CHAPTER 4

RAPID CLOSED FREQUENT ITEMSET ALGORITHM

4.1 INTRODUCTION

In the past, most previous works focus on efficient finding all frequent itemsets. As an early intensive work, Srikant et al proposed five algorithms that apply the horizontal database format and breath-first search strategy like Apriori algorithms. These algorithms spend a lot of time in multiple scanning a database. As an algorithm, Prutax was proposed and applying a vertical database format to reduce the time needed in database scanning. Nevertheless, a limitation of this work is the cost of checking whether their ancestor itemsets are frequent or not using a hash tree. There exists a slightly different task for dealing with multiple minimum supports. This thesis is efficiently mined all frequent itemsets, furthermore to improve time complexity of the mining process. The main idea of these approaches focus to find only a small set of closed frequent itemsets, which is the representative of a large set of frequent itemsets. This technique helps us reduce the computational time. Thus, this work intends to apply this traditional concept to deal with the itemsets in association rule mining. In this work, this work proposes a novel efficient algorithm, named RCFIA and testified in Chess, connect, pumbs and mushroom datasets to mine closed frequent itemsets.
4.2 PROBLEM DEFINITIONS

Data mining frequently generates a vast number of patterns to fulfil the min_sup threshold, particularly while min_sup is very low. It is most important challenge in data mining frequent patterns from a large data set. But large frequent patterns may hold an number of smaller, frequent sub-patterns. It was a main problem of frequent pattern mining task. To defeat this problem, some efficient methods were used. They were closed and maximal frequent pattern mining.

A pattern may be a closed frequent pattern or maximal frequent pattern. Whatever it is, a particular pattern a is frequent in a Data set D and if there was no correct super-pattern b, then the pattern b has the same support count as the pattern a and also frequent in the Data set D.

\[ a \subseteq b \]

However, the closed frequent pattern contains the full information about its equivalent frequent pattern for its min-sup threshold. But in the maximal frequent pattern does not contain the full information about its equivalent frequent pattern.

An association rule can be formally stated as follows:

Let \( I = \{a, b, c, d, e, f, g, h\} \) be a set of distinct items,

\[ T = \{T_1, T_2, T_3, T_4, T_5, T_6\} \] be a set of transaction identifiers (tid’s).
The database can be viewed in two formats, i.e. horizontal format and vertical format. An edge in a taxonomy represents is-a relationship. \( A \) is called an ancestor item of \( B, C, A \) and \( B \). \( A \) is called a descendant item of \( U \) and \( V \). Leaf items of a taxonomy are presented in the original database. Intuitively, the database can be extended to contain the ancestor items by adding the record of ancestor items of which tidsets are given by the union of their children.

A set \( I_G \subseteq I \) is called a itemset (I) when \( I_G \) is a set of generalized items where no any items in the set is an ancestor item of the others.

The support of \( I_G \), denoted by \( \sigma (IG) \), is defined as a percentage of the number of transactions in which \( I_G \) occurs as a subset to the total number of transactions. Only the “T” that has its support greater than or equal to a user-specified minimum support (\( \text{minsup} \)) is called a frequent itemset (FI).

A rule is an implication of the form

\[ R: I_1 \rightarrow I_2, \text{ where } I_1, I_2 \subseteq I. \]

\( I_1 \cup I_2 = \emptyset, I_1 \subseteq I_2 \) is generalized FI, and no item in \( I_2 \) is an ancestor of any items in \( I_1 \). The confidence of a rule, defined as \( \sigma (I_1 \subseteq I_2) / \sigma (I_1) \), is the conditional probability that a transaction contains \( I_2 \), given that it contains \( I_1 \). The rule is called a Association Rule (AR) if its confidence is greater than or equal to a user-specified minimum confidence (\( \text{minconf} \)).
The task of ARM can be divided into two steps,

i) Finding all FIs and generating the ARs.

ii) The second step is straightforward while the first step takes intensive computational time. This work try to improve the first step by exploiting the concept of closed itemsets to ARM, and find only a small set of closed itemsets to reduce the computational time.

