CS524 : Data Mining Assignment #1 Q2

Problem Statement : Apply Frequent Itemset Mining algorithms namely, Apriori, Eclat and FP-Growth with some possible optimizations on given datasets. Also compare these algorithms.

Submitted By : Aman Bilaiya - 2018CSB1069

This report contains frequent itemsets Mining Algorithms. Here 3 Algorithms (Apriori, FPTree, Eclat) are explained and how they perform on the different datasets.

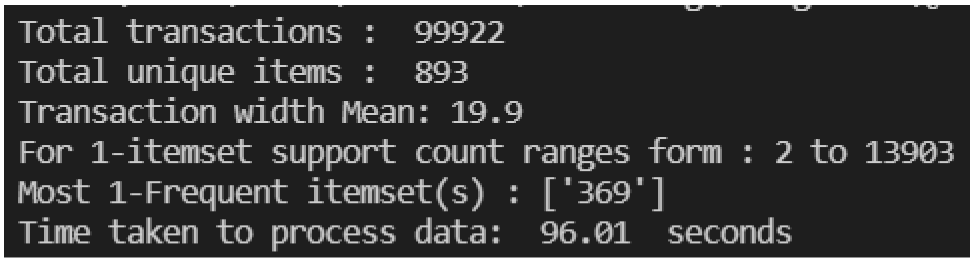
★ INTRODUCTION

In many applications one is interested in how often two or more objects of interest co-occur. The quest is to mine frequent patterns. The prototypical application is market basket analysis, i.e., to mine the sets of items that are frequently bought together at a supermarket by analyzing the customer shopping carts (the so-called "market baskets"). Once we mine the frequent sets, they allow us to extract association rules among the item sets, where we make some statement about how likely are two sets of items to co-occur or to conditionally occur.

★ BRIEF DATASET DESCRIPTION

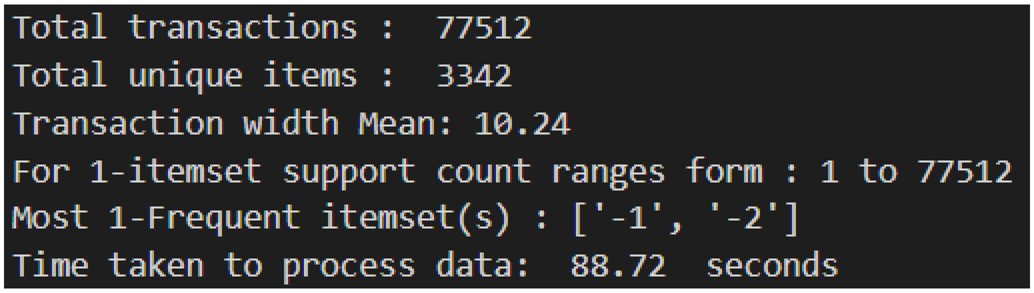
Total 4 datasets are given in .txt format for comparison of performance of each algorithm :-

1. T20i6D100k Dataset link



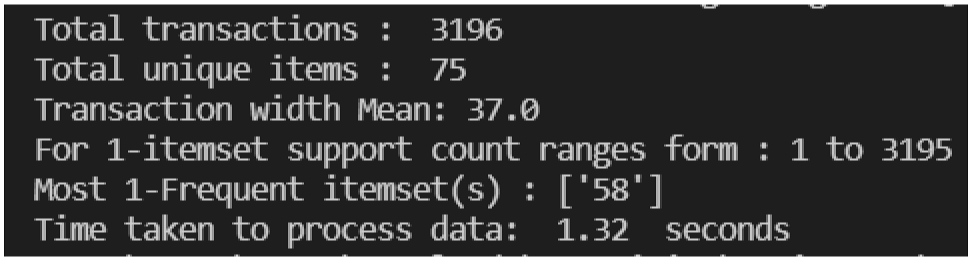
```
Total transactions :  99922
Total unique items :  893
Transaction width Mean: 19.9
For 1-itemset support count ranges form : 2 to 13903
Most 1-Frequent itemset(s) : ['369']
Time taken to process data:  96.01  seconds
```

2. BMS WebView 2 (KDD CUP 2000) link



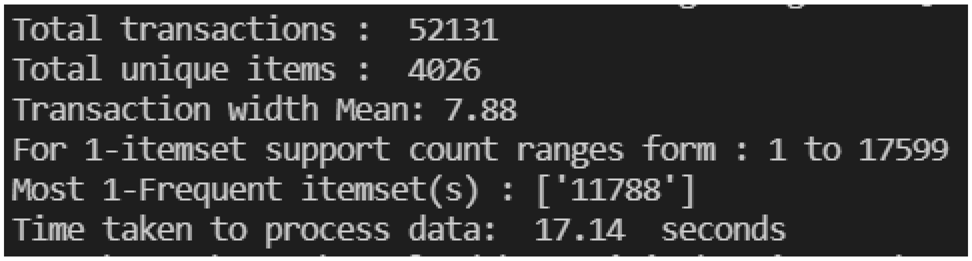
```
Total transactions :  77512
Total unique items :  3342
Transaction width Mean: 10.24
For 1-itemset support count ranges form : 1 to 77512
Most 1-Frequent itemset(s) : ['-1', '-2']
Time taken to process data:  88.72  seconds
```

