ANALYSIS OF FP-TREE & APRIORI ALGORITHMS

DATA MINING [COURSE -MTL782]

PREPARED BY

Harsh Kumar [2016MT10629]

Parminder Singh [2016MT10630]

Anshuman Shrivastava [2016MT10620]

Improvements in Apriori:

Improvement 1 (reference paper-3):

In this improvement, we used L1 to identify the target transactions for Ck. Before scanning all transaction records to count the support count of each candidate, use L1 to get the transaction IDs of the minimum support count between x and y, and thus scan for C2 only in these specific transactions i.e. if in C2 if we have x and y items then we take the item with the minimum count . Let us say between x and y, x has less count than y. Then we find the transaction ids corresponding to transactions in which x is present. Now to check the count of (x,y) set we will only check in these transaction ids we don't check in the whole data for the count of (x,y).

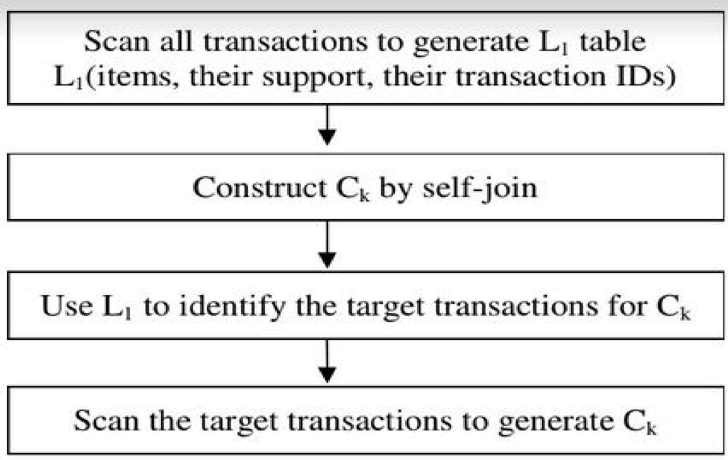


Figure 1: Steps for C_k generation

```
def findwithmin(t,d1):
    temp=t.split(" ")
    imin=10000000000
    k="-1"
    for i in temp:
        if imin>d1[i]:
            imin=d1[i]
            k=i
    return k
def update(data,d,C,s,k,d1,d2):
  for trans in C:
```

```
    k=findwithmin(trans,d1)                                     #IMPROVEMENT3
    lis=d2[k]
    for i in lis:
      if set(trans.split(" ")).issubset(set(data[i].split(" "))):
        if d[trans]<s:                                          #IMPROVEMENT1
            d[trans]+=1
        else:
            break
```

Improvement2 (reference paper1):

While updating the count for the itemsets we break the loop when the count for the itemset becomes equal to the minsupport count value. Because while we check all elements Ck for minsup value get Lk we don't need the exact value of the count we just need to check whether it is greater than minsup or not

```
 if set(trans.split(" ")).issubset(set(data[i].split(" "))):
        if d[trans]<s:                                          #IMPROVEMENT1
            d[trans]+=1
        else:
            break
```

Improvement 3 (reference paper-2):

We reduce the size of the data set using itemsets of length 1 in L1. If the item is not present in L1 it cannot be present in Lk (for k=2,3,...). So we removed the items which are not in L1 from each transaction of data.

