ISSN (e): 2250 – 3005 || Vol, 04 || Issue, 5 || May – 2014 ||
International Journal of Computational Engineering Research (IJCER)

Efficient Ranking and Suggesting Popular Itemsets In Mobile Stores Using Fp Tree Approach

[1]B.Sujatha Asst prof,[2]Shaista Nousheen Asst.prof, [3]Tasneem rahath Asst prof, [4] Nikhath Fatima Asst.prof

Abstract

We considered the problem of ranking the popularity of items and suggesting popular items based on user feedback. User feedback is obtained by iteratively presenting a set of suggested items, and users selecting items based on their own preferences either the true popularity ranking of items, and suggest true popular items. We consider FP tree approach with some modifications overcoming the complexity that has been seen in other randomized algorithms. The most effective feature of this approach is that it reduces the number of database scans and complexity.

I. INTRODUCTION

1.1 TERMINOLOGY:

In this section we first want to introduce the different terms that we were going to use in our paper as fallows.

1.1.1 Ranking: Ranking is giving rank scores to the most popular item by taking user feedback. The most frequently occurring item is given the highest rank score.

1.1.2 Selection: We focus on the ranking of items where the only available information is the observed selection of items. In learning of the users preference over items, one may leverage some side information about items, but this is out of the scope of this paper.

1.1.3 Imitate: The user study was conducted in very famous mobile stores and which has been used to set of mobiles. The user may check the list and select the set of mobiles which they like most and depending on those like results the new suggestion list has been developed by the algorithm.

1.1.4 Popular: In practice, one may use prior information about item popularity. For example, in the survey the user may select the suggested mobile or they may also select the others. If they selected the already suggested items they will become more popular and if he don't they may get out of the popular list.
1.1.5 Association Rule: Association Rules are if/then statements that help uncover relationships between seemingly unrelated data in the relational database or other information repository. An example of an association rule would be **if a customer buys a nokia mobile, he is 70% interested in also purchasing nokia accessories**.

II. THEORETICAL STUDY

We consider the mobile phone selection and suggesting the best sold mobile and their combinations that were most liked by most of the users. Consider a set of mobiles M: (m1, m2, m3, m4, ….mn) where n > 1. Now we were calculating the set of items in C where were mostly sold and mostly liked by the users, as S

S: (s1, s2, s3, s4, …. sg) where g > 1.

We need to consider an item I, we interpret si as the portion of users that would select item i if suggestions were not made. We assume that the popularity rank scores s as follows:

a) Items of set S were estimated to is as s1 $\geq$ s2 $\geq$ s3 $\geq$ …. sc,
b) s is completely normalized such that it is a probability distribution, i.e., s1 + s2 + s3 + …. +sc = 1. c) si is always positive for all items i.

III. PROPOSED ALGORITHM AND STUDY

We have some of the systems already existing in the same field and we have also identified some of the disadvantages in them as follows:

- the popularity for any item is given based on the production of that item. This may not give good result because customers may not have the knowledge of true popularity they needed and depend on the results given by the producer.
- the updates are performed regardless of the true popularity by virtual analysis.
- Producer have to analyse things manually and complexity involves in this. Due to this time consumption may be high.
- the algorithms used in this system may fail to achieve true popularity.

We consider the problem learning of the popularity of items that is assumed to be unknown but has to be learned from the observed user’s selection of items. We have selected a mobile market and mobile distribution outlets as our data set and examined them completely in all areas where we can give the list of items suggested by the users and we have made web-application to make an survey at real-time and considered the data given by more that 1000 members of different categories of people and applied our proposed Fp tree approach on the set of data and started suggesting the item in the mobile outlets for the actual users, which had helped the mobile phone companies and also the outlet in-charges. We have implemented the same in some of the mobile outlets in INDIA where we got very good results. The actual goal of the system is to efficiently learn the popularity of items and suggest the popular items to users. This was done to the user to suggest them the mostly used mobiles and their accessories, such that they also buy the best and at the same time the outlet owner will also get benefited. The most important feature in our project is suggesting the users by refreshing the latest results every time the user gives the input and changes his like list.

Now we have overcome many of the disadvantages of the existing systems and achieved many advantages with the proposed algorithm and method as follows:

- In our approach, we consider the problem of ranking the popularity of items and suggesting popular items based on user feedback.
- User feedback is obtained by iteratively presenting a set of suggested items, and users selecting items based on their own preferences either from this suggestion set or from the set of all possible items.
- The goal is to quickly learn the true popularity ranking of items and suggest true popular items.
- In this system better algorithms are used. The algorithms use ranking rules and suggestion rules in order to achieve true popularity.

IV. PROPOSED APPROACHFP-TREE

Like most traditional studies in association mining, we de_ne the frequent pattern mining problem as follows.De_nition 1 (Frequent pattern) Let I = fa1; a2; : : :; amg be a set of items, and a transaction database DB = hT1; T2; : : :; Tni, where Ti (i 2 [1::n]) is a transaction which contains a set of items in I. The support1(or occurrence frequency) of a pattern A, which is a set of items, is the number of transactions containing A in DB. A, is a frequent pattern if A's support is no less than a prede_ned minimum support threshold, _. 2Given a transaction database DB and a minimum support threshold, _, the problem of _nding the complete set of frequent patterns is called the frequent pattern mining problem.

2.1 Frequent Pattern Tree

To design a compact data structure for efficient frequent pattern mining, let's _rst examine a tiny example.Example 1 Let the transaction database, DB, be (the _rst two columns of) Table 1 and the minimum supportthreshold be 3.

TID	Items bought	(Ordered)frequent items
100	f, a, c, d, g, i,m,p	f, c, a,m, p
200	a, b, c, f, l,m,o	f, c, a, b,m
300	b, f, h, j,o	f, b
400	b, c, k, s,p	c, b, p
500	a, f, c, e, l, p,m,n	f, c, a,m, p

A compact data structure can be designed based on the following observations:

[1]. Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database *DB* to identify the set of frequent items (with *frequency count* obtained as a by-product).

[2]. If the *set* of frequent items of each transaction can be stored in some compact structure,it may be possible to avoid repeatedly scanning the original transaction database.

[3]. If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as *count*. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.

