International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Incremental CFP-Tree Optimization For Efficient Representative Pattern Set Mining

Vivek Satpute[1], Prof. Digambar Padulkar[2]

[1, 2] Savitribai Phule Pune University, *Department of Computer Engineering, VPCOE, Baramati, Maharashtra, India.*

Abstract: ***In the area of data mining frequent pattern mining is a significant hitch. Frequent pattern mining is more often performed on a transaction database that contains set of items. A pattern is frequent pattern when it has bigger support than user define threshold. For mining frequent patterns numerous capable algorithms have been developed. However, RPglobal is awfully time-consuming and space-consuming. Barely it becomes realistic when the number of frequent patterns is not large. RPlocal is dreadfully efficient, although it produces extra representative patterns than RPglobal. Here, two algorithms MinRPset and FlexRPset are in picture. Algorithm MinRPset is comparable to RPglobal, but it utilizes numerous techniques to diminish the running time and memory usage. CFP-tree structure is used in MinRPset, it is a tree structure. FlexRPset provides one parameter K in addition, which allows users to build a swap between efficiency and the number of representative patterns that he has chosen. Use of the techniques is supportive to improve the effectiveness of MinRPset by considering closed patterns only and by using a structure called CFP-tree to find C(X)s efficiently.***

Keywords: pattern, closed pattern, covering pattern, representative pattern

1. Introduction

Representative pattern is a pattern that covers the all patterns of the cluster from that it belongs[1]. Mining of frequent pattern was initially put forwarded by Agrawal et al.[2] in1993 for market basket analysis in the form of association rule mining. That examines buying behavior of customer by discovering associations between the different items that customers buys from market and places it in their shopping baskets. In data mining, frequent pattern mining is an essential problem. Frequent itemsets carry necessary role in lots of data mining assignment that aims to discover attractive patterns from databases. Technically, frequent pattern mining is the method of discovering relationships or patterns among massive databases. Pattern mining is not only the extraction of unseen analytical information from huge databases, but it is a amazing technology with enormous potential to assist companies to focus on the most significant information into their data warehouses. Pattern mining is the technique that is able to respond business queries that traditionally were very time consuming to solve. Discovering the frequent patterns has specific importance in mining associations, correlations and different other motivating relationships among data. Additionally, it also helps out in data indexing, clustering, classification, and many other tasks of data mining.

After clustering there may have lots of frequent patterns. And those patterns are of huge in numbers, like in thousand, even sometimes millions in numbers. As it is depends on threshold that user sets, it may differ for different threshold. These patterns again become overhead some times for understanding relations among them and for further processing too. So there is need of something that will become solution for this. Then the approach of one representative pattern for one cluster comes in picture. This pattern covers all remaining patterns of the cluster. It will become a representative of all patterns that are in same cluster. And further try to make these representative patterns as less as possible. So these few can be representative of huge. The minimum number of these representative pattern then can be easy to process, easy to understand and easy to handle. Ultimately finding representative pattern from frequent pattern is now center of attraction.

2. Literature Survey

Xin et al. [3] proposed concept of δ-covered to make simpler the concept of frequent closed pattern. The aim is to discover a minimum set of representative patterns that can δ- cover all frequent patterns. They conclude that the set cover problem can be relate to the main problem, and they build up two algorithms, RPglobal and RPlocal. RPglobal first generates the set of patterns that can be δ-covered by each pattern, and then employs the well-known greedy algorithm for the set cover problem to discover representative patterns. First, both RPglobal and RPlocal are clever to discover a subset of representative patterns; second, even if RPlocal gives extra patterns than RPglobal, the feat of RPlocal is awfully close to RPglobal. Nearly all the outputs of RPlocal are contained by two times of RPglobal. The outcome of RPglobal is partial as minimum support becomes near to the ground, the number of closed patterns grows awfully speedy, and the running times of RPglobal go beyond the time limit (30 minutes). RPglobal does not sizes in good health w.r.t. lot of patterns, also it takes more time than RPlocal. Moreover RPglobal is very slow and requires much more space. It is only usable when number of frequent patterns are less. RPlocal is constructed on beam of FPclose [4]. It integrates frequent pattern mining with representative pattern finding. RPlocal is very efficient, but it produces more representative patterns than RPglobal. Yan et al. [5] apply profiles to summarize patterns. A profile consists of a master pattern, a support and a probability distribution vector, which contains the probability of the items in the master pattern. The set of patterns represented by a profile are subsets of the master pattern, and their support is designed by multiplying the support of the profile and the likelihood of the related items. To summarize a group of