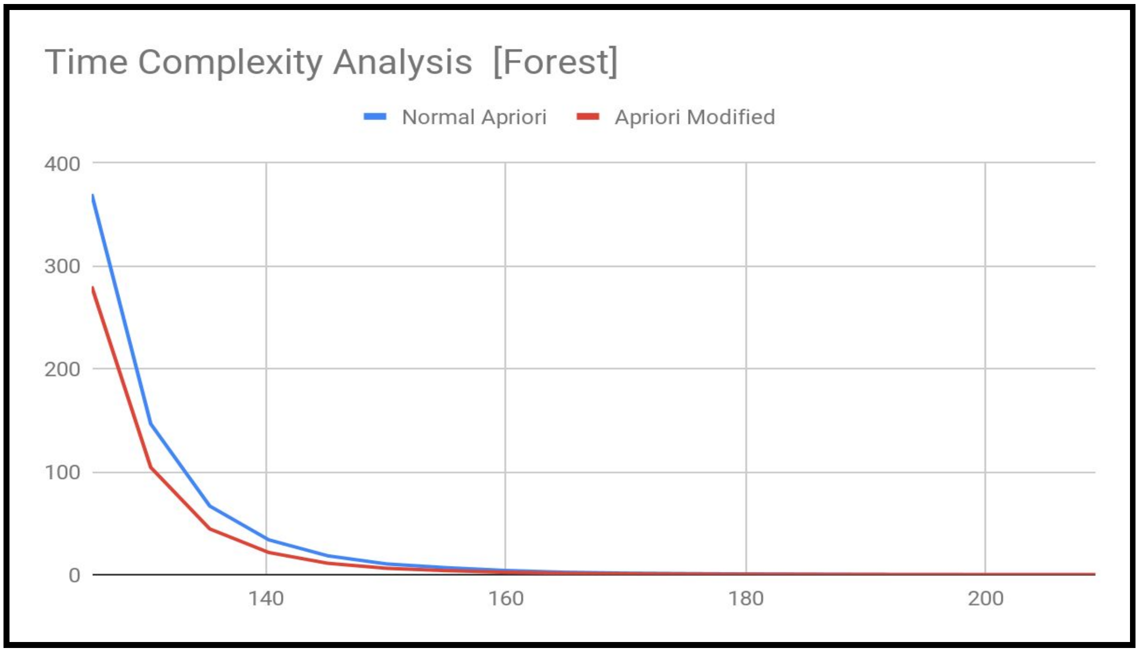
```
for i  in range(len(data)):
#IMPROVEMENT2
    tempdata=data[i].split(" ")
    for t in tempdata:
        if t not in L1:
            tempdata.remove(t)
    data[i]=" ".join(tempdata)ta)
```

Improvement 4:

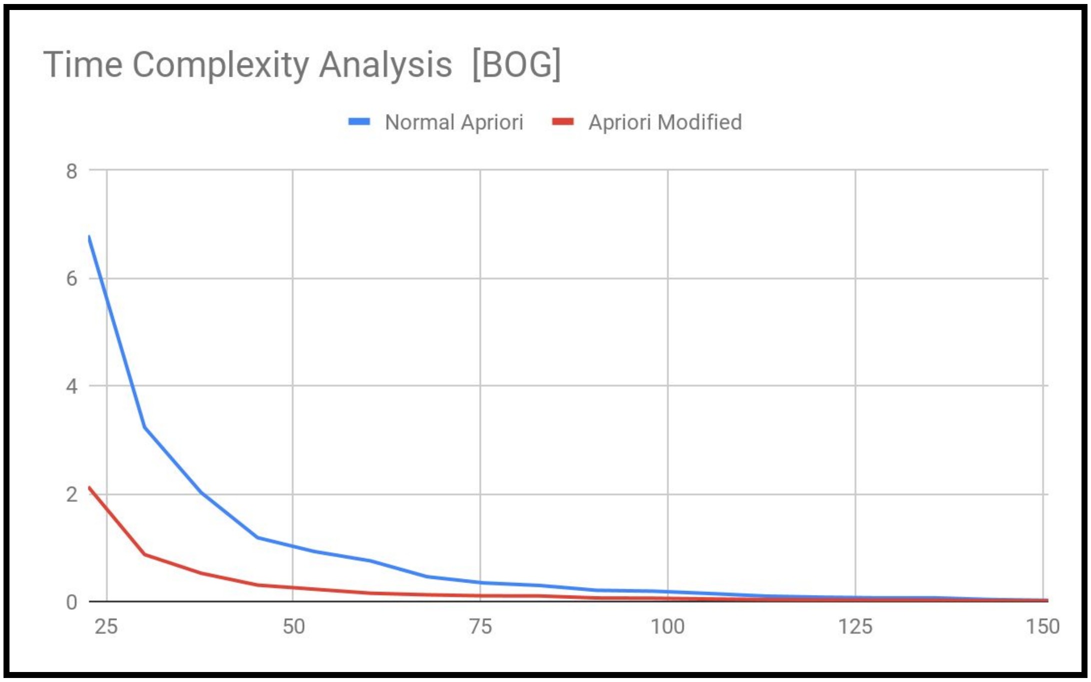
Python dictionary is used in place of the hash tree which is used in classical apriori. In hash tree time taken to the search value is reduced but in a python dictionary, we can find whether an item is present in O(1) time.