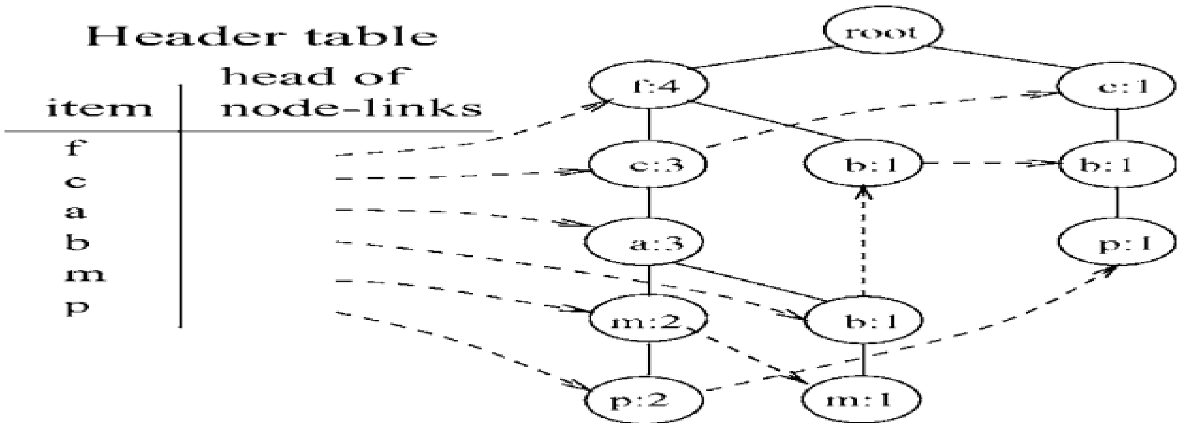
[4]. If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the *count* is registered properly. If the frequent items are sorted in their *frequency descending order*,there are better chances that more prefix strings can be shared.With the above observations, one may construct a frequent-pattern tree as follows.

First, a scan of *DB* derives a *list* of frequent items, _(*f* :4), (*c*:4), (*a*:3), (*b*:3), (*m*:3), (*p*:3)_(the number after ":" indicates the support), in which items are ordered in frequency descending rder. This ordering is important since each path of a tree will follow this order. For convenience of later discussions, the frequent items in each transaction are listed in this ordering in the rightmost column of Table 1.Second, the root of a tree is created and labeled with "*null*". The FP-tree is constructed as follows by scanning the transaction database *DB* the second time.

[1]. The scan of the first transaction leads to the construction of the first branch of the tree:
[2]. _(*f* :1), (*c*:1), (*a*:1), (*m*:1), (*p*:1)_. Notice that the frequent items in the transaction are listed according to the order in the *list* of frequent items.
[3]. For the second transaction, since its (ordered) frequent item list _ *f, c, a, b,m*_ shares a
[4]. common prefix _ *f, c, a*_ with the existing path _ *f, c, a,m, p*_, the count of each node along the prefix is incremented by 1, and one new node (*b*:1) is created and linked as a child of (*a*:2) and another new node (*m*:1) is created and linked as the child of (*b*:1).
[5]. For the third transaction, since its frequent item list _ *f, b*_ shares only the node _ *f* _ with
[6]. the *f* -prefix subtree, *f* 's count is incremented by 1, and a new node (*b*:1) is created and linked as a child of (*f* :3).
[7]. The scan of the fourth transaction leads to the construction of the second branch of the tree, _(*c*:1), (*b*:1), (*p*:1)_.
[8]. For the last transaction, since its frequent item list _ *f, c, a,m, p*_ is identical to the first one, the path is shared with the count of each node along the path incremented by 1.
[9]. To facilitate tree traversal, an item header table is built in which each item points to its first occurrence in the tree via a node-link. Nodes with the same item-name are linked in sequence via such *node-links*. After scanning all the transactions, the tree, together with the associated node-links, are shown in figure 1.

Based on this example, a *frequent-pattern tree* can be designed as follows. *Definition 1* (*FP-tree*). A *frequent-pattern tree* (or *FP-tree* in short) is a tree structure defined below.

[1]. It consists of one root labeled as "*null*", a set of item-prefix sub trees as the children of the root, and a frequent-item-header table.
[2]. Each node in the item-prefix sub tree consists of three fields: *item-name*, *count*, and *node-link*, where *item-name* registers which item this node represents, *count* registers the number of transactions represented by the portion of the path reaching this node, and



node-link links to the next node in the FP-tree carrying the same item-name, or null if
there is none.

[3] Each entry in the frequent-item-header table consists of two fields, (1) *item-name* and (2) *head of node-link* (a pointer pointing to the first node in the FP-tree carrying the *item-name*).Based on this definition, we have the following FP-tree construction algorithm.

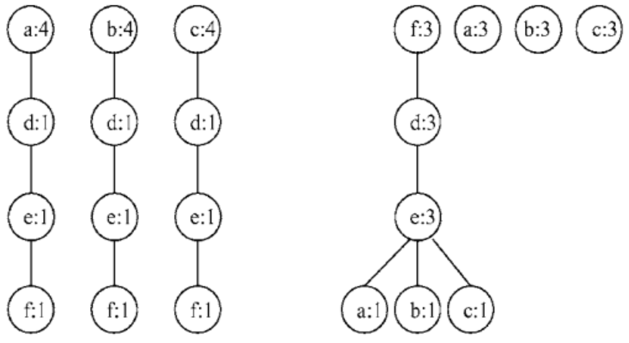
Algorithm 1 (FP-tree construction)**.**
Input: A transaction database *DB* and a minimum support threshold ξ .
Output: FP-tree, the frequent-pattern tree of *DB*.
Method: The FP-tree is constructed as follows.
1. Scan the transaction database *DB* once. Collect *F*, the set of frequent items, and the support of each frequent item. Sort *F* in support-descending order as *FList*, the *list* of frequent items.
2. Create the root of an FP-tree, *T* , and label it as "null". For each transaction *Trans* in *DB* do the following. Select the frequent items in *Trans* and sort them according to the order of *FList*. Let the sorted frequent-item list in *Trans* be [*p* | *P*], where *p* is the first element and *P* is theremaining list. Call *insert tree*([*p* | *P*], *T*). The function *insert tree*([*p* | *P*], *T*) is performed as follows. If *T* has a child *N* such that *N.item-name* = *p.item-name*, then increment *N*'s count by 1; else create a new node *N*, with its count initialized to 1, its parent link linked to *T* , and its node-link linked to the nodes with the same *item-name* via the node-link structure. If *P* is nonempty, call *nserttree*(*P*, *N*) recursively. Analysis. The FP-tree construction takes exactly two scans of the transaction database: The first scan collects the set of frequent items, and the second scan constructs the FP-tree. The cost of inserting a transaction *Trans* into the FP-tree is (|*freq*(*Trans*)|),where*freq*(*Trans*) is the set of frequent items in *Trans*. We will show that the FP-tree contains the complete information for frequent-pattern mining.

Completeness and compactness of FP-tree

There are several important properties of FP-tree that can be derived from the FP-tree construction process. Given a transaction database *DB* and a support threshold ξ . Let *F* be the frequent items in *DB*. For each transaction *T* , *freq*(*T*) is the set of frequent items in *T* , i.e., *freq*(*T*) = *T* ∩ *F*,and is called the *frequent item projection* of transaction *T* . According to the *Apriori* principle, the set of frequent item projections of transactions in the database is sufficient for mining the complete set of frequent patterns, because an infrequent item plays no role in frequent patterns.