3. Chess link



```
Total transactions :  3196
Total unique items :  75
Transaction width Mean: 37.0
For 1-itemset support count ranges form : 1 to 3195
Most 1-Frequent itemset(s) : ['58']
Time taken to process data:  1.32  seconds
```

4. Liquor 11 link



```
Total transactions :  52131
Total unique items :  4026
Transaction width Mean: 7.88
For 1-itemset support count ranges form : 1 to 17599
Most 1-Frequent itemset(s) : ['11788']
Time taken to process data:  17.14  seconds
```

The above dataset details are calculated using “dataset_info.py” where I am creating a horizontal and vertical transaction table by iterating over the dataset and also calculating some values shown above. Horizontal transaction table will be used in Apriori & FP Growth Algorithm whereas Vertical transaction table will be used in Eclat Algorithm. Moreover I skipped the BMS WebView 2 dataset as told.

★ ALGORITHMS IMPLEMENTED - Details

I have implemented the algorithms below by referring to pseudo algorithms given in Zaki’s and other books. Also have applied some optimizations to improve algorithm efficiency.

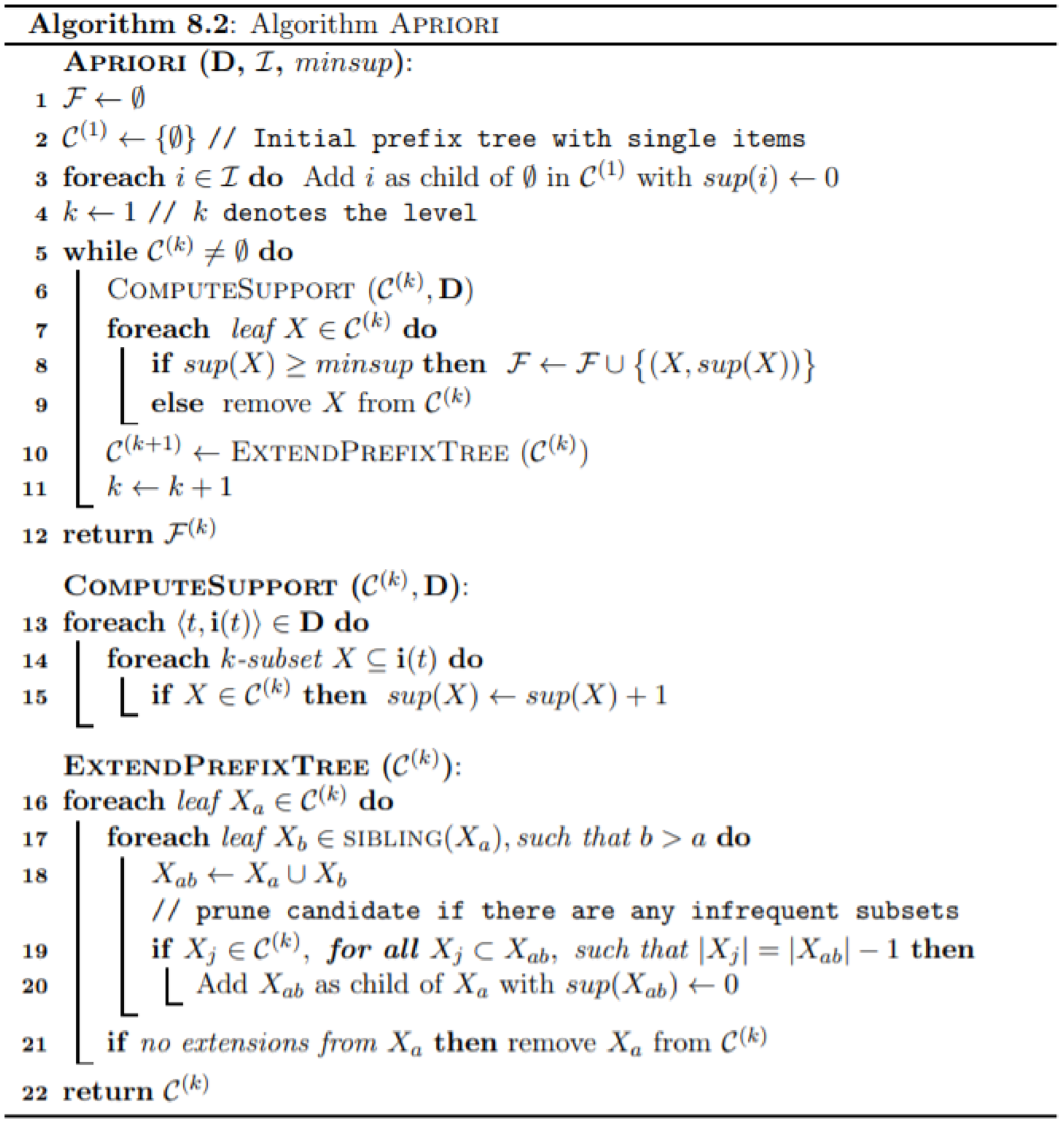
Brute way is to generate all candidate sets and find which ones are frequent but this approach will take exponentially time. So I implemented below 3 algorithms that are better and optimal than brute force.