patterns with k profiles, they divided the patterns into k clusters, and apply a profile to depict each cluster. But it makes opposing assumptions. Poernomo et al[6] employed conditional independence to reduce restoration error. It appended an additional factor to every profile that is a pattern base, and then newfangled profile is called c-profile that is conditional profile. C-profile is actually itemset profile, extended from a base denoting the form where the individual summary is applicable. The cp-summary contain a record of c- profiles, every one of which encodes numerous frequent item sets in a natural way. The items in a c-profile should be independent with respect to the pattern base. Though, a pattern from a c-profile frequently contributes slightly similarity, thus a c-profile is not remain representative of its patterns to any further extent.

3. System Implementation

System of representative pattern set mining is made with unitizing different modules. In first module frequent patterns get extracted with considering levels of threshold and support. Then in second module data structure called CFP-tree is generated. This structure stores the frequent patterns that are extracted from dataset. CFP-tree is a very compact structure for storing these frequent patterns. By some rules and conventions these patterns are placed in the tree. After this MinRPset algorithmis applies on tree. MinRPset calls Flex_Search_CXs algorithm which call Search_CX algorithm which gives C(X) back to Flex_Search_CXs. This same goes repeating for every root node of tree and finally MinRPset get set of C(X). FlexRPset provides one extra parameter K to allow users to make a tradeoff between result size and efficiency. User may increment K until he becomes satisfied. Latter on if there remains any non closed entries in C(X)s that get removed. Finally by applying greedy set cover algorithm, required patterns will get i.e. small number of representative patterns that approximate all other patterns.

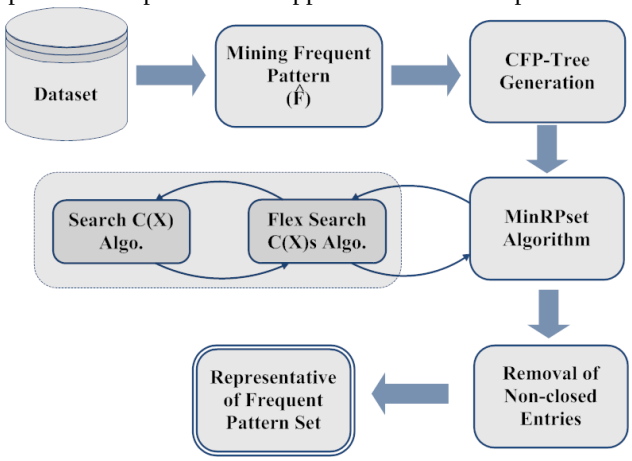

Figure 1: System Architecture

3.1 Mining Frequent patterns

From dataset there can have lots of patterns. Every pattern is associated with support. Support of pattern is calculated by considering the items in patterns and their individual occurrence. User decides a threshold of support called min_sup. Then support of every pattern will map with threshold. Pattern having support greater than threshold are considered as frequent pattern. These patterns are of interest for further processing. Frequent patterns hold a property called anti-monotone property. In accordance with this property, if a pattern is a frequent pattern, then all of its subsets are also frequent, that means, support of any pattern is always same or greater than its superset.

