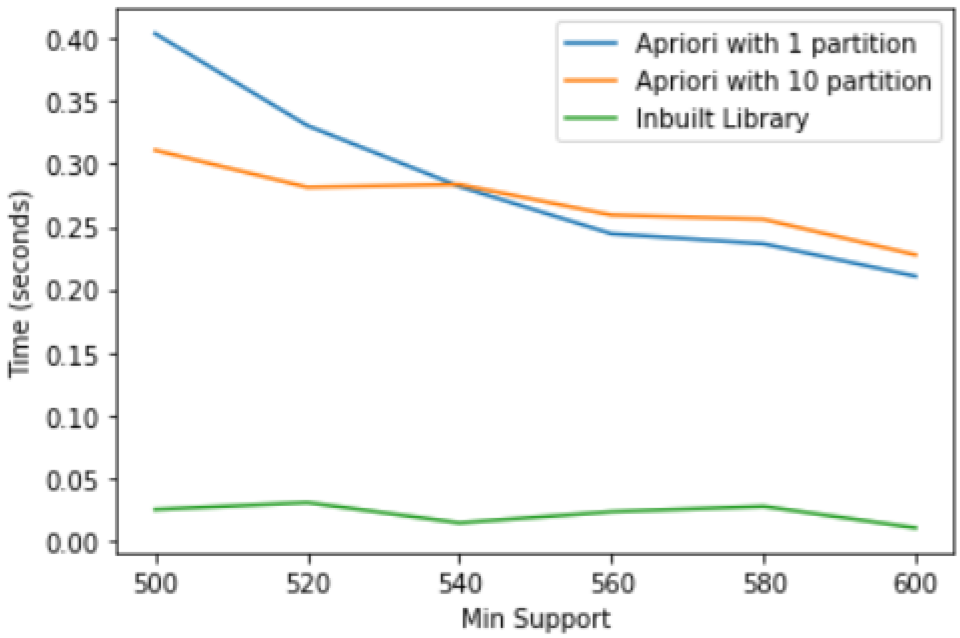
OUTPUT AND PLOTS:

```
/mnt/c/Users/Tushar Choudhary/Desktop/DA/Data-Analytics/Project-2 main *2 !5 ?1 > py 2019111019\ 2019111026\ apriori.py

Testing with number of partitions
NUMBER OF PARTITIONS: 1    TIME: 0.345383882522583 seconds
NUMBER OF PARTITIONS: 2    TIME: 0.3364138603210449 seconds
NUMBER OF PARTITIONS: 3    TIME: 0.3265235424041748 seconds
NUMBER OF PARTITIONS: 4    TIME: 0.31917619705200195 seconds
NUMBER OF PARTITIONS: 9    TIME: 0.3172569274902344 seconds

Testing with different minimum supports
MINSUP: 600    TIME: 0.21045970916748047 seconds
MINSUP: 580    TIME: 0.23634862899780273 seconds
MINSUP: 560    TIME: 0.2443389892578125 seconds
MINSUP: 540    TIME: 0.2820107936859131 seconds
MINSUP: 520    TIME: 0.3306095600128174 seconds
MINSUP: 500    TIME: 0.4038703441619873 seconds

Testing with inbuilt library
MINSUP: 600    TIME: 0.010236263275146484 seconds
MINSUP: 580    TIME: 0.027364730834960938 seconds
MINSUP: 560    TIME: 0.023106813430786133 seconds
MINSUP: 540    TIME: 0.014223098754882812 seconds
MINSUP: 520    TIME: 0.030476093292236328 seconds
MINSUP: 500    TIME: 0.024868488311767578 seconds
```

Plot for Time vs Number of partitions:



Plot for support vs time:



Observations:

Partitioning technique produces a little improvement and runs faster when the number of frequent itemsets are high (that is, **when minimum support is low**). However, **when the minimum support is high**, the number of frequent itemsets is low, and the algorithms waste time when scanning the partitions individually (as not many item sets are to be found, we can scan the whole dataset at once) and give higher runtime compared to the naive approach.