Lemma 1. *Given a transaction database DB and a support threshold ξ, the complete set of frequent item projections of transactions in the database can be derived from DB's FP-tree.*Rationale. Based on the FP-tree construction process, for each transaction in the *DB*, its frequent item projection is mapped to one path in the FP-tree.For a path *a*1*a*2 . . . *ak* from the root to a node in the FP-tree, let *cak* be the count at the node labeled *ak* and *c_ak* be the sum of counts of children nodes of *ak* . Then, according to the construction of the FP-tree, the path registers frequent item projections of *cak*− *c_ak* transactions.Therefore, the FP-tree registers the complete set of frequent item projections without duplication.Based on this lemma, after an FP-tree for *DB* is constructed, it contains the complete information for mining frequent patterns from the transaction database. Thereafter, only the FP-tree is needed in the remaining mining process, regardless of the number and length of the frequent patterns.

Lemma 2. *Given a transaction database DB and a support threshold ξ. Without consideringthe* (*null*) *root, the size of an FP-tree is bounded by _T∈DB |freq*(*T*)|, *and the height of the tree is bounded by* max*T∈DB*{|*freq*(*T*)|}, *where freq*(*T*) *is the frequent item projection of transaction T* Rationale. Based on the FP-tree construction process, for any transaction *T* in *DB*, there exists a path in the FP-tree starting from the corresponding item prefix subtree so that the set of nodes in the path is exactly the same set of frequent items in *T* . The root is the only extra node that is not created by frequent-item insertion, and each node contains one node-link and one count. Thus we have the bound of the size of the tree stated in the Lemma.The height of any *p*-prefix subtree is the maximum number of frequent items in any transaction with *p* appearing at the head of its frequent item list. Therefore, the height of the tree is bounded by the maximal number of frequent items in any transaction in the database, if we do not consider the additional level added by the root.Lemma 2.2 shows an important benefit of FP-tree: the size of an FP-tree is bounded by the size of its corresponding database because each transaction will contribute at most one path to the FP-tree, with the length equal to the number of frequent items in that transaction. Since there are often a lot of sharings of frequent items among transactions, the size of the tree is usually much smaller than its original database. Unlike the *Apriori*-like method which may generate an exponential number of candidates in the worst case, under no circumstances, may an FP-tree with an exponential number of nodes be generated.FP-tree is a highly compact structure which stores the information for frequent-patternmining. Since a single path "$a1 \rightarrow a2 \rightarrow \cdots \rightarrow an$" in the *a*1-prefix subtree registers all the transactions whose maximal frequent set is in the form of "$a1 \rightarrow a2 \rightarrow \cdots \rightarrow ak$" for any $1 \leq k \leq n$, the size of the FP-tree is substantially smaller than the size of the database and that of the candidate sets generated in the association rule mining.The items in the frequent item set are ordered in the support-descending order: More frequently occurring items are more likely to be shared and thus they are arranged closer to the top of the FP-tree. This ordering enhances the compactness of the FP-tree structure. However, this does not mean that the tree so constructed *always* achieves the maximal compactness. With the knowledge of particular data characteristics, it is sometimes possible to achieve even better compression than the frequency-descending ordering. Consider the following example. Let the set of transactions be: {*adef* , *bdef* , *cdef* , *a*, *a*, *a*, *b*, *b*, *b*, *c*, *c*, *c*}, and the minimum support threshold be 3. The frequent item set associated with support count becomes {*a*:4, *b*:4, *c*:4, *d*:3, *e*:3, *f* :3}. Following the item frequency ordering $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f$, the FP-tree constructed will contain 12 nodes, as shown in figure 2(a). However, following another item ordering $f \rightarrow d \rightarrow e \rightarrow a \rightarrow b \rightarrow c$, it will contain only 9 nodes, as shown in figure 2(b).The compactness of FP-tree is also verified by our experiments. Sometimes a rather small FP-tree is resulted from a quite large database. For example, for the database *Connect-4* used in *MaxMiner* (Bayardo, 1998), which contains 67,557 transactions with 43 items in each transaction, when the support threshold is 50% (which is used in the *MaxMiner e*xperiments (Bayardo, 1998)), the total number of occurrences of frequent items is 2,219,609, whereas the total number of nodes in the FP-tree is 13,449 which represents a reduction ratio of 165.04, while it still holds hundreds of thousands of frequent patterns! (Notice that for databases with mostly short transactions, the reduction ratio is not that high.)



a) FPtree follows the support ordering b) FPtree does not follow the support ordering

Therefore,it is not surprising some gigabyte transaction database containing many long patterns may even generate an FP-tree that fits in main memory. Nevertheless, one cannot assume that an FP-tree can always fit in main memory no matter how large a database is. Methods for highly scalable *FP-growth* mining will be discussed in Section 5.

4.1 Mining frequent patterns using FP-tree: Construction of a compact FP-tree ensures that subsequent mining can be performed with a rather compact data structure. However, this does not automatically guarantee that it will be highly efficient since one may still encounter the combinatorial problem of candidate generation if one simply uses this FP-tree to generate and check all the candidate patterns. In this section, we study how to explore the compact information stored in an FP-tree, develop the principles of frequent-pattern growth by examination of our running example, explore how to perform further optimization when there exists a single prefix path in an FP-tree, and propose a frequent-pattern growth algorithm, *FP-growth*, for mining the *complete set of frequent patterns* using FP-tree. *4.1.1 Principles of frequent-pattern growth for FP-tree mining* In this subsection, we examine some interesting properties of the FP-tree structure which will facilitate frequent-pattern mining.