Here concept of ϵ-cover is used. ϵ is a real number $\epsilon \in [0,1)$ and two patterns $X_1$and X_2 are there, then it can say that pattern X_1 is ϵ -covered by X_2 if $X_1 \subseteq X_2$ and $D(X_1,X_2) \leq \epsilon$. Where $D(X_1,X_2)$ is the distance between patterns X_1 and X_2. Condition $X_1 \subseteq X_2$ ensures that the two patterns X_1 and X_2 have similar items, and condition $D(X_1,X_2) \leq \epsilon$ ensures that these two patterns have similar supporting transaction sets and similar support. Then for two patterns X_1 and X_2, if pattern X_1 is ϵ -covered by pattern X_2 and using $supp(X_2)$ to approximate $supp(X_1)$, then the relative error is no larger than ϵ. That means if a frequent pattern X_1 is ϵ -covered by pattern X_2, then $supp(X_2) \geq min\_sup \cdot (1-\epsilon)$. And that relative error is [1]

$$\frac{supp(X_1)-supp(X_2)}{supp(X_1)} \quad (1)$$

Consider set of frequent patterns F in a dataset D with respect to threshold min_sup, and $\hat{F}$ be the set of patterns with support no less than $min\_sup \cdot (1-\epsilon)$ in D, then obviously, $F \subseteq \hat{F}$. Then for a pattern $X \in F$, C(X) denotes the set of frequent patterns that can be ϵ-covered by X. Thus, $C(X) \subseteq F$, and if X is frequent, $X \in C(X)$.And this module gives $\hat{F}$, as discribed above.

3.2 CFP-Tree Generation

The structure CFP-tree is particularly designed and built for storing and querying frequent patterns. It is similar to a setenumeration tree [7]. The assembly of CFP-tree is based on a pattern-growth approach. The root node of tree holds all frequent items that are sorted in way of ascending frequency. Fig.2 shows the frequent patterns by considering support greater than 3. From these patterns CFP-tree in fig.3 is generated.

ID	Itemsets	ID	itemsets	ID	itemsets
1	a:6	9	ac:3	17	acm:3
2	b:3	10	af:4	18	afm:3
3	c:3	11	am:5	19	afp:3
4	d:3	12	ap:3	20	amp:3
5	e:3	13	cm:3	21	fmp:3
6	f:5	14	fm:3	22	afmp:3
7	m:5	15	fp:4		
8	p:4	16	mp:3		

Figure 2: Frequent Patterns(min_sup=3)

Every item i in the root node may have a subtree, and this subtree saves all frequent patterns that are determined from i's conditional database. All node of a CFP-tree is an array of variable-length. If a node has more than one entry, then each entry holds one item precisely. And if a node contains just one entry, then it is called as 'singleton node'. Singleton nodes may hold multiple items. An entry E stores numerous quantity of information: (1) number of items: m, ($m \geq 1$), (2)

support of E, (3) pointer that points to the child node of E and (4) id of entry that is allotted by preordering. The structure CFP-tree shares the storage of different patterns by means of their prefixes and suffixes. Prefix sharing is simple to understand. E.g. patterns {f, m} and{f, a} share the similar prefix {f} in Fig.3. In CFP-tree node with multi-entry the items that comes next to E can only come in subtree of E, and these items are called 'candidate extensions' of E. The total entries in a CFP-tree are greatly lesser than that of total amount of patterns stored in the tree. For an entry E, simply its longest pattern is closed one. Other patterns of E those are shorter than the longest pattern are not closed. If the longest pattern of an entry is not closed, then that entry is a non-closed entry.

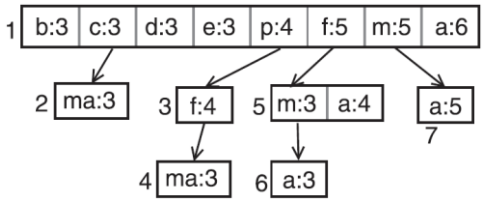

Figure 3: CFP-Tree Structure

Each entry in a CFP-tree corresponds to one or more patterns with the same support, and these patterns include the items on the path from the root to the entry. Items included in singleton nodes are elective. Let E be an entry, X_m be the set of items in the multiple-entry nodes and X_S be the set of items in the singleton nodes on the path from the root to the parent of E respectively. The set of patterns represented by E is $\{X_m \cup Y \cup Z \mid Y \subseteq Xs, Z \subseteq E.items, Z \text{ is not } \phi\}$. The longest pattern represented by E is $X_m \cup Xs \cup E.items$. Node 4 in the given above fig.3 contains only one entry. For this entry $X_m = \{p\}$, $X_s = \{f\}$ and E.items={m, a}. Therefore node No.4 represents six patterns :{p, m}, {p, a}, {p, m, a}, {p, f, m}, {p, f, a} and {p, f, m, a}. E.pattern is used to denote the longest pattern represented by E.