Runtime Analysis

SN	Dataset	Minsup	Minsup_count	Normal Apriori	Apriori Modified
1.1	Forest Dataset	0.51	125.46	369.8628353	280.085547
1.2	Forest Dataset	0.53	130.38	146.5202382	104.2610773
1.3	Forest Dataset	0.55	135.3	66.68889585	44.47044943
1.4	Forest Dataset	0.57	140.22	33.82776898	21.54700125
1.5	Forest Dataset	0.59	145.14	18.31644186	11.06271628
1.6	Forest Dataset	0.61	150.06	10.40996315	6.177876277
1.7	Forest Dataset	0.63	154.98	6.839886994	3.944019754
1.8	Forest Dataset	0.65	159.9	4.105988598	2.338918992
1.9	Forest Dataset	0.67	164.82	2.383645968	1.328967941
2	Forest Dataset	0.69	169.74	1.509015117	0.825363508
2.1	Forest Dataset	0.71	174.66	1.074644173	0.557106452
2.2	Forest Dataset	0.73	179.58	0.723508757	0.356011074
2.3	Forest Dataset	0.75	184.5	0.513597631	0.25579329
2.4	Forest Dataset	0.77	189.42	0.305207917	0.158231215
2.5	Forest Dataset	0.79	194.34	0.15373922	0.08039776
2.6	Forest Dataset	0.81	199.26	0.115379374	0.05816541798
2.7	Forest Dataset	0.83	204.18	0.06625803499	0.03789041698
2.8	Forest Dataset	0.85	209.1	0.041250337	0.026663022



SN	Dataset	Minsup	Minsup_count	Normal Apriori	Apriori Modified
1.1	Bog Dataset	0.06	22.62	6.792396808	2.129213304
1.2	Bog Dataset	0.08	30.16	3.229618345	0.870452203
1.3	Bog Dataset	0.1	37.7	2.021565655	0.52427963
1.4	Bog Dataset	0.12	45.24	1.183418456	0.304271091
1.5	Bog Dataset	0.14	52.78	0.928064682	0.227992747
1.6	Bog Dataset	0.16	60.32	0.753034808	0.153345775
1.7	Bog Dataset	0.18	67.86	0.45938158	0.12378127
1.8	Bog Dataset	0.2	75.4	0.344660401	0.104807444
1.9	Bog Dataset	0.22	82.94	0.296480747	0.102046439
2	Bog Dataset	0.24	90.48	0.207210118	0.06480499
2.1	Bog Dataset	0.26	98.02	0.191440365	0.060770246
2.2	Bog Dataset	0.28	105.56	0.147925834	0.046980096
2.3	Bog Dataset	0.3	113.1	0.099376764	0.035631683
2.4	Bog Dataset	0.32	120.64	0.079399446	0.030545766
2.5	Bog Dataset	0.34	128.18	0.067173102	0.02649865
2.6	Bog Dataset	0.36	135.72	0.067285209	0.024668042
2.7	Bog Dataset	0.38	143.26	0.035038495	0.014230867
2.8	Bog Dataset	0.4	150.8	0.020616904	0.010799221



Time Complexity Analysis [BOG]
Normal Apriori
Apriori Modified
8
6
4
2
0
25
50
75
100
125
150