Property 1 (*Node-link property*)**.** *For any frequent item ai , all the possible patterns containing only frequent items and ai can be obtained by following ai 's node-links, starting from ai 's head in the FP-tree header.*This property is directly from the FP-tree construction process, and it facilitates the access of all the frequent-pattern information related to *ai* by traversing the FP-tree once following *ai* 's node-links.To facilitate the understanding of other properties of FP-tree related to mining, we first go through an example which performs mining on the constructed FP-tree (figure 1) in Example 1. *Example 2*. Let us examine the mining process based on the constructed FP-tree shown in figure 1. Based on Property 3.1, all the patterns containing frequent items that a node *ai*participates can be collected by starting at *ai* 's node-link head and following its node-links.
We examine the mining process by starting from the bottom of the node-link header table. For node *p*, its immediate frequent pattern is (*p*:3), and it has two paths in the FP-tree:_ *f* :4, *c*:3, *a*:3,*m*:2, *p*:2_ and _*c*:1, *b*:1, *p*:1_. The first path indicates that string"(*f, c, a,m, p*)" appears twice in the database. Notice the path also indicates that string _ *f, c, a*_ appears three times and _ *f* _ itself appears even four times. However, they only appear twice *together* with *p*. Thus, to study which string appear together with *p*, only *p*'s prefix path _ *f* :2, *c*:2, *a*:2,*m*:2_ (or simply, _ *f cam*:2_) counts. Similarly, the second path indicates string "(*c, b, p*)" appears once in the set of transactions in *DB*, or *p*'s prefix pathis _*cb*:1_. These two prefix paths of *p*, "*{*(*f cam*:2), (*cb*:1)*}*", form *p*'s subpattern-base, which is called *p*'s conditional pattern base (i.e., the subpattern-base under the condition of *p*'s existence). Construction of an FP-tree on this conditional pattern-base (which is called *p*'s conditional FP-tree) leads to only one branch (*c*:3). Hence, only one frequent pattern (*cp*:3) is derived. (Notice that a pattern is an itemset and is denoted by a string here.) The search for frequent patterns associated with *p* terminates. For node *m*, its immediate frequent pattern is (*m*:3), and it has two paths, _ *f* :4, *c*:3, *a*:3,*m*:2_ and _ *f* :4, *c*:3, *a*:3, *b*:1, *m*:1_. Notice *p* appears together with *m* as well, however, there is no need to include *p* here in the analysis since any frequent patterns involving *p* has been analyzed in the previous examination of *p*. Similar to the above analysis, *m*'s conditional pattern-base is *{*(*fca*:2), (*fcab*:1)*}*. Constructing an FP-tree on it, we derive *m*'s conditional FP-tree, _ *f* :3, *c*:3, *a*:3_, a single frequent pattern path, as shown in figure 3. This conditional FP-tree is then mined recursively by calling *mine*(_ *f* :3, *c*:3, *a*:3_ |*m*). Figure 3 shows that "*mine*(_ *f* :3, *c*:3, *a*:3_ |*m*)" involves mining three items (*a*), (*c*), (*f*) in sequence. The first derives a frequent pattern (*am*:3), a conditional pattern-base *{*(*fc*:3)*}*, and then a call "*mine*(_ *f* :3, *c*:3_ | *am*)"; the second derives a frequent pattern (*cm*:3), a conditional pattern-base *{*(*f* :3)*}*, and then a call "*mine*(_ *f* :3_ | *cm*)"; and the third derives only a frequent pattern (*fm*:3). Further recursive call of "*mine*(_ *f* :3, *c*:3_ | *am*)" derives two patterns (*cam*:3) and (*fam*:3), and a conditional pattern-base *{*(*f* :3)*}*, which then leads to a call "*mine*(_ *f* :3_ | *cam*)", that derives the longest pattern (*fcam*:3). Similarly, the call of "*mine*(_ *f* :3_ | *cm*)" derives one pattern (*fcm*:3). Therefore, the set of frequent patterns involving *m* is *{*(*m*:3), (*am*:3), (*cm*:3), (*f m*:3), (*cam*:3), (*fam*:3), (*fcam*:3), (*fcm*:3)*}*. This indicates that *a single path FP-tree can be mined by outputting all the combinations of the items in the path*.Similarly, node *b* derives (*b*:3) and it has three paths: _ *f* :4, *c*:3, *a*:3, *b*:1_, _ *f* :4, *b*:1_, and _*c*:1, *b*:1_. Since *b*'s conditional pattern-base *{*(*fca*:1), (*f* :1), (*c*:1)*}* generates no frequent item, the mining for *b* terminates. Node *a* derives one frequent pattern *{*(*a*:3)*}* and one subpattern base *{*(*f c*:3)*}*, a single-path conditional FP-tree. Thus, its set of frequent pattern

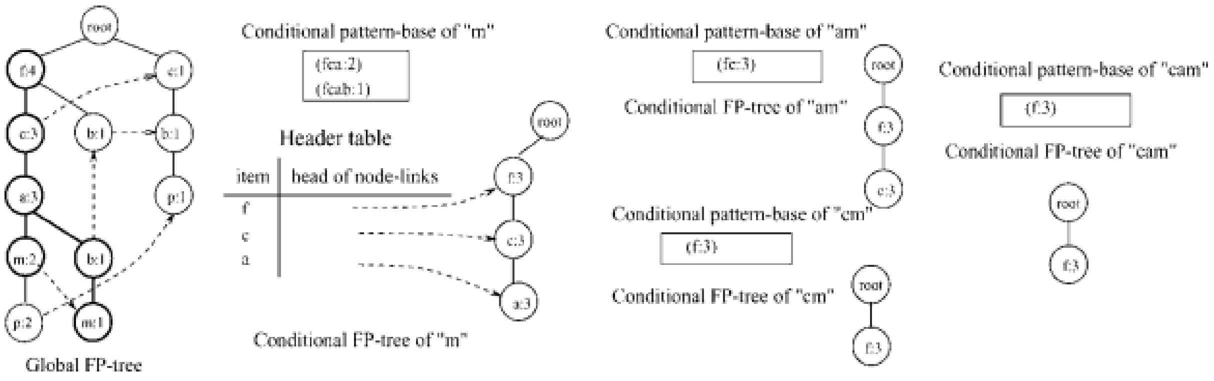

root
f:4
c:1
c:3
b:1
b:1
a:3
p:1
m:2
b:1
p:2
m:1
Global FP-tree
Conditional pattern-base of "m"
(fca:2)
(fcab:1)
Header table
item
head of node-links
f
c
a
root
f:3
c:3
a:3
Conditional FP-tree of "m"
Conditional pattern-base of "am"
(fc:3)
Conditional FP-tree of "am"
root
f:3
c:3
Conditional pattern-base of "cm"
(f:3)
Conditional FP-tree of "cm"
root
f:3
Conditional pattern-base of "cam"
(f:3)
Conditional FP-tree of "cam"
root
f:3

Table 2. Mining frequent patterns by creating conditional (sub)pattern-bases.

Item	Conditional pattern-base	Conditional FP-tree
P	{(*f cam*:2), (*cb*:1)}	{(*c*:3)}|*p*
M	{(*f ca*:2), (*fcab*:1)}	{(*f* :3, *c*:3, *a*:3)}|*m*
b	{(*f ca*:1), (*f* :1), (*c*:1)}	∅
a	{(*f c*:3)}	{(*f* :3, *c*:3)}|*a*
c	{(*f* :3)}	{(*f* :3)}|*c*
f	∅	∅

can be generated by taking their combinations. Concatenating them with (*a*:3), we have *{*(*f a*:3), (*ca*:3), (*fca*:3)*}*. Node *c* derives (*c*:4) and one subpattern-base *{*(*f* :3)*}*, and the set of frequent patterns associated with (*c*:3) is *{*(*fc*:3)*}*. Node *f* derives only (*f* :4) but no conditional pattern-base.The conditional pattern-bases and the conditional FP-trees generated are summarized in Table 2.The correctness and completeness of the process in Example 2 should be justified.This is accomplished by first introducing a few important properties related to the mining process.