1) Apriori Algorithm : Level wise approach with extended prefix tree

The Apriori algorithm uses the “Apriori Principle” which significantly improves the brute force method. I have implemented an iterative algorithm where k-frequent itemsets are used to find k+1 candidate itemsets and so-on. So we build itemsets in a top-down manner starting from k=1.

Optimization: For faster candidate set generation, “Extended Prefix Tree” is used that avoids redundant itemsets connections using prefix based extension for candidate generation & thus improves computation efficiency. For more details refer below code or Zaki’s book which is somewhat similar to what I have implemented.

Below is the implementation details via pseudo apriori algorithm with optimization given in Algorithm 8.2 Zaki’s book. [Refer “aprioriOpt.py” for actual code].



Algorithm 8.2: Algorithm APRIORI

```
APRIORI (D, $\mathcal{I}$, minsup):
$\mathcal{F} \leftarrow \emptyset$
$\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$ // Initial prefix tree with single items
foreach $i \in \mathcal{I}$ do Add $i$ as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(i) \leftarrow 0$
$k \leftarrow 1$ // $k$ denotes the level
while $\mathcal{C}^{(k)} \neq \emptyset$ do
    COMPUTESUPPORT ($\mathcal{C}^{(k)}$, D)
    foreach leaf $X \in \mathcal{C}^{(k)}$ do
        if $sup(X) \geq minsup$ then $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
        else remove $X$ from $\mathcal{C}^{(k)}$
    $\mathcal{C}^{(k+1)} \leftarrow$ EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$)
    $k \leftarrow k + 1$
return $\mathcal{F}^{(k)}$

COMPUTESUPPORT ($\mathcal{C}^{(k)}$, D):
foreach $\langle t, \mathbf{i}(t) \rangle \in$ D do
    foreach $k$-subset $X \subseteq \mathbf{i}(t)$ do
        if $X \in \mathcal{C}^{(k)}$ then $sup(X) \leftarrow sup(X) + 1$

EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$):
foreach leaf $X_a \in \mathcal{C}^{(k)}$ do
    foreach leaf $X_b \in$ SIBLING($X_a$), such that $b > a$ do
        $X_{ab} \leftarrow X_a \cup X_b$
        // prune candidate if there are any infrequent subsets
        if $X_j \in \mathcal{C}^{(k)}$, for all $X_j \subset X_{ab}$, such that $|X_j| = |X_{ab}| - 1$ then
            Add $X_{ab}$ as child of $X_a$ with $sup(X_{ab}) \leftarrow 0$
    if no extensions from $X_a$ then remove $X_a$ from $\mathcal{C}^{(k)}$
return $\mathcal{C}^{(k)}$
```

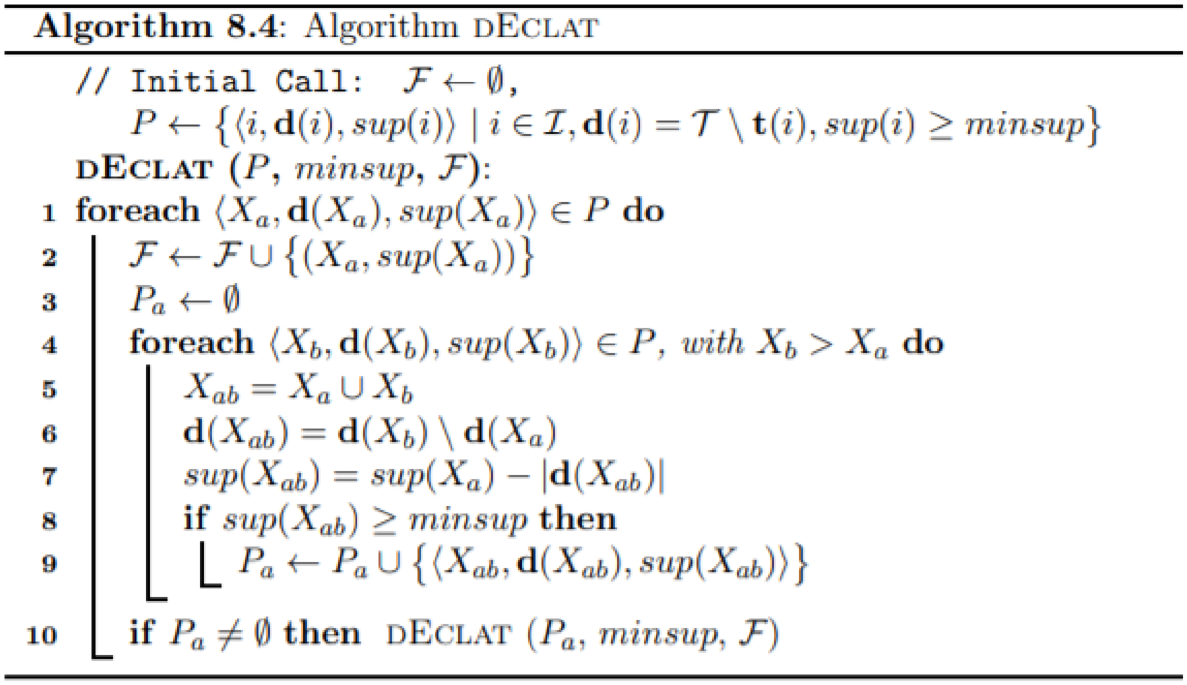
[[[But somehow the optimized apriori version is taking too much time for large datasets, so I have used “apriori_algo.py” for obtaining outputs and results.]]]