The structure CFP-tree holds two important properties.[8]

1) The a priori property: The support of entry E cannot be greater than that of its ancestors. This property can be used when processing queries with minimum support constraints. If the support of an entry does not satisfy the minimum support constraint specified in a query, then there is no need to access the subtree pointed by the entry.
2) The left containment property: In a CFP-tree node, the item of entry E can appear in the sub trees pointed by the entries before E, but cannot appear in the subtrees pointed by the entries after E. For example, in the root node of the CFP-tree shown in Fig. 3, item f can appear in the subtree pointed by item c, d or p, but it cannot appear in the subtree pointed by item m or a. The left containment property can be utilized when processing superset queries.

3.3 MinRPset Algorithm

Finding a minimum representative pattern set is equal to finding a minimum number of sets in S that can cover all the frequent patterns in F. This is a set cover problem, and it is NP-hard. There have the well-known greedy algorithm [5] to solve the problem, which achieves an approximation ratio of $\sum_{i=1}^{k} \frac{1}{i}$where k is the maximalsize of the sets in S. And that is algorithm MinRPset. This algorithm takes root ofCFP-Tree. It gives output with working two more algorithms 1)Flex_Search_CXs and2)Search_CX.

MinRPset Algorithm

1. Mine patterns with support ≧min_sup·(1- ϵ) and store them in a CFP-tree;

Let root be the root node of the tree;

2. Flex_Search_CXs(root);
3. Remove non-closed entries from C(X)s;
4. Apply the greedy set cover algorithm on C(X)s to find representative patterns and output them;

3.4 Flex_Search_CXs

The MinRPset algorithm sometimes can become awfully time-consuming especially when the quantity of frequent patterns is huge on a dataset, as it needs to look for subsets over a big CFP-tree for a large quantity of patterns. Moreover, the main memory does not become enough to fit large amount of set of C(X)s in it. To solve this problem, instead of searching C(X)s for every closed patterns, there can selectively generate C(X)s such that each frequent pattern is covered a sufficient number of times, in the hope that the greedy set cover algorithm can still find a near optimal solution. Naturally, the fewer the number of C(X)s generated, the more efficient the algorithm is.This is the basic idea of the FlexRPset algorithm..

To control the minimum number of times that a frequent pattern needs to be covered the FlexRPset algorithm uses a parameter K. Algorithm uses the depth first order to traverse a CFP-tree from left to right. It traverses the subtree of an entry E first (line 3-4) before it processes E (line 5-8), which means that when E is processed, all the supersets of E have been processed already, and E cannot be covered any more except by itself. If E is frequent and it is covered less than K times, then there can generate C(E.pattern) to cover E. If E has already been covered at least K times when Eis visited, then focus is at the ancestor entries of E. For an ancestor entry E of E, most of its supersets are already processed too when E is visited, hence not many remaining entries can cover E. If E is frequent, E can be ϵ-covered by E and E is covered less thanK times, then there can also generate C(E.pattern) to cover E. User may start with value of K=1, as value of K is small,lot of representative patterns get generated, increasing value k=10 minimizes size of pattern and so on. As k gets increases, running time becomes longer. User decides to stops increasing value of k when he becomes happy with pattern.

Flex_Search_CXs Algorithm

- **Input:**cnode is a CFP-tree node; //cnode is the root node initially.
  K is the minimum number of times that a frequent closed pattern needs tobe covered;
- **Output:**C(X)s;
- **Description:**

1. **for** each entry E ∈cnode from left to right **do**
2. **if** E is not marked as non-closed **then**
3. **if**E.child≠ NULL **then**
4. Flex_Search_CXs(E.child);
5. **if** E is more frequent than its child entries **then**
6. **if** (E is frequent AND E is covered less than K)**OR**(∃ an ancestor entry E′ of E such that E′is frequent E′ can be ϵ-coveredby E and E′ is covered less than K times) **then**
7. X=E.pattern;
8. C(X) = Search_CX(root, X, E.support);