Property 2 (*Prefix path property*)**.** *To calculate the frequent patterns with suffix ai , onlythe* prefix subpathes *of nodes labeled ai in the FP-tree need to be accumulated, and the frequency count of every node in the prefix path should carry the same count as that in the corresponding node ai in the path.*Rationale. Let the nodes along the path *P* be labeled as *a*1, . . . , *an* in such an order that *a*1 is the root of the prefix subtree, *an* is the leaf of the subtree in *P*, and *ai* (1 ≤ *i* ≤ *n*) is the node being referenced. Based on the process of FP-tree construction presented in Algorithm 1, for each prefix node *ak* (1 ≤ *k* < *i*), the prefix subpath of the node *ai* in *P* occurs together with *ak* exactly *ai .count* times. Thus every such prefix node should carry the same count as node *ai* . Notice that a postfix node *am* (for *i* < *m* ≤ *n*) along the same path also co-occurs with node *ai*. However, the patterns with *am* will be generated when examining the suffix node *am*, enclosing them here will lead to redundant generation of the patterns that would have been generated for *am*. Therefore, we only need to examine the prefix subpath of *ai* in *P*. For example, in Example 2, node *m* is involved in a path _ *f* :4, *c*:3, *a*:3,*m*:2, *p*:2_, to calculate the frequent patterns for node *m* in this path, only the prefix subpath of node *m*, which is _ *f* :4, *c*:3, *a*:3_, need to be extracted, and the frequency count of every node in the prefix path should carry the same count as node *m*. That is, the node counts in the prefix path should be adjusted to _ *f* :2, *c*:2, *a*:2_. Based on this property, the prefix subpath of node *ai* in a path *P* can be copied and transformed into a count-adjusted prefix subpath by adjusting the frequency count of every node in the prefix subpath to the same as the count of node *ai* . The prefix path so transformed is called the *transformed prefix path* of *ai* for path *P*. Notice that the set of transformed prefix paths of *ai* forms a small database of patterns which co-occur with *ai* . Such a database of patterns occurring with *ai* is called *ai 's conditional pattern-base*, and is denoted as "*pattern base* | *ai* ". Then one can compute all the frequent patterns associated with *ai* in this *ai* -conditional pattern-base by creating a small FP-tree, called *ai 's conditional FP-tree* and denoted as "FP-tree | *ai* ". Subsequent mining can be performed on this small conditional FP-tree. The processes of construction of conditional pattern-bases and conditional FP-trees have been demonstrated in Example 2. This process is performed recursively, and the frequent patterns can be obtained by a pattern-growth method, based on the following lemmas and corollary.

Lemma 1 (*Fragment growth*)**.** *Let α be an itemset in DB, B be α's conditional patternbase, and β be an itemset in B. Then the support of α ∪β in DB is equivalent to the support of β in B.* Rationale. According to the definition of conditional pattern-base, each (sub)transaction in *B* occurs under the condition of the occurrence of *α* in the original transaction database *DB*. If an itemset *β* appears in *B ψ* times, it appears with *α* in *DBψ* times as well. Moreover, since all such items are collected in the conditional pattern-base of *α*, *α* ∪ *β* occurs exactly *ψ* times in *DB* as well. Thus we have the lemma. From this lemma, we can directly derive an important corollary.
Corollary 1 (*Pattern growth*)**.** *Let α be a frequent itemset in DB, B be α's conditional pattern-base, and β be an itemset in B. Then α ∪ β is frequent in DB if and only if β is frequent in B.* Based on Corollary 3.1, mining can be performed by first identifying the set of frequent 1-itemsets in *DB*, and then for each such frequent 1-itemset, constructing its conditional pattern-bases, and mining its set of frequent 1-itemsets in the conditional pattern-base, and so on. This indicates that the process of mining frequent patterns can be viewed as first mining frequent 1-itemset and then progressively growing each such itemset by mining its conditional pattern-base, which can in turn be done similarly. By doing so, a frequent *k*-itemset mining problem is successfully transformed into a sequence of *k* frequent 1- itemset mining problems via a set of conditional pattern-bases. Since mining is done by pattern growth, there is no need to generate any candidate sets in the entire mining process. Notice also in the construction of a new FP-tree from a *conditional pattern-base* obtained during the mining of an FP-tree, the items in the frequent itemset should be ordered in the frequency descending order of *node occurrence* of each item instead of its *support* (which represents item occurrence). This is because each node in an FP-tree may represent many occurrences of an item but such a node represents a single unit (i.e., the itemset whose elements always occur together) in the construction of an item-associated FP-tree.

3.2. Frequent-pattern growth with single prefix path of FP-tree

The frequent-pattern growth method described above works for all kinds of FP-trees. However, further optimization can be explored on a special kind of FP-tree, called *single prefixpathFP-tree*, and such an optimization is especially useful at mining long frequent patterns. A single prefix-path FP-tree is an FP-tree that consists of only a single path or a single prefix path stretching from the root to the first branching node of the tree, where a *branching node* is a node containing more than one child. Let us examine an example.
***Example 3*.** Figure 4(a) is a single prefix-path FP-tree that consists of one prefix path,_(*a*:10)→(*b*:8)→(*c*:7)_, stretching from the root of the tree to the first branching node (*c*:7).Although it can be mined using the frequent-pattern growth method described above, a bettermethod is to split the tree into two fragments: the single prefix-path, _(*a*:10)→(*b*:8)→(*c*:7)_, as shown in figure 4(b), and the multipath part, with the root replaced by a pseudoroot *R*, as shown in figure 4(c). These two parts can be mined separately and then combined together.Let us examine the two separate mining processes. All the frequent patterns associated with the first part, the single prefix-path *P* = _(*a*:10)→(*b*:8)→(*c*:7)_, can be mined by enumeration of all the combinations of the subpaths of *P* with the support set to the minimum support of the items contained in the subpath. This is because each such subpath is distinct and occurs the same number of times as the *minimum occurrence frequency among the items in the subpath* which is equal to the support of the last item in the subpath. Thus, path *P* generates the following set of frequent patterns, *freq pattern set*(*P*) = {(*a*:10), (*b*:8), (*c*:7), (*ab*:8), (*ac*:7), (*bc*:7), (*abc*:7)}.Let *Q* be the second FP-tree (figure 4(c)), the multipath part rooted with *R*. *Q* can be mined as follows.First, *R* is treated as a *null* root, and *Q* forms a multipath FP-tree, which can be mined using a typical frequent-pattern growth method. The mining result is: *freq pattern set*(*Q*)= {(*d*:4), (*e*:3), (*f* :3), (*d f* :3)}. *Figure*