Also, the hashing technique produces significant improvement compared to the naive approach.

Combined Analysis

For the Apriori Algorithm

● Hashing technique **produces improvement**: This is because in the Apriori algorithm the bottleneck happens at the 2-candidate and 2-itemset generation and hashing technique is targeting that very step and trying to optimize that step itself

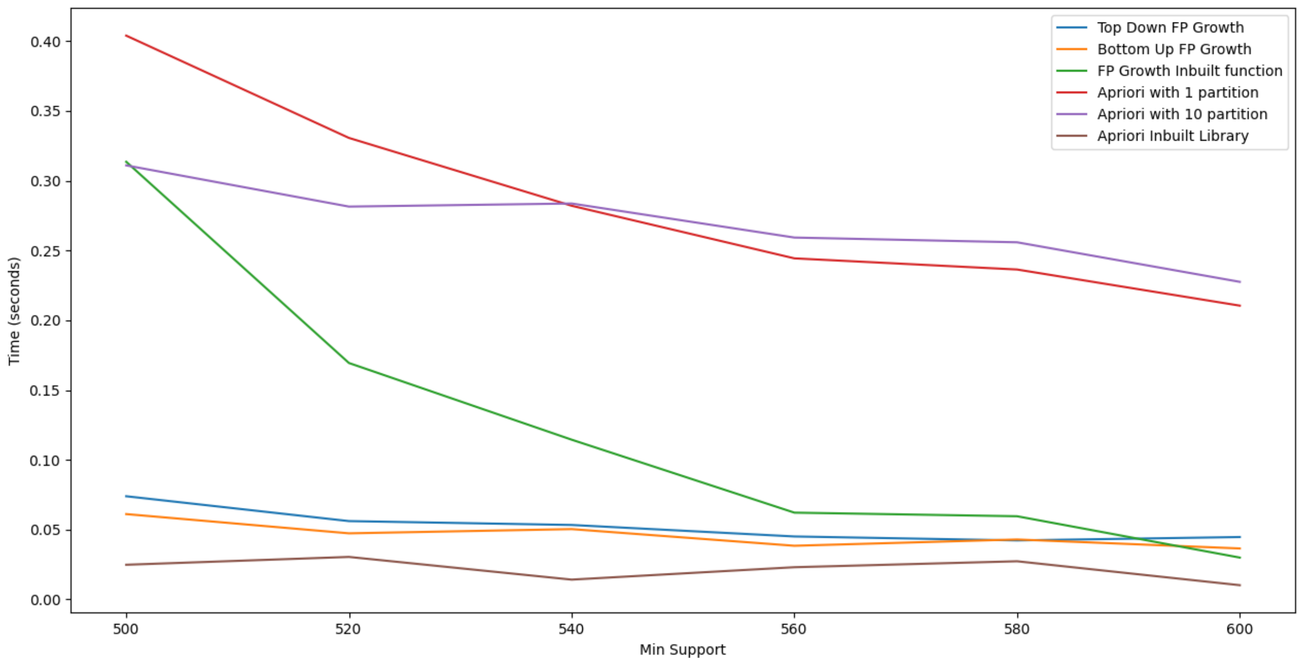
● Apriori is **most steep** with respect to support. If a lower support leads to huge number of frequent itemsets then apriori tends to become very slow which tells us that it is not a very good algorithm for frequent itemset mining

● If the number of unique elements are large then the 2-itemset 2-candidate generation becomes very slow since it is O(n^2) with respect to the number of unique elements

For the FP growth Algorithm

● It looks like that Bottom Up FP growth **is producing results better** than the Top Down FP growth. This might be due to the optimization strategy or could be because of implementational differences that such a thing is happening as the run time difference is very small.

Plots for FP Growth and Apriori :



From here we can see that FP growth with bottom up strategy is giving the best run time and apriori without partitioning is giving the highest runtime among all the algorithms. Also, do notice that apriori algorithms are most steep with respect to the minimum support.

Dataset source: https://www.philippe-fournier-viger.com/spmf/datasets/SIGN.txt