4.3 RELATED WORKS

Pasquit et al was proposed his algorithm called frequent closed itemset of mining when A-close (Apriori-based algorithm) algorithm was presented. Many closed pattern mining algorithm were already exist, they are CLOSET+ (Wang et al.,2003a), CHARM (Zaki and Hsiao et al 2002), CLOSET (Pei et al.2000), AFOPT (Liu et al. 2003) and FPClose (Grahne and Zhu et al 2003). Checking whether a pattern is closed or maximal is a main challenge in closed frequent pattern mining.

To approach this issue there are two strategies available:

- The pattern by hashing TID values are used to keep track of the TID list of a pattern and index. This method maintains a compact TID list called a diffset by CHARM.

- Maintaining the pattern in a pattern-tree is similar to FP-tree.
CLOSET+, AFOPT and FPClose are used to break this method. Goethals and Zaki (2003) was committed to the workshop called Frequent Itemset Mining Implementation (FIMI) to implement the methods of frequent itemset mining. Because of inherits the similar analytical from mining closed itemsets it provides an attractive and significant alternative to mining frequent itemsets, but generates a much smaller set of results. Hence closed itemset mining achieves an enhanced scalability and interpretability.

4.4 FREQUENT CLOSED ITEMSETS ALGORITHMS

In Data mining, lots of new algorithms are coming up to attain the frequent closed itemsets from frequent itemsets. When finding maximal frequent itemsets, these algorithms finds the frequent closed itemsets.

Also, without any extra computation time, the extension of Apriori-Close algorithm discover frequent and frequent closed itemsets. It is similar to the Apriori algorithm.

According to the theorem, if the items are stored in lexicographic order then consider k is the size of the largest frequent itemsets. Finding frequent itemset and count their support, numerous and efficient algorithms have been proposed. These types of algorithms first discover the maximal frequent itemsets and later attain the all frequent itemsets.
All type of these algorithm provide the result as: $\text{LO} = \bigcup_{i=1}^{i=k} \text{UP}\bigcup_{i=k}^{i=r}$

Here $\text{LO}_i$ means it hold all frequent $i$-itemsets. According to the theorem and Proposition 1, from the frequent itemsets and their support counts without any dataset contact the frequent itemsets and their support counts are calculated.

The following re-presented DFCA algorithm is used to find the frequent closed itemsets form the frequent itemsets. $\text{M}_i$ is the sets and it is the input of the algorithm. And $\text{M}_j \quad 1 \leq J \leq r$, means it holds all frequent itemsets in the dataset. It reputedly find the sets $\text{FRC}_j, \quad 0 \leq J \leq r$, of all frequent closed $i$-itemsets from $\text{FRC}_i$ to $\text{FRC}_0$

\begin{verbatim}
1. $\text{FRC}_i \leftarrow \text{M}_i$
2. for ($j \leftarrow r-n; j \neq 0; j\downarrow$) do begin
3.  $\text{FRC}_j \leftarrow \{}$; // $\text{FRC}_j$ means frequent closed $i$-itemsets and support
4.  forall itemsets $n \in \text{M}_j$ do begin
5.  isclosed $\leftarrow$ true;
6.  forall itemsets $n' \in \text{M}_{j+1}$ do begin // $\text{M}_i$ means frequent $i$-itemsets and support
7.  if ($n \subseteq n'$) and (n.support = n'.support)
8.  then isclosed $\leftarrow$ false;
9.  end
10. if (isclosed = true) then $\text{FRC}_j \leftarrow \text{FRC}_j \cup \{n\};$
11. end
12. end
13. $\text{FRC}_0 \leftarrow \{}$;
14. forall itemsets $n \in \text{M}_1$ do begin
15. if (n.support = $||\text{O}||$) then $\text{FRC}_0 \leftarrow \{}$;
16. end
\end{verbatim}