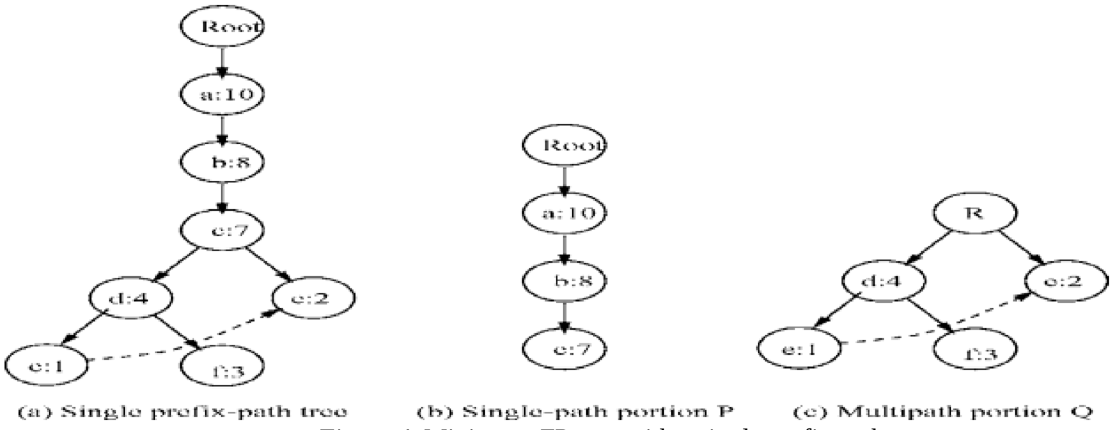


Figure 4. Mining an FP-tree with a single prefix path.

Second, for each frequent itemset in *Q*, *R* can be viewed as a conditional frequent pattern-base, and each itemset in *Q* with each pattern generated from *R* may form a distinct frequent pattern. For example, for (*d*:4) in *freq pattern set*(*Q*), *P* can be viewed as its conditional pattern-base, and a pattern generated from *P*, such as (*a*:10), will generate with it a new frequent itemset, (*ad*:4), since *a* appears together with *d* at most four times. Thus, for (*d*:4) the set of frequent patterns generated will be (*d*:4)×*freq pattern set*(*P*) = {(*ad*:4), (*bd*:4), (*cd*:4), (*abd*:4), (*acd*:4), (*bcd*:4), (*abcd*:4)}, where *X* × *Y* means that every pattern in *X* is combined with everyone in *Y* to form a "cross-product-like" larger itemset with the support being the minimum support between the two patterns. Thus, the complete set of frequent patterns generated by combining the results of *P* and *Q* will be *freq pattern set*(*Q*)×*freq pattern set*(*P*), with the support being the support of the itemset in *Q* (which is always no more than the support of the itemset from *P*). In summary, the set of frequent patterns generated from such a single prefix path consists of three distinct sets: (1) *freq pattern set*(*P*), the set of frequent patterns generated from the single prefix-path, *P*; (2) *freq pattern set*(*Q*), the set of frequent patterns generated from the multipath part of the FP-tree, *Q*; and (3) *freq pattern set*(*Q*)×*freq pattern set*(*P*), the set of frequent patterns involving both parts. We first showif an FP-tree consists of a single path *P*, one can generate the set of frequent patterns according to the following lemma.

Lemma 2 (*Pattern generation for an FP-tree consisting of single path*)**.** *Suppose an FP-tree T consists of a single path P. The complete set of the frequent patterns of T can be generated by enumeration of all the combinations of the subpaths of P with the support being the minimum support of the items contained in the subpath.* Rationale. Let the single path *P* of the FP-tree be _*a*1:*s*1→*a*2:*s*2→ ·· ·→*ak* :*sk* _. Since the FP-tree contains a single path *P*, the support frequency *si* of each item *ai* (for 1 ≤ *i* ≤ *k*) is the frequency of *ai* co-occurring with its prefix string. Thus, any combination of the items in the path, such as _*ai* , . . . , *aj* _ (for 1 ≤ *i*, *j* ≤ *k*), is a frequent pattern, with their cooccurrence frequency being the minimum support among those items. Since every item in each path *P* is unique, there is no redundant pattern to be generated with such a combinational generation. Moreover, no frequent patterns can be generated outside the FP-tree. Therefore, we have the lemma. We then show if an FP-tree consists of a single prefix-path, the set of frequent patterns can be generated by splitting the tree into two according to the following lemma.

Lemma 3 (*Pattern generation for an FP-tree consisting of single prefix path*)**.** *Suppose an FP-tree T, similar to the tree in figure* 4(a), *consists of* (1) *a single prefix path P, similar to the tree P in figure* 4(b), *and* (2) *the multipath part, Q, which can be viewed as an independent FP-tree with a pseudo-root R, similar to the tree Q in figure* 4(c). *The complete set of the frequent patterns of T consists of the following three portions:*

1. *The set of frequent patterns generated from P by enumeration of all the combinations of the items along path P, with the support being the minimum support among all the items that the pattern contains.*
2. *The set of frequent patterns generated from Q by taking root R as "null."*
3. *The set of frequent patterns combining P and Q formed by taken the cross-product of the frequent patterns enerated from P and Q, denoted as freq pattern set*(*P*) × *freq pattern set*(*Q*), *that is, each frequent itemset is the union of one frequent itemset from P and one from Q and its support is the minimum one between the supports of the two itemsets.* Rationale. Based on the FP-tree construction rules, each node *ai* in the single prefix path of the FP-tree appears only once in the tree. The single prefix-path of the FP-tree forms a new FP-tree *P*, and the multipath part forms another FP-tree *Q*. They do not share nodes representing the same item. Thus, the two FP-trees can be mined separately. First, we show that each pattern generated from one of the three portions by llowing the pattern generation rules is distinct and frequent. According to Lemma 3.2, each pattern generated from *P*, the FP-tree formed by the single prefix-path, is distinct and frequent. The set of frequent patterns generated from *Q* by taking root *R* as "null" is also distinct and frequent since such patterns exist without combining any items in their conditional databases (which are in the items in *P*. The set of frequent patterns generated by combining *P* and *Q*, that is, taking the cross-product of the frequent patterns generated from *P* and *Q*, with the support being the minimum one between the supports of the two itemsets, is also distinct and frequent. This is because each frequent pattern generated by *P* can be considered as a frequent pattern in the conditional pattern-base of a frequent item in *Q*, and whose support should be the minimum one between the two supports since this is the frequency that both patterns appear together.Second, we show that no patterns can be generated out of this three portions. Sinceaccording to Lemma 3.1, the FP-tree *T* without being split into two FP-trees *P* and *Q* generatesthe complete set of frequent patterns by pattern growth. Since each pattern generated from *T* will be generated from either the portion *P* or *Q* or their combination, the method generates the complete set of frequent patterns. *The frequent-pattern growth algorithm* Based on the above lemmas and properties, we have the following algorithm for mining frequent patterns using FP-tree.