FP Tree: Modification

Fpmax: Mining MFI's We extend the FP-growth method and get algorithm Fpmax described in Figure 3. Like FP growth, algorithm Fpmax is also recursive. In the initial call, an FP-tree is constructed from the first scan of the database. A linkedlist Head contains the items that form the conditional base of the current call. Before recursively calling Fpmax, we already know that the set containing all items in Head and the items in the FP-tree is not a subset of any existing MFI. If there is only one single path in the FP-tree, this single path together with Head is an MFI of the database. In line 2, we use the MFI tree data structure to keep track of all MFI's. If the FP-tree is not a single-path tree, then for each item in the header-table, the item is appended to Head, and line 7 calls function subset checking to check if the new Head together with all frequent items in the Head-conditional pattern base is a subset of any existing MFI. If not, Fpmax will be called recursively.

Procedure Fpmax(T)

Input: T: an FP-tree
Global:
MFIT: an MFI-tree.
Head: a linked list of items.
Output: The MFIT that contains all MFI's
Method:
1.if T only contains a single path P
2. insert Head U P into MFIT
3.else for each i in Header-table of T
4. Append i to Head;
5. Construct the Head-pattern base
6. Tail = {frequent items in base}
7. subset checking(Head U Tail);
8. if Head U Tail is not in MFIT
9. construct the FP-tree THead;
10. call Fpmax(THead);
11. remove i from Head.

FP-Tree Code block:

```
def frequent_itemsets(itemsets, minsup):

    """ Initiates the fpgrowth algorithm """
    tree = build_tree(itemsets, minsup)[0]
    for itemset in fpgrowth(tree, minsup):
    yield itemset
```

FP-Tree Modified Code block:

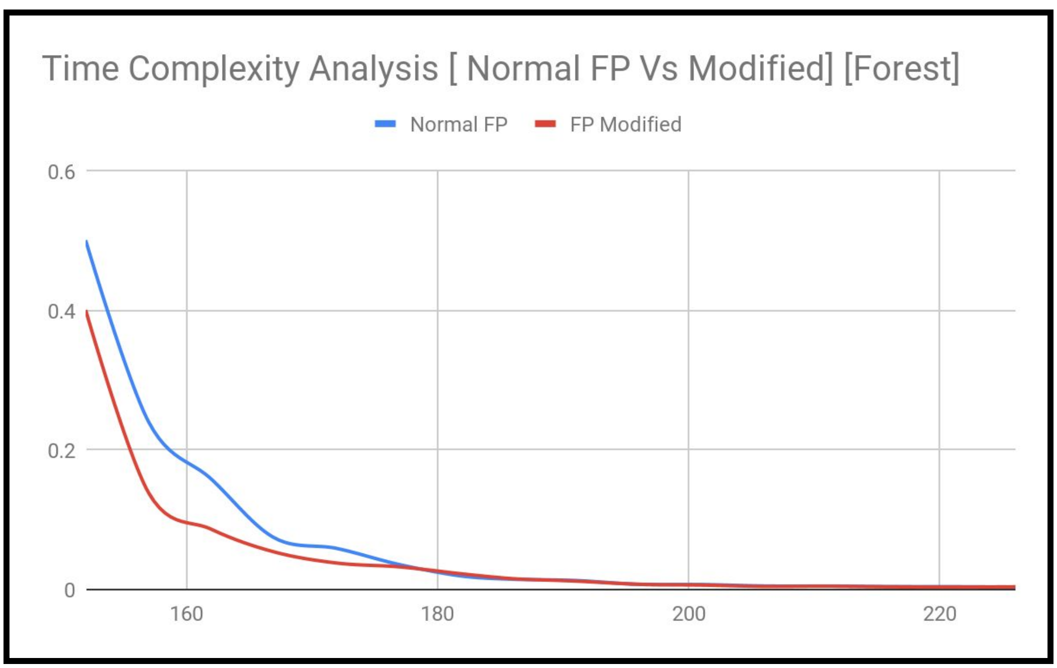
```
def maximal_frequent_itemsets(itemsets, minsup):
      """ Starts the fpmax algorithm """
      tree, rank = build_tree(itemsets, minsup)
      mfit = MFITree(rank)
      for itemset in fpmax(tree, minsup, mfit):
      yield itemset

def fpmax(tree, minsup, mfit):
      """
      Performs the fpmax algorithm on the given tree to yield all
*maximal* frequent itemsets.
      Parameters
      ----------
      tree : FPTree
      minsup : int
      mfit : MFITree
      Keeps track of what itemsets have already been output
      Yields
      ------
      lists of strings
  *Maximal* Set of items that has occurred in minsup itemsets.
      """
      items = list(tree.nodes.keys())
      largest_set = sorted(tree.cond_items+items, key=mfit.rank.get)
      if tree.is_path:
      if not mfit.contains(largest_set):
            largest_set.reverse()
            mfit.cache = largest_set
            mfit.insert_itemset(largest_set)
            yield largest_set
      else:
      # Loop over each item in tree creating another conditional tree
      items.sort(key=tree.rank.get)
      for item in items:
            # Check if the tree will produce a subset already produced
            if mfit.contains(largest_set):
            return
            largest_set.remove(item)
            cond_tree = tree.conditional_tree(item, minsup)
            for mfi in fpmax(cond_tree, minsup, mfit):
            yield mfi
```