3.5 Search_CX

Algorithm Search_CX shows the pseudo-codes for retrieving C(X). Primarily, root node of the CFP-tree is cnode. Parameter Y contains the set of items to be searched in cnode. It is set to X initially. Once an entry E is visited, the item of E is removed from Y when Y is passed to the subtree of E (line 8, 18). The item of E is also excluded when Y is passed to the entries after E (line 21). This is because the

Search_CX Algorithm

- **Input:** cnode is a CFP-tree node; //cnode is the root node initially.
  Y is the set of items to be searched in cnode; //Y=X initially. supp(X) isthe support of X;
- **Output:**C(X);
- **Description:**

1. **if**cnode contains only one entry E **then**
2. **if**E.support==supp(X) AND E.pattern⊂ X ANDE is on the right of X **then**
3. Mark E as non-closed;
4. **if** E is not marked as non-closed AND E is frequent **then**
5. **if**E.items⋂ Y≠ ∅ AND E.support ≤ $\frac{supp(X)}{(1-\epsilon)}$**then**
6. **Put**E.preorder into C(X);
7. **if**E.child≠ NULL AND Y-E.items≠ ∅**then**
8. Search_CX(E.child, Y-E.items,supp(X));
9. **elseif**cnode contains multiple entries **then**
10. **for**each entry E∈cnode from left to right **do**
11. **if**E.items∈Y AND E is frequent **then**
12. **if**E.support == supp(X) AND E.pattern⊂ X AND E ison the right of X **then**
13. Mark E as non-closed;
14. **if** E is not marked as non-closed **then**
15. **if**E.support ≤ $\frac{supp(X)}{(1-\epsilon)}$ AND E is more frequent thanits child entries **then**
16. **Put**E.preorder into C(X);
17. **if**E.child≠ NULL AND Y-E.items≠ ∅**then**
18. Search_CX(E.child, Y-E.items, supp(X));
19. **if**supp(E.pattern⋃Y) >$\frac{supp(X)}{(1-\epsilon)}$**then**
20. **return**;
21. Y=Y-E.items;

item of E cannot appear in the subtrees pointed by entries after E. During the search of C(X)s, there can also mark non-closed patterns. From the patterns of E, if longest among them is a proper subset of X, E.support=supp(X)and E occurs on the right of X, then E is marked as non-closed (line2-3, 12-13), and it is skipped in subsequent search. For example, when user search for the subsets of pattern {p, f, m, a} using Algorithm Search_CX in Fig. 3, he finds that one subset {f, m} has the same support as {p, f, m, a} and {f, m} occurs on the right of {p, f, m, a}. Hence the entry of {f, m}, which is the first entry in node 5, is marked as non-closed. All the patterns in the subtree pointed by this entry cannot be closed either because for every pattern Z in the subtree, Z′ = Z ∪{p, a} is a proper superset of Z and it has the same support as Z. Entry {f, m} and its subtree are skipped in subsequent traversal.

3.6 Removal of Non-closed Entries

After having set of C(X), further task is to remove any non-closed entries. For desire work only closed patterns are in interest. A pattern is closed if it is more frequent than all of its supersets. So, by considering only closed one, non-closed entries get remove. And then greedy set cover algorithm is applied, that gives the less number of patterns.

4. Dataset and Results

Dataset foodmart can be obtain from (http://pentaho.dlpage.phiintegration.com/mondrian/mysqlfoodmartdatabase). Other dataset can also obtain from the FIMI repository (http://fimi.ua.ac.be/data/). Compression ratio is found 9 to 13 :1