2) Eclat Algorithm : Tidset intersection approach with diffset optimization

Eclat uses vertical representation of the database whereas Apriori and FP Growth used horizontal representation.Eclat is also called as vertical apriori algorithm. Vertical representation allows faster calculation of support count for each item. Using set intersection, eclat finds support for itemsets of size k+1 with help of itemsets of size k. I have implemented a recursive algorithm.

Optimization : Used “Diffset **(D)** Optimization” instead of normal Tidset **(T)** eclat. T-eclat does simple tidsets intersection to generate itemsets but what if the itemsets size at level-k if large then it will take time to calculate support count. So, diffsets are used since as length of largest frequent itemset increases → length of diffset decreases thereby aiding faster support count calculation even for large size itemsets. For more details refer below code or Zaki’s book which is somewhat similar to what I have implemented.

Below is the implementation details via pseudo apriori algorithm with optimization given in Algorithm 8.4 Zaki’s book. [Refer “eclat_algo.py” for actual code].



Algorithm 8.4: Algorithm dEclat

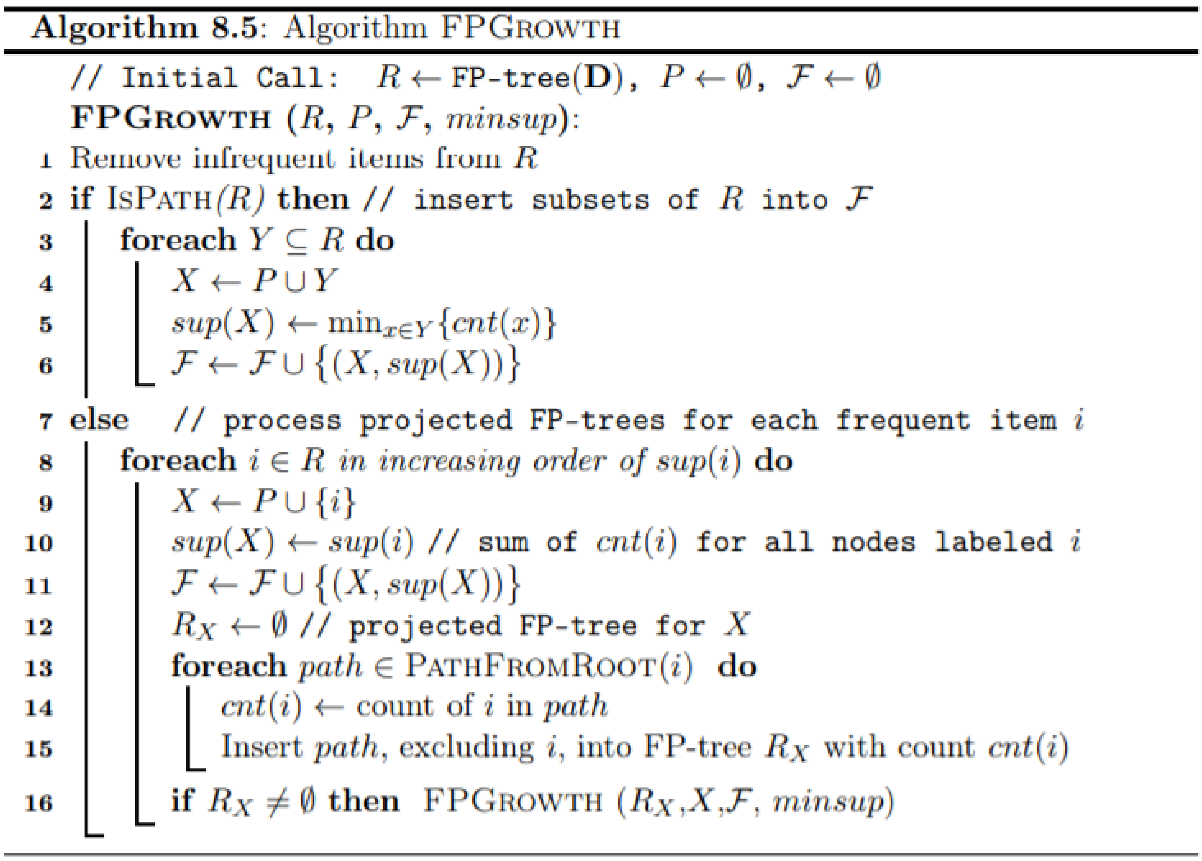
```
// Initial Call: $\mathcal{F} \leftarrow \emptyset$,
    $P \leftarrow \{\langle i, \mathbf{d}(i), sup(i) \rangle \mid i \in \mathcal{I}, \mathbf{d}(i) = \mathcal{T} \setminus \mathbf{t}(i), sup(i) \geq minsup\}$
dEclat ($P$, $minsup$, $\mathcal{F}$):
foreach $\langle X_a, \mathbf{d}(X_a), sup(X_a) \rangle \in P$ do
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\}$
    $P_a \leftarrow \emptyset$
    foreach $\langle X_b, \mathbf{d}(X_b), sup(X_b) \rangle \in P$, with $X_b > X_a$ do
        $X_{ab} = X_a \cup X_b$
        $\mathbf{d}(X_{ab}) = \mathbf{d}(X_b) \setminus \mathbf{d}(X_a)$
        $sup(X_{ab}) = sup(X_a) - |\mathbf{d}(X_{ab})|$
        if $sup(X_{ab}) \geq minsup$ then
            $P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{d}(X_{ab}), sup(X_{ab}) \rangle\}$
    if $P_a \neq \emptyset$ then dEclat ($P_a$, $minsup$, $\mathcal{F}$)
```