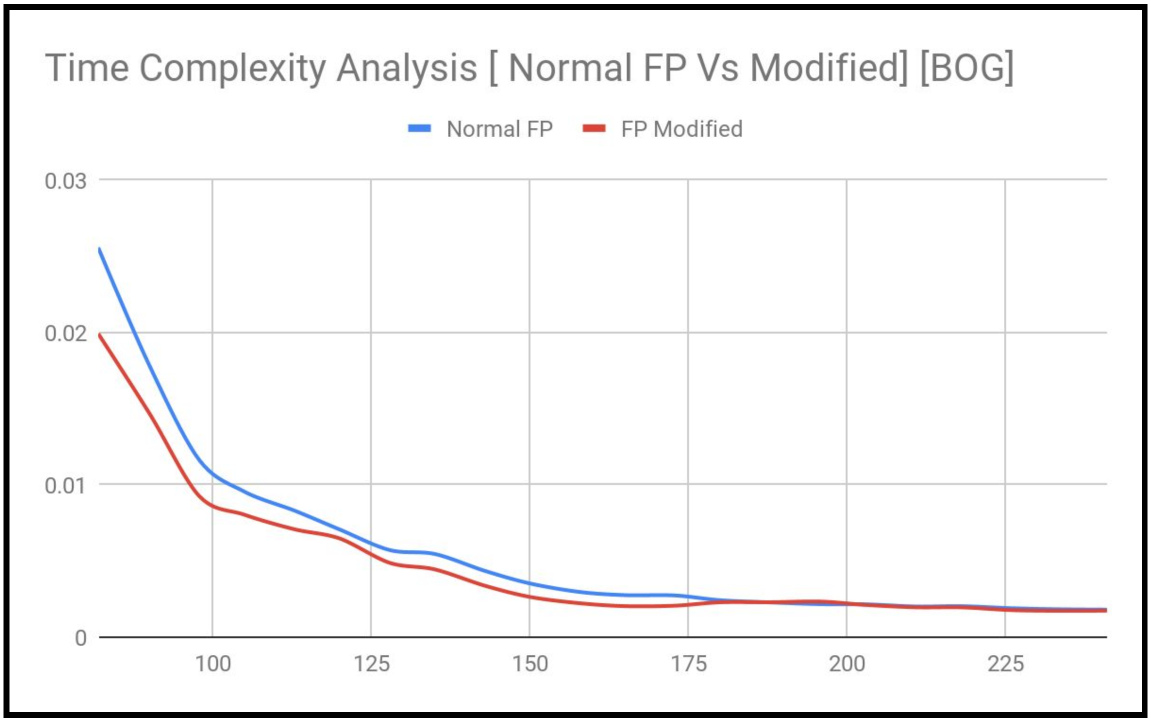
Runtime Analysis

Comparing Normal FP-Tree with Modified One

SN	Dataset	Minsup	Minsup_count	Normal FP	FP Modified
1.1	Forest Dataset	0.62	152	0.500808	0.400808
1.2	Forest Dataset	0.64	157	0.240808	0.1391
1.3	Forest Dataset	0.66	162	0.158367	0.086438
1.4	Forest Dataset	0.68	167	0.074336	0.053972
1.5	Forest Dataset	0.7	172	0.058916	0.037973
1.6	Forest Dataset	0.72	177	0.03571	0.032487
1.7	Forest Dataset	0.74	182	0.019175	0.022402
1.8	Forest Dataset	0.76	186	0.01479	0.015623
1.9	Forest Dataset	0.78	191	0.012979	0.012128
2	Forest Dataset	0.8	196	0.007536	0.007609
2.1	Forest Dataset	0.82	201	0.007231	0.00641
2.2	Forest Dataset	0.84	206	0.005148	0.004466
2.3	Forest Dataset	0.86	211	0.004887	0.004776
2.4	Forest Dataset	0.88	216	0.004307	0.003843
2.5	Forest Dataset	0.9	221	0.004113	0.00347
2.6	Forest Dataset	0.92	226	0.00369	0.003937



SN	Dataset	Minsup	Minsup_count	Normal FP	FP Modified
1.1	Bog Dataset	0.22	82	0.025559	0.019912
1.2	Bog Dataset	0.24	90	0.017886	0.0147297
1.3	Bog Dataset	0.26	98	0.011585	0.009226
1.4	Bog Dataset	0.28	105	0.009558	0.0080401
1.5	Bog Dataset	0.3	113	0.008295	0.007075
1.6	Bog Dataset	0.32	120	0.007077	0.006487
1.7	Bog Dataset	0.34	128	0.005712	0.004872
1.8	Bog Dataset	0.36	135	0.005464	0.004454
1.9	Bog Dataset	0.38	143	0.004345	0.003371
2	Bog Dataset	0.4	150	0.003531	0.002647