Algorithm 2 (FP-growth*: Mining frequent patterns with FP-tree by pattern fragment growth*)**.**

Input: A database *DB*, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ξ .

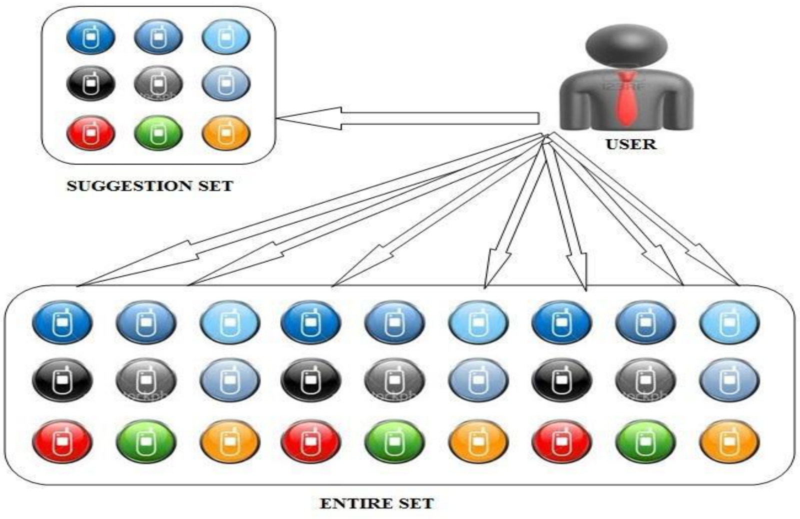
Output: The complete set of frequent patterns.68 HAN ET AL.
Method: *call FP-growth*(FP-tree, *null*).
Procedure *FP-growth*(*Tree, α*)

```
{
(1) if Tree contains a single prefix path // Mining single prefix-path FP-tree
(2) then {
(3) let P be the single prefix-path part of Tree;
(4) let Q be the multipath part with the top branching node replaced by a null root;
(5) for each combination (denoted as $\beta$) of the nodes in the path P do
(6) generate pattern $\beta \cup \alpha$ with support = minimum support of nodes in $\beta$;
(7) let freq pattern set(P) be the set of patterns so generated; }
(8) else let Q be Tree;
(9) for each item ai in Q do { // Mining multipath FP-tree
(10) generate pattern $\beta = ai \cup \alpha$ with support = ai .support;
(11) construct $\beta$'s conditional pattern-base and then $\beta$'s conditional FP-tree Tree$\beta$ ;
(12) if Tree$\beta = \emptyset$
(13) then call FP-growth(Tree$\beta$, $\beta$);
(14) let freq pattern set(Q) be the set of patterns so generated; }
(15) return(freq pattern set(P) ∪ freq pattern set(Q) ∪ (freq pattern set(P)
×freq pattern set(Q)))
}
```

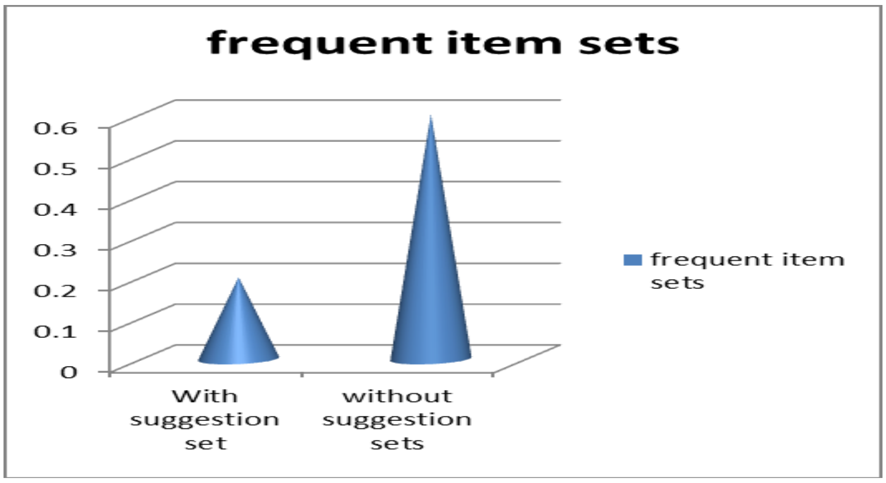
Analysis. With the properties and lemmas in Sections 2 and 3, we show that the algorithm correctly finds the complete set of frequent itemsets in transaction database *DB*. As shown in Lemma 2.1, FP-tree of *DB* contains the complete information of *DB* in relevance to frequent pattern mining under the support threshold ξ . If an FP-tree contains a single prefix-path, according to Lemma 3.3, the generation of the complete set of frequent patterns can be partitioned into three portions: the single prefix-path portion *P*, the multipath portion *Q*, and their combinations. Hence we have lines (1)-(4) and line (15) of the procedure. According to Lemma 3.2, the generated patterns for the single prefix path are the enumerations of the subpaths of the prefix path, with the support being the minimum support of the nodes in the subpath. Thus we have lines (5)-(7) of the procedure.
After that, one can treat the multipath portion or the FP-tree that does not contain the single prefix-path as portion *Q* (lines (4) and (8)) and construct conditional pattern-base and mine its conditional FP-tree for each frequent itemset *ai* . The correctness and completeness of the prefix path transformation are shown in Property 3.2. Thus the conditional pattern-bases store the complete information for frequent pattern mining for *Q*. According to Lemmas 3.1 and its corollary, the patterns successively grown from the conditional FP-trees are the set of sound and complete frequent patterns. Especially, according to the fragment growth property, the support of the combined fragments takes the support of the frequent itemsets generated in the conditional pattern-base. Therefore, we have lines (9)-(14) of the procedure. Line (15) sums up the complete result according to Lemma 3.3. Let's now examine the efficiency of the algorithm. The *FP-growth* mining process scans the FP-tree of *DB* once and generates a small pattern-base *Bai* for each frequent item *ai* , each consisting of the set of transformed prefix paths of *ai* . Frequent pattern mining is then recursively performed on the small pattern-base *Bai* by constructing a conditional FP-tree for *Bai*. As reasoned in the analysis of Algorithm 1, an FP-tree is usually much smaller than the size of *DB*. Similarly, since the conditional FP-tree, "FP-tree | *ai* ", is constructed on the pattern-base *Bai* , it should be usually much smaller and never bigger than *Bai* . Moreover, a pattern-base *Bai* is usually much smaller than its original FP-tree, because it consists of the transformed prefix paths related to only one of the frequent items, *ai* . Thus, each subsequent mining process works on a set of usually much smaller pattern-bases and conditional FPtrees. Moreover, the mining operations consist of mainly prefix count adjustment, counting local frequent items, and pattern fragment concatenation. This is much less costly than generation and test of a very large number of candidate patterns. Thus the algorithm is efficient.
From the algorithm and its reasoning, one can see that the *FP-growth* mining process is a divide-and-conquer process, and the scale of shrinking is usually quite dramatic. If the shrinking factor is around 20-100 for constructing an FP-tree from a database, it is expected to be another hundreds of times reduction for constructing each conditional FP-tree from its already quite small conditional frequent pattern-base. Notice that even in the case that a database may generate an exponential number of frequent patterns, the size of the FP-tree is usually quite small and will never grow exponentially. For example, for a frequent pattern of length 100, "$a1$, . . . , $a100$", the FP-tree construction results in only one path of length 100 for it, possibly "_$a1,\rightarrow\cdots\rightarrow a100$_" (if the items are ordered in the *list* of frequent items as $a1$, . . . , $a100$). The *FP-growth* algorithm will still generate about 1030 frequent patterns (if time permits!!), such as "$a1$, $a2$, . . ., $a1a2$,. . ., $a1a2a3$, . . ., $a1$ . . . $a100$." However, the FP-tree contains only one frequent pattern path of 100 nodes, and according to Lemma 3.2, there is even no need to construct any conditional FP-tree in order to find all the patterns.