3) FP-Growth Algorithm : Frequent pattern tree approach

In this, first FP Tree is built followed by conditional pattern base and conditional FP tree generation to find frequent pattern sets. It is a space optimized algorithm as it uses compact trees to store transaction information. I have implemented a recursive algorithm.

Optimization : For faster tree creation which is the most crucial part of the algorithm I have recursively generated the whole dataset rather than just generating conditional FP trees recursively on the data. For more details refer below code or Zaki's book which is somewhat similar to what I have implemented.

Below is the implementation details via pseudo apriori algorithm with optimization given in Algorithm 8.5 Zaki's book. [Refer "fp_growth_algo.py" for actual code].



Algorithm 8.5: Algorithm FPGrowth

```
// Initial Call: $R \leftarrow$ FP-tree($\mathbf{D}$), $P \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$
FPGrowth ($R$, $P$, $\mathcal{F}$, $minsup$):
Remove infrequent items from $R$
if IsPath($R$) then // insert subsets of $R$ into $\mathcal{F}$
    foreach $Y \subseteq R$ do
        $X \leftarrow P \cup Y$
        $sup(X) \leftarrow \min_{x \in Y}\{cnt(x)\}$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
else // process projected FP-trees for each frequent item $i$
    foreach $i \in R$ in increasing order of $sup(i)$ do
        $X \leftarrow P \cup \{i\}$
        $sup(X) \leftarrow sup(i)$ // sum of $cnt(i)$ for all nodes labeled $i$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
        $R_X \leftarrow \emptyset$ // projected FP-tree for $X$
        foreach $path \in$ PathFromRoot($i$) do
            $cnt(i) \leftarrow$ count of $i$ in $path$
            Insert $path$, excluding $i$, into FP-tree $R_X$ with count $cnt(i)$
        if $R_X \neq \emptyset$ then FPGrowth ($R_X$, $X$, $\mathcal{F}$, $minsup$)
```

NOTE : Refer "readme" file for instructions on how to run the above 3 code files.

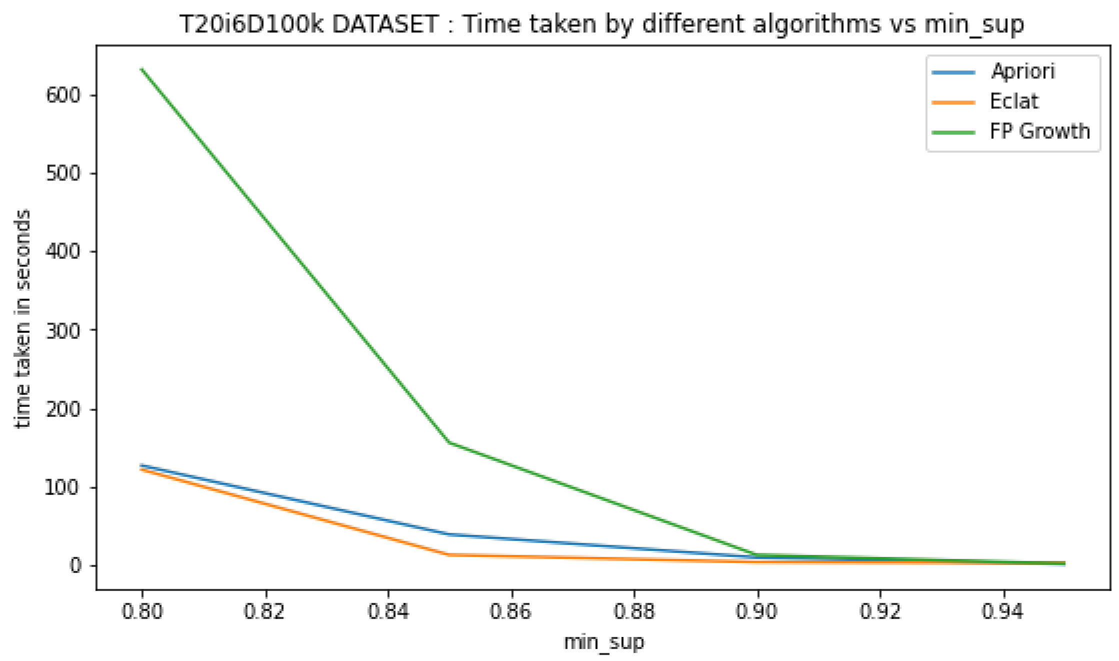
★ ALGORITHMS COMPARISON - Time and Space Analysis