\textit{Figure 4.1:} DFCA (Derived Frequent Closed Algorithm)
First assign the largest frequent itemsets \( M_r \) to the set \( FRC_r \). Then it repeatedly finds the i-itemsets in \( M_j \) from \( M_{k-n} \) to \( M_n \) which are closed. Then it check that \( n \) has the same support count as a frequent \((j+1)\)itemset \( n' \) in \( M_{j+n} \) from \( M_{r+n} \) to \( M_n \). If so, the itemset \( n' \) subset of \( p(n) \) and then \( n \neq p(n) \): \( n \) and it is not closed. If not, \( n \) is a frequent closed itemset and is inserted in \( FRC_i \). In the final phase, the algorithm finds if the null itemset is closed by the starting initializing \( FRC_0 \) with the null iteset and then taking into account all frequent 1-itemsets in \( M_1 \). In this 1 common to all objects if 1-itemset 1 has a support count equal to the number of objects in the context then the itemset maynot be closed \((\text{supp}((\emptyset)) = \|O\| = \text{supp}(n))\) and is removed from \( FRC_0 \). Hence, each set \( FRC_i \) must contains all frequent closed i-itemsets, at the end of this algorithm. The computation of the set \( FRC_r \) containing the largest frequent closed itemsets, because of the all maximal frequent itemsets are maximal frequent closed itemsets, and it is correct.

The accuracy of the calculation of sets \( FRC_i \) \( J < r \) relies on Proposition 1. This proposition enables to determine if a frequent i-itemset \( l \) is closed by comparing its support and the supports of the frequent \((j+1)\)-itemsets in which \( l \) is included. If one of them has the same support as \( l \), then \( l \) cannot be closed.
4.4.1 Apriori-Close Algorithm

In this area present an extension of the Apriori algorithm computing simultaneously frequent and frequent closed itemsets. The re-presented DF_Apriori-Close pseudo-code is given in Figure. 4.2. The algorithm iteratively generates the sets $M_i$ of frequent $i$-itemsets from $M_1$ to $M_k$. Besides, during the $i$th iteration, all frequent closed $(i-1)$-itemsets in $FRC_{i-1}$ are determined. The set $FRC_k$ is determined during the last step of the algorithm.

First, the variable $k$ is initialized to 0. Then, the set $L_1$ of frequent 1-itemsets is initialized with the list of items in the context and one pass is performed to compute their support. The set $FRC_0$ is initialized with the empty itemset and the supports of $L_i$ Set of frequent $i$-itemsets, their support and marker closed itemset is indicating if closed or not. $FRC_i$ Set of frequent closed $i$-itemsets and their support.

```
1. k ← 0;
2. itemsets in $L_1$ ← \{1-itemsets\};
3. $L_1$ ← Support-Count($L_1$
4. $FRC_0$ ← \{\emptyset\};
5. \textbf{forall} itemsets $l$ ∈ $M_1$ \textbf{do begin}
6. \textbf{if} ($l$.support < minsupp) \textbf{then} $M_1$ ← $M_1$ − \{l\};
7. \textbf{else if} ($l$.support = ||O||) \textbf{then} $FRC_0$ ← \{\};
```
8. end

9. for (i <- 1; Mi !={} ; i++) do begin

10. forall itemsets l !∈ Mi do l'.isclosed ← true; // Isclosed- indicating if closed or not

11. Mi+1 ← Apriori-Gen(Li);

12. forall itemsets l ∈ Mi+1 do begin

13. forall i-subsets l’ of l do begin

14. if (l’ !∈ Mi) then Mi+1 Mi+1 ∪ {l};

15. end

16. end

17. Mi+1 Support-Count(Mi+1);

18. forall itemsets l ∈ Mi+1 do begin

19. if (l.support < minsupp) then Mi+1 Mi+1 ∪ {l};

20. else do begin

21. forall i-subsets l’ ∈ Mi of l do begin

22. if (l.support = l’.support) then l’.isclosed ← false;

23. end

24. end

25. end

26. FRCi ← {l ∈ Mi | l.isclosed = true}; // FRCi- Set of frequent closed i-itemsets and their support

27. k ← i;