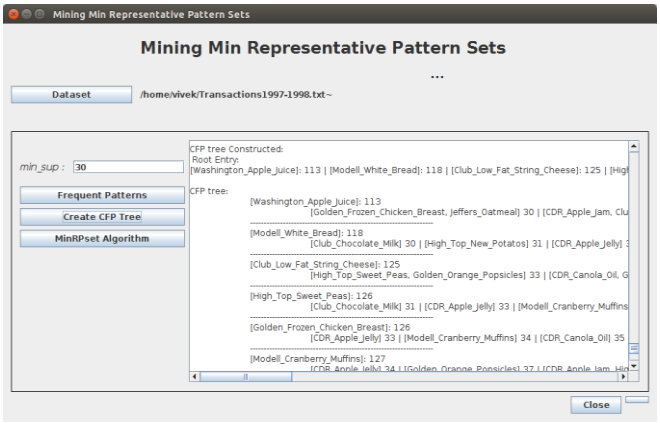

Figure 4: CFP-Tree constructed

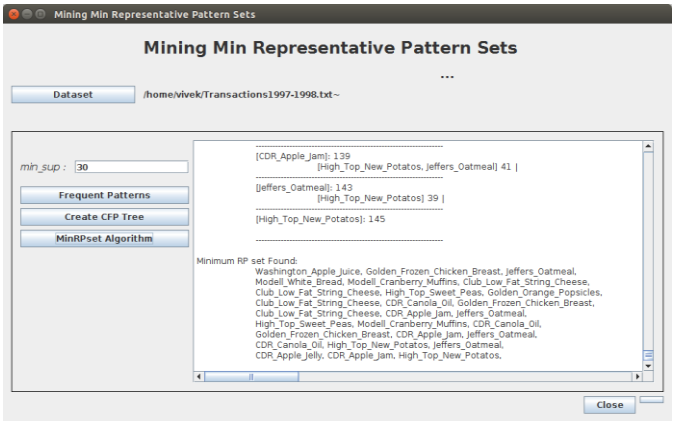

Figure 5: Representative Pattern Set

We tried different min_sup for this system, at value of min_sup for 10, 20, 30 and 35 we got 26, 10, 9 and 6 MinRP respectively by keeping value of k=20.

5. Conclusions

For finding minimum representative pattern sets, here have described algorithms MinRPset, Flex_Search_CXs and Search_CX. First frequent patterns are mined, and then used CFP-tree .By applying these algorithms on the CFP-tree, representative patterns generated. This also will provide some more benefits rather than minimum representative pattern sets. Users may not know what value should be used for ϵ at the beginning. The post processing strategy allows users to try different ϵ values. This is especially beneficial on very large datasets. It is easy to keep record of the set of patterns covered by each representative pattern. This information is useful for users to inspect individual representative patterns in more details.

6. Acknowledgement

I express great many thanks to Prof. D.M. Padulkar for his great effort of supervising and leading me, to accomplish this fine work. Without his Coordination, guidance and reviewing, this task could not be completed alone.

References

[1] G. Liu, H. Zhang, and L. Wong, "A Flexible Approach to Finding Representative Pattern Sets," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 7, july 2014.
[2] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," *in Proc. SIGMOD, Washington, DC, USA*, 1993, pp. 207216.
[3] D. Xin, J. Han, X. Yan, and H. Cheng, "Mining compressed frequent-pattern sets," *in Proc. 31st Int. Conf. VLDB, Trondheim, Norway*, 2005, pp. 709720.
[4] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," *in Proc. FIMI*, 2003.
[5] X. Yan, H. Cheng, J. Han, and D. Xin, "Summarizing itemset patterns: A profile based approach," *in Proc. KDD, Chicago, IL, USA*, 2005, pp. 314323.
[6] A. K. Poernomo and V. Gopalkrishnan, "CP-summary: A concise representation forbrowsing frequent itemsets," *in Proc. KDD, New York, NY, USA*, 2009, pp. 687696.
[7] R. Rymon, "Search through systematic set enumeration," *in Proc. KR*, 1992, pp.539550.
[8] G. Liu, H. Lu, and J. X. Yu, "CFP-tree: A compact disk-based structure for storingand querying frequent itemsets," *Inf. Syst.*, vol. 32, no. 2, pp. 295319, 2007.

Author Profile

Mr. Vivek Satpute received the Bachelors degree(B. E.) computer Engineering from savitribai phule pune university he is now pursuing M. E. degree.

Prof. Digambar Padulkar is working as Assistant professor, Department Of Computer Engineering, Vidya Pratishthan's College Of Engineering, Baramati, Pune, Maharashtra.. His research areas include Uncertain Data Mining, Applications of Data Mining in Business and Intelligent predictions.