V. RESULTS

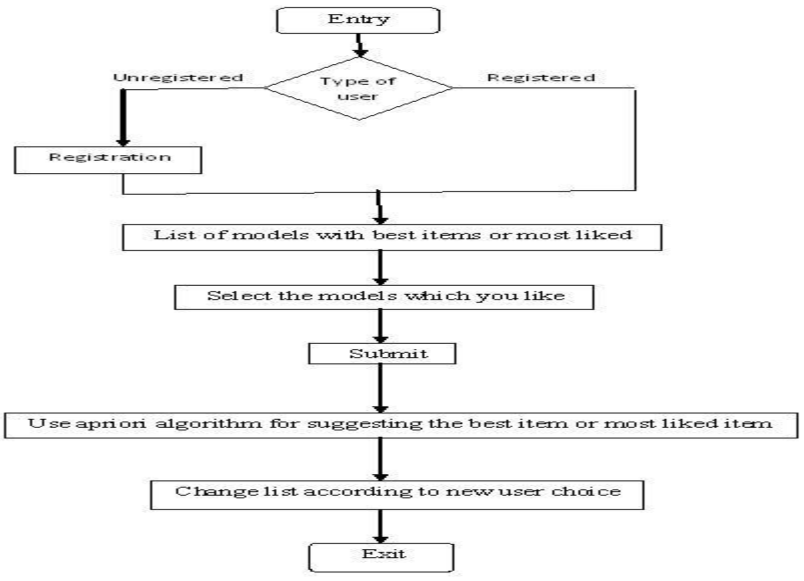
The above is the best method of ranking and suggesting the best methods in the scenario of mobile phone outlets in INDIA, which is shown in the following diagram:



As it was shown in the above diagram we were going to take the most liked items from the users and suggesting the best mobiles or the best set of suggestions that the most of the users liked or ordered.



The confidence of the suggestions were also proved by an traditional confidence calculations as follows In this section we are going to discuss about algorithms. Till now we have discussed some ranking rules , suggestion rules and Frequency move2set algorithm. We have some problems with these, so we go for an algorithm which suits our requirements well. The algorithm is Apriori algorithm. In order to know these algorithms we need to know some concepts of data mining.


Entry
Type of user
Unregistered
Registered
Registration
List of models with best items or most liked
Select the models which you like
Submit
Use apriori algorithm for suggesting the best item or most liked item
Change list according to new user choice
Exit

Frequent itemsets: Let I={I1, I2, I3,…., Im} be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that T is a subset of I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A is a subset of T. An association rule is an implication of the form A > B, where A is subset of I, B is subset of I and A∩B =Ø. The rule A > B holds in the transaction set D with support s, where s is the percentage of transactions in D that contain AUB. This is taken to be the probability ,P(AUB).The rule A > B has confidence c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B/A). That is, Support(A=>B) = P(AUB) Confidence(A=>B) = P(B/A) Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called strong. The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply as the frequency, support count,or count of the itemset. The set of frequent k-itemset is commonly denoted by Lk. confidence(A > B) = P (A / B) = support(AUB) / support(A) = supportcount(AUB) / supportcount(A).

Mining frequent itemsets: In general, association rule mining can be viewed as a two-step process: 1. Finding all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min-sup. 2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence

VI. CONCLUSION

All the previous process already proposed were very complex and contains very complicated computations which made the ranking and suggesting the best and popular items have been more and more complex and not getting to the actual end users. Now we have proposed as very simple randomized algorithm for ranking and suggesting popular items designed to account for popularity bias. This was utilized by many of the mobile outlets in the country successfully.

REFERENCES

[1]. Huidrom Romesh Chandra Singh, T. kalaikumaran, Dr. S. Karthik, Suggestion of True Popular Items, IJCSE, 2010.
[2]. Y.Maanasa, V.Kumar, P.Satish Babu, Framework for suggesting POPULAR ITEMS to users by Analyzing Randomized Algorithms, IJCTA, 2011.
[3]. V. Anantharam, P. Varaiya, and J. Walrand, ―Asymptotically Efficient Allocation Rules for the Multiarmed Bandit Problem with Multiple Plays—Part i: i.i.d. Rewards,‖ IEEE Trans. Automatic Control, vol. 32, no. 11, pp. 968-976, Nov. 1987.
[4]. J.R. Anderson, ―The Adaptive Nature of Human Categorization‖ Psychological Rev., vol. 98, no. 3, pp. 409-429, 1991.
[5]. Yanbin Ye, Chia-Chu Chiang, A Parallel Apriori Algorithm for Frequent Itemsets Mining, IEEE, 2006.
[6]. Cong-Rui Ji, Zhi-Hong Deng, Mining Frequent Ordered Patterns without Candidate Generation.
[7]. Huang Chiung-Fen, Tsai Wen-Chih, Chen An-Pin, Application of new Apriori Algorithm MDNC to Exchange Traded Fund, International Conference on Computational Science and Engineering, 2009.
[8]. Milan Vojnovi_c, James Cruise, Dinan Gunawardena, and Peter Marbach, Ranking and Suggesting Popular Items, IEEE, 2009.