28. end

29. FRCk ← Mk;

Figure 4.2: DF_Apriori-Close (Discovering Frequent and Frequent Closed with Apriori-Close)
Consider the itemsets in $M_1$ (from steps 5 to 8). All infrequent 1-itemsets are removed from $M_1$ and if a frequent 1-itemset has a support count which is equal to the number of objects in the context then the empty itemset is removed from $FRC_0$ (step 7). During each of the following iterations, frequent itemsets of size $i+1$, $k > J \geq 1$, and then compute the frequent closed itemsets of size $i$ as follows: For all frequent i-itemsets in $M_i$, the marker basket itemset closed assigned to true. When applying the Apriori-Gen function to the set $M_i$, a set $M_{i+1}$ of possible frequent $(i+1)$-itemsets are created. And then it check that all its subsets of size $i$ exist in $M_i$, for each of the possible frequent $(i+1)$-itemsets.

To compute the supports of the remaining itemsets in $M_{i+1}$ one pass is allotted. If $l$ is infrequent then it is discarded from $M_{i+1}$ for each $(i+1)$-itemsets $l \in M_{i+1}$. If not, verify that supports of $l0$ and $l$ are equal for all $i$-subsets $l0$ of $l$, if so, then $l0$ cannot be a closed itemset and set to false to its marker isclosed $i$ (steps 20 to 24). Then, marker isclosed which is true are inserted in the set $FRC_i$ of frequent closed i-itemsets for all frequent i-itemsets in $M_i$ and the variable $k$ is set to the value of $i$. Finally, the set $FRC_k$ is initialized with the frequent k-itemsets in $M_k$.

I. Apriori-Gen function: When this Apriori-Gen function [2] is applied to a set $M_i$ of frequent i-itemsets, it returns a set $M_{i+1}$ of potential frequent $(i+1)$-itemsets. While joining two itemsets in $M_i$, a new itemset in $M_{i+1}$ is created and they share common first i-1 items.
II. **Support-Count function:** Here, a set $M_i$ of $i$-itemsets are taken as the Support-Count function argument. It competently computes the support-counts of all itemsets $l \subseteq M_i$. It need only one dataset pass, and the supports of all itemsets $l \subseteq M_i$ that are included in the set of items associated with $o$, i.e. $l \subseteq f(\{o\})$ while each object $o$ is red, and they are incremented. Subset functions are used to found the subsets of $f(\{o\})$ quickly.

III. **Correctness**

Since the support of a frequent closed itemset $l$ is different from the support of all its supersets the computation of sets $FRC_i$ for $i < k$ is correct. Hence, a frequent $i$-itemset $l' \subseteq M_i$ is determined closed or not by comparing its support with the supports of all frequent $(i + 1)$-itemsets $l \subseteq M_{i+1}$ for which $l' \subseteq l$. The correctness of the computation of the set $FRC_k$ is containing the largest frequent closed itemsets.

4.4.2 **Relative Performance of Apriori and Apriori-Close**

Characteristics of the datasets used for Apriori and Apriori-close algorithms are discussed below. The datasets are chess, mushrooms and market basket.
These datasets are the market basket data and the Chess Dataset. In all experiments, one can attempted to choose significant minimum support and confidence threshold values observed and examined rules extracted in the bases.

<table>
<thead>
<tr>
<th>Name</th>
<th>No of Objects</th>
<th>Average size of objects</th>
<th>Number of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>100,000</td>
<td>10</td>
<td>1,000</td>
</tr>
<tr>
<td>Mushrooms</td>
<td>8,416</td>
<td>23</td>
<td>127</td>
</tr>
<tr>
<td>Market basket</td>
<td>8,416</td>
<td>20</td>
<td>386</td>
</tr>
</tbody>
</table>

**Table 4.1:** Datasets.

This experiment is used to compare response times obtained with Apriori and Apriori-Close on the three datasets as per the table 4.1. The chess and market basket datasets results are shown in the table 4.2 and observed from the table execution times are the same for the two algorithms.
<table>
<thead>
<tr>
<th>Minsupp</th>
<th>Apriori Secs</th>
<th>Apriori-Close Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0%</td>
<td>1.99</td>
<td>1.97</td>
</tr>
<tr>
<td>1.0%</td>
<td>3.47</td>
<td>3.46</td>
</tr>
<tr>
<td>0.5%</td>
<td>9.62</td>
<td>9.70</td>
</tr>
<tr>
<td>0.25%</td>
<td>15.02</td>
<td>14.92</td>
</tr>
</tbody>
</table>