1) T20i6D100k Dataset

TIME AND MEMORY USED BY PROGRAM

Min Support	Apriori	Eclat	FP Growth
0.02	3244.87 seconds 277.171875 MB	See **	632.5755 seconds 749.55859375 MB
0.05	1480.3603 seconds 277.32421875 MB	12.3131 seconds 2866.97265625 MB	155.6620 seconds 423.2578125 MB
0.07	123.4319 seconds 279.36328125 MB	3.1366 seconds 693.390625 MB	12.2902 seconds 301.16796875 MB
0.1	9.3694 seconds 279.13671875 MB	2.0266 seconds 314.86328125 MB	1.1938 seconds 281.24609375 MB

** VS Code froze in between (may be memory limit exceeded) In high end laptop, 121.3131 seconds and 2911.56189 MB

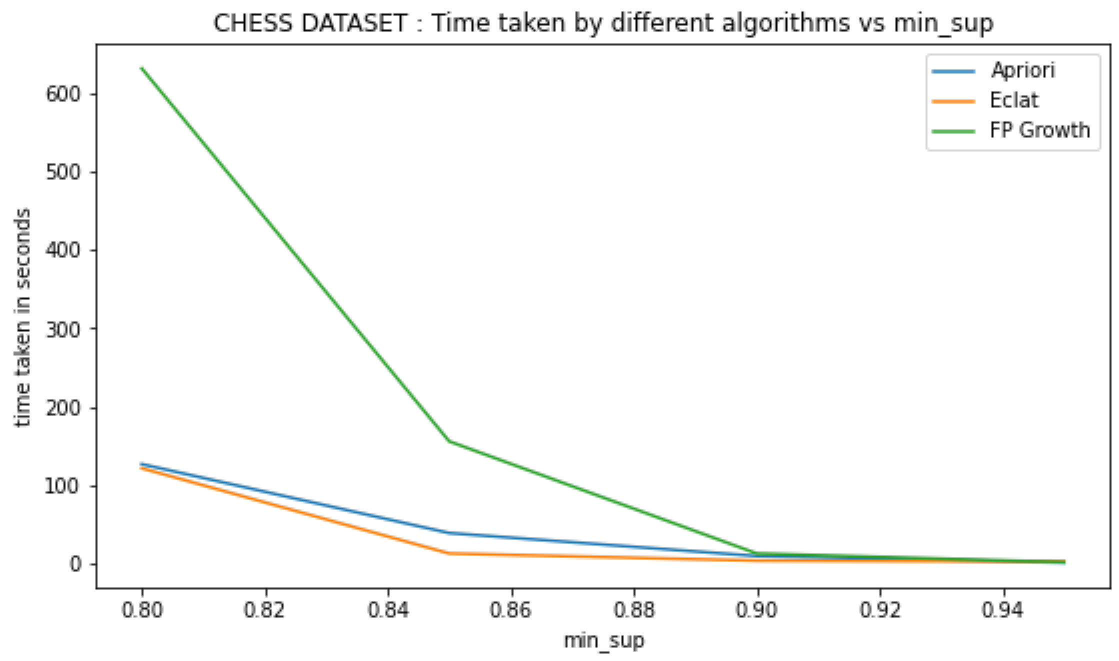


Memory → FP Growth < Apriori < Eclat

2) Chess

TIME AND MEMORY USED BY PROGRAM

Min Support	Apriori	Eclat	FP Growth
0.8	126.4864 seconds 119.046875 MB	0.3481 seconds 109.0625 MB	42.4557 seconds 55.5 MB
0.85	38.4003 seconds 66.96484375 MB	0.0548 seconds 66.421875 MB	12.8666 seconds 50.85546875 MB
0.9	9.0059 seconds 50.1640625 MB	0.0189 seconds 51.1953125 MB	3.2174 seconds 50.11328125 MB
0.95	0.5834 seconds 46.58984375 MB	0.0060 seconds 47.2265625 MB	0.4239 seconds 48.08984375 MB