2.1	Bog Dataset	0.42	158	0.002963	0.002225
2.2	Bog Dataset	0.44	165	0.002761	0.00204
2.3	Bog Dataset	0.46	173	0.002738	0.00207
2.4	Bog Dataset	0.48	180	0.002428	0.002292
2.5	Bog Dataset	0.5	188	0.002283	0.002297
2.6	Bog Dataset	0.52	196	0.002169	0.002335
2.7	Bog Dataset	0.54	203	0.002157	0.002118
2.8	Bog Dataset	0.56	211	0.002014	0.001962
2.9	Bog Dataset	0.58	218	0.002038	0.001961
3	Bog Dataset	0.6	226	0.001902	0.001778
3.1	Bog Dataset	0.62	233	0.001836	0.001735
3.2	Bog Dataset	0.64	241	0.001808	0.001743



References:

[1] Fast Algorithms for Frequent Itemset Mining Using FP-Trees [Online]. Available :https://users.encs.concordia.ca/~grahne/papers/tdke04.pdf.

[2] The FP-Growth Algorithm. [Online]. Available: https://adataanalyst.com/machine-learning/fp-growth-algorithm-python-3/

[3] AN IMPROVED APRIORI ALGORITHM FOR ASSOCIATION RULES. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1403/1403.3948.pdf

[4] Modified Apriori Algorithm For Predefined Support And Confidence In Cloud Computing Environment For Frequent Pattern Mining [Online]. Available: https://www.ijert.org/research/modified-apriori-algorithm-for-predefind-support-and-confidence-in-cloud-computing-environment-for-frequent-pattern-mining-IJERTV2IS50493.pdf

[5] An Approach of Improvisation in Efficiency of Apriori Algorithm [Online]. Available: https://peerj.com/preprints/1159v1.pdf