**Table 4.2**: Execution Times of Apriori and Apriori-Close.

a. Chess dataset

<table>
<thead>
<tr>
<th>Minsupp</th>
<th>Apriori Secs</th>
<th>Apriori-Close Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>70%</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td>50%</td>
<td>2.40</td>
<td>2.70</td>
</tr>
<tr>
<td>30%</td>
<td>18.22</td>
<td>17.93</td>
</tr>
</tbody>
</table>

b. Market basket data
4.5 RAPID CLOSED FREQUENT ITEMSET ALGORITHM

Procedure RCFIA_main (Dataset D, Minsupp \( \alpha \))

Begin

Step 1:  root = null tree

Step 2:  NewChild (root, FIs from second level of taxonomy)

Step 3:  RCFIA_extend(Root)  RCFIA_extend(Father)

Step 4:  for each \( F_i \) in Father.Child

Step 5:  \( C = \text{TaxChild}(F_i) \)

Step 6:  if supp\((C) \geq \text{minsup} \) then

Step 7:  RCFIA_property(nodes, C)

Step 8:  for \( j = i+1 \) to |Father.Child|

Step 9:  \( C = F_i U F_j \)

Step 10:  if supp\((C) > \text{minsup} \) then

Step 11:  RCFIA_property(Nodes,C)

Step 12:  if \( F_i \text{Child} \neq \text{NULL} \) then RCFIA_extend\((F_i) \)

Step 13:  RCFIA_property(Node,C)

Step 14:  if \( t(F_i) = t(F_j) \) and Child\((F_i)\) = \( \) then

Step 15:  else if \( t(F_i) \neq t(F_j) \) and Child\((F_i)\) = \( \) then

101
**Step 16:** replace($F_i$) with $C$

**Step 17:** else if $t(F_i) . t(F_j)$ then

**Step 18:** remove($F_j$);

**Step 19:** if Hash($t(C)$) then NewChild($F_i, C$)

**Step 20:** else if Hash($t(C)$) then NewChild($F_i, C$)

End procedure

**Figure 4.3 RCFIA**

---

**4.6 EXPERIMENTAL RESULTS**

**4.6.1 Implementation Study**

The search tree’s size is moderately sensitive to the worth of the search window. Also, **RCFIA** re-searches area extra overhead to the transposition table. And checking is important for the transposition table which works properly or not. **RCFIA** does more re-search. The search tree’s $c$ size is differ from one position to another position. Usage of very a large test set in our experiments various attributes like without extensions, without null-move pruning, different transposition table sizes, disregard counting of quiescence nodes and storage schemes, disable some parts of the move ordering. To understand the searching technique in much better way is to understand the search better, the idea is to disable as much smarts as possible, and study the behavior of a clean, noise-less in all debugging.
In a re-search, each pass of RCFIA would re-explore lot of its nodes. To make the RCFIA to be efficient to store the nodes as it has searched. A normal transposition table of reasonable size suffices, used as our experiments in RCFIA is a minimax search algorithm, simpler and more efficient algorithms.

The original items contain in the leaf-level of taxonomy. In real datasets, the number of CFIs is much smaller than that of FIs. With the same datasets, the ratio of the number of FIs to that of CFIs typically increases when this worked on the minsup. The more time reduction is gained. The ratio can grow up to around more times, which results in reduction of running time takes minimal times. Note that in datasets, the number of FIs is slightly different from the number of CFIs. This indicates that the real datasets are dense but the syntactic datasets are sparse. This result makes us possible to reduce more computational time by using RCFIA in real situations.

Using the data generator, this work generates chess datasets as described below.

- Independent datasets.
- Correlated datasets.
- Anti-correlated datasets.