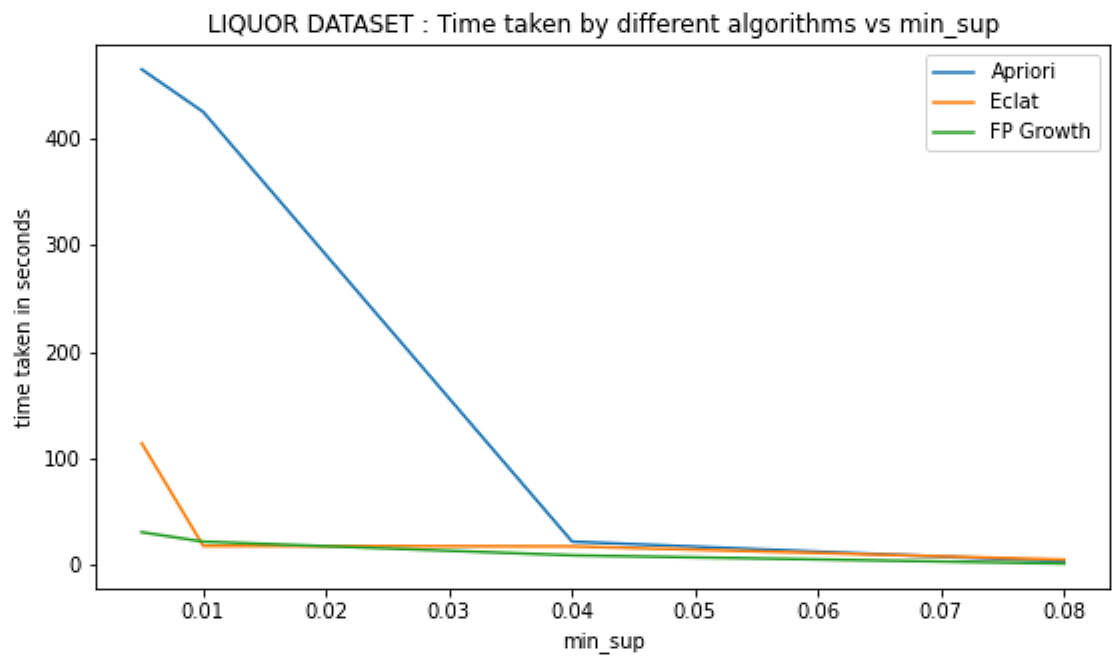


Memory → FP Growth < Eclat < Apriori

3) Liquor_11

TIME AND MEMORY USED BY PROGRAM

Min Support	Apriori	Eclat	FP Growth
0.005	464.7182 seconds 92.16796875 MB	113.9917 seconds 3378.00390625 MB	30.5733 seconds 169.5078125 MB
0.01	424.8332 seconds 90.34765625 MB	17.7150 seconds 2018.25390625 MB	21.7429 seconds 152.5078125 MB
0.04	21.7083 seconds 92.12890625 MB	17.2891 seconds 303.96875 MB	8.9001 seconds 106.62109375 MB
0.08	2.9837 seconds 89.6015625 MB	4.8473 seconds 143.4765625 MB	1.1585 seconds 93.265625 MB



Memory → Apriori < FP Growth < Eclat

Above execution time does not include data preparation time and the data is prepared separately using the “dataset_info.py” file. The graphs are plotted using “plottingComparisions.ipynb”.

NOTE : These execution time and space used are relative based on system processor, RAM and other factors. They may vary for each other run of the program. But the relative comparison will remain the same for the 3 algorithms.

★ OBSERVATIONS - Drawn from above analysis

The **Eclat algorithm** works best when the number of items are much lesser as compared to the number of transactions because the vertical transactions list helps in faster support calculation. Here in the **chess dataset**, there are 75 unique items and 3196 transactions and in **T20i6D100k,** there are 893 items and 99222 transactions**,** thus Eclat performed well.

For **low min support ratios** Eclat and FP growth works better as compared to Apriori algorithm because in Apriori algorithms itemsets with lesser support than minSup are also generated which are then eliminated in candidate itemset pruning resulting in more execution time.

For **high min support ratios**, 1-itemsets or 2-itemsets are there in frequent itemsets and all the 3 algorithms performed equally in terms of run time.

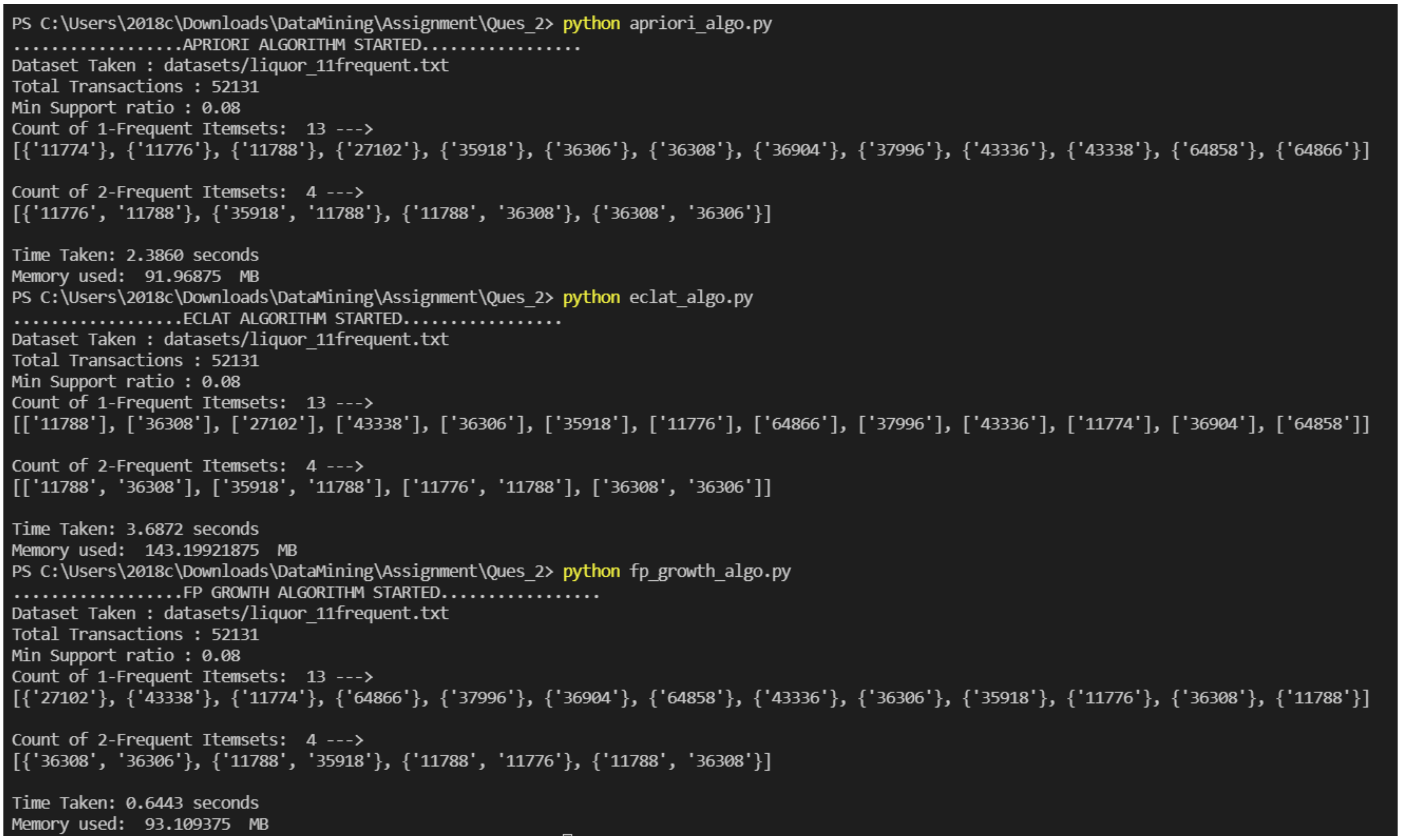
If the transaction dataset has **more repetitions of a group of items** then FP Growth works better because the FP Tree becomes more compact due to its prefix based nature and thus memory efficient. On the other hand, Eclat suffers memory limitation problems because for vertical dataset - width increases due to **large transaction size with low min support**. Here in the **T20i6D100k** dataset, there are around 100K transactions.