Moreover, in the index, each axis in chess is divided into a number of equal intervals which is fixed as t.
This work first compares the query performance of the algorithm RCFIA and path tree, pruning. This work focuses on evaluating the efficiency of the pruning technique. This work implements the first set of experiments in the following experimental setting:

1. The cardinality of datasets is set to $1 \times 10^n$ and $8 \times 10^n$;

2. $k = 8$ square and $v$ varies from 1 to 64 square.

For independent datasets and correlated and anti-correlated datasets, respectively,

Correlated datasets

(a) Cardinality = $1 \times 10^n$.

(b) Cardinality = $8 \times 10^n$.

Anti-correlated datasets.

(a) Cardinality = $1 \times 10^n$.

(b) Cardinality = $8 \times 10^n$.

The results of experiments are independent. Observe this research work that each algorithm needs to spend more query time in this case. It is mainly because there is more space set in anti-correlated datasets.
The following figure illustrate the maximal probability of the pieces movements on chess boards squares.

![Diagram of chess pieces movements]

**Figure 4.4 Levels of Chess pieces movement**

Any movement of “n” pieces on the board can be calculated using the formulae \( n \times v \), here \( v = 1 \) to \( 8 \) and \( n \) is number of pieces.

In the series of experiments, this work studies the impact of the pruning technique on the query performance of RCFIA. Square vary from one
movement to other movements (occurrences). It can be observe that the pruning technique can optimize the RCFIA algorithm in most cases and the pruning effect becomes more evident as the cardinality of datasets increases.

4.6.2 Comparative Study

The following figure 4.5 to 4.8 shows the comparative study of proposed RCFIA with Apriori, A-close and CHARM algorithms. This works is applied in various datasets chess, connect, pumsb and mushroom in order to prove the efficiency of the research work:. Performance of proposed algorithm is better than other three existing algorithms from low to high support levels. From this study, the support is low; the execution time is high, whereas the support is high, the execution time become low.
Figure 4.5 Comparison of Execution Time between Apriori, A-Close, CHARMA and RCFIA in various Support threshold using Chess Dataset
Figure 4.6 Comparison of Execution Time between Apriori, A-Close, CHARM and RCFIA in various Support threshold using Connect Dataset
Figure 4.7 Comparison of Execution Time between Apriori, A-Close, CHARM and RCFIA in various Support threshold using pumsb Dataset
Figure 4.8 Comparison of Execution Time between Apriori, A-Close, CHARMA and RCFIA in various Support threshold using Mushroom Dataset.
The following table illustrated the execution time of RCFIA through the candidate generation for searching closed frequent itemset.

<table>
<thead>
<tr>
<th>% Minsupp</th>
<th>Number of Items</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Freq</td>
<td>#closed</td>
</tr>
<tr>
<td>50</td>
<td>16859</td>
<td>05</td>
</tr>
<tr>
<td>46</td>
<td>344414</td>
<td>46</td>
</tr>
<tr>
<td>60</td>
<td>1565631</td>
<td>299</td>
</tr>
<tr>
<td>80</td>
<td>6843135</td>
<td>1181</td>
</tr>
</tbody>
</table>

Table 4.3 Execution time for RCFIA

This work can mine a set of closed frequent itemsets and then result in reducing computational time. This work proposed an algorithm, named RCFIA, by applying some constraints and conditional properties to efficiently enumerate only closed frequent itemsets. The advantage of RCFIA becomes more central when minimum support is low and the dataset is dense and complicated. This approach makes it possible to mine the data in real situations.
4.7 SUMMARY

The introduction about RCFIA algorithm is explained in this Chapter, The relative performance of Apriori related and Apriori-Close algorithm are using DFCIA and DF Apriori close Algorithm results is merged the proposed rapid search datamining algorithm RSDCA with apriori based closed frequent pattern mining algorithm. The proposed algorithm is explained and implemented through various datasets. The experimental study of the proposed work is shown in section 5.5. The discussed algorithm namely RCFIA is performed well than other existing algorithms further detailed compared with the RSDCA in this Chapter 6