In general,

- The Apriori algorithm is slow compared to the other two.
- The Eclat algorithm is faster compared to Apriori as it uses vertical transaction sets.
- Database scans → Apriori **>** Eclat **>** FP Growth
- In FP Growth, FP tree is used which helps to store transactions in compact form resulting in low memory usage.
- Implementation Complexity → Apriori **<** Eclat **<** FP Growth

So one should analyse the dataset properly and choose the best fit itemset mining algorithm which is optimal both in terms of space and time.

★ SAMPLE OUTPUT



```
PS C:\Users\2018c\Downloads\DataMining\Assignment\Ques_2> python apriori_algo.py
..................APRIORI ALGORITHM STARTED.................
Dataset Taken : datasets/liquor_11frequent.txt
Total Transactions : 52131
Min Support ratio : 0.08
Count of 1-Frequent Itemsets:  13 --->
[{'11774'}, {'11776'}, {'11788'}, {'27102'}, {'35918'}, {'36306'}, {'36308'}, {'36904'}, {'37996'}, {'43336'}, {'43338'}, {'64858'}, {'64866'}]

Count of 2-Frequent Itemsets:  4 --->
[{'11776', '11788'}, {'35918', '11788'}, {'11788', '36308'}, {'36308', '36306'}]

Time Taken: 2.3860 seconds
Memory used:  91.96875  MB
PS C:\Users\2018c\Downloads\DataMining\Assignment\Ques_2> python eclat_algo.py
..................ECLAT ALGORITHM STARTED.................
Dataset Taken : datasets/liquor_11frequent.txt
Total Transactions : 52131
Min Support ratio : 0.08
Count of 1-Frequent Itemsets:  13 --->
[['11788'], ['36308'], ['27102'], ['43338'], ['36306'], ['35918'], ['11776'], ['64866'], ['37996'], ['43336'], ['11774'], ['36904'], ['64858']]

Count of 2-Frequent Itemsets:  4 --->
[['11788', '36308'], ['35918', '11788'], ['11776', '11788'], ['36308', '36306']]

Time Taken: 3.6872 seconds
Memory used:  143.19921875  MB
PS C:\Users\2018c\Downloads\DataMining\Assignment\Ques_2> python fp_growth_algo.py
..................FP GROWTH ALGORITHM STARTED.................
Dataset Taken : datasets/liquor_11frequent.txt
Total Transactions : 52131
Min Support ratio : 0.08
Count of 1-Frequent Itemsets:  13 --->
[{'27102'}, {'43338'}, {'11774'}, {'64866'}, {'37996'}, {'36904'}, {'64858'}, {'43336'}, {'36306'}, {'35918'}, {'11776'}, {'36308'}, {'11788'}]

Count of 2-Frequent Itemsets:  4 --->
[{'36308', '36306'}, {'11788', '35918'}, {'11788', '11776'}, {'11788', '36308'}]

Time Taken: 0.6443 seconds
Memory used:  93.109375  MB
```

Refer to the "outputs" folder where you can find outputs[Frequent Itemsets, Time Taken, Memory Used] for all 3 algorithms for different datasets and min Support value being used. Naming Convention Used : <algo>_<datasetName>_<minSupport>.txt

Size of maximal frequent itemsets, Number of maximal frequent itemsets for some minimum support values is printed over the output.

REFERENCES :-

[1] Chapter 8 : Itemset Mining, Mohammed J.Zaki, “Data Mining and Analysis: Fundamental Concepts and Algorithms”

[2] Lecture Slides

[3] Numpy, time, collections, itertools modules documentation