Association Rule Based Closed Frequent Item Set Algorithm (CFIA) For Chess Dataset

E.Ramaraj[1], R.Gokulakrishnan[2], K.Rameshkumar[3]

ABSTRACT

Taxonomy over the items in database can be more initiative and informative in the area of knowledge discovery. Association Rule Mining (ARM) focuses on finding a set of all subsets of items called item sets. This paper proposes a new approach for closed item sets using association rules. A smaller set of closed item sets can be a substitute of a larger item set. A new algorithm to mine a set of closed frequent item set called CFIA is presented as a part of this paper. The CFIA algorithm outperforms the traditional approaches of mining frequent item sets especially when the item sets intensity is dense in the database or when there are complicated permutations, probability of occurrences like Chess datasets

Keywords : Association Rule Mining, Frequent Items, Machine Learning, Chess Dataset.

1. INTRODUCTION

Item sets frequently occur in database records or transactions. Extracting a subset of items influences the presence of another subset. The task of Association Rule Mining (ARM) is one important topic in the area of knowledge discovery in databases (KDD). However, the rules may not provide informative knowledge about the database with limitations due to granularity of the items.

[1]Director, Computer Centre, Alagappa University, Karaikudi. e mail: dr_ramaraj@yahoo.co.in
[2]Lecturer, Department of IT, J.J.College of Arts and Science, Pudukkottai. e_mail: rajgokul2002@yahoo.com
[3]FullTime Ph.D Scholar, Department of CSE, Alagappa University, Karaikudi, e_mail: rameshkumar_phd@yahoo.co.in

Association rule mining (ARM) is evolved using the information of pre-defined taxonomy over the items. The taxonomy may classify products or items by brands, groups, categories, and so forth. Given a taxonomy where only leaf items are present in the database, more initiative and informative details can be mined from the database using association rules. Each rule contains a set of items picked up from any level in the taxonomy. Most previous works focus in finding all frequent item sets in an efficient way. Srikant pioneered five algorithms that apply to item sets horizontally in a database. The algorithms spend a lot of time in multiple scanning of a database. A known limitation of this work is the cost of checking ancestor item sets frequently using a hash tree, gives us a direction to reducing search time during mining processes. The basic bottom line of these approaches would focus to find small sets of closed frequent item sets instead of a larger set reducing computational time. This paper proposes and presents an efficient algorithm; named CFIA testified on Chess datasets keeping in mind the traditional concept of dealing with the item sets using Association Rule Mining.

2. PROBLEM DEFINITIONS

An association rule can be formally stated as follows.

Let I = {A, B, C, D, E, U, V, W} be a set of distinct items,

T = {1, 2, 3, 4, 5, 6} be a set of transaction identifiers (t_{id}'s).

The database can be viewed into two formats, i.e. horizontal format and vertical format. An edge in

taxonomy connects a relationship with other taxonomy. V is called an ancestor item of U, 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 item set (I) when IG is a set of items where no any items in the set is an ancestor item of the others. The support of I_G, denoted by σ (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 "I" that has its support greater than or equal to a user-specified minimum support (minsup) is called a frequent item set (FI).

A rule is an implication of the form

$R : I_1 \rightarrow I_2$, where $I_1, I_2 \subseteq I$.

$I_1 \cap I_2 = \theta$, $I_1 \square I_2$ is GFI, and no item in I_2 is an ancestor of any items in I_1. The confidence of a rule, defined as $\sigma (I_1 \square I_2) / \sigma (I_1)$, 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 (minconf).

The task of ARM can be divided into two steps,

1) Finding all FIs and 2) generating the ARs.

2) The second step is straightforward. While the first step takes intensive computational time, it is improved by exploiting the concept of closed item sets to ARM, and finds only a small set of closed item sets to reduce the computational time.

3. Closed Item set (CI)

The concept of CI is defined by adapting the traditional concept of closed item sets in ARM, set of closed frequent item sets is sufficient to be the representative of a large set of FIs. Let the binary relation $\delta \subseteq I \times T$ is the extension database. For an arbitrary a $\square$ I and y $\square$ T, a δ b can be denoted when "a" is related to its "b" in database.

Let it $A \subseteq (I)$ and $(B) \subseteq (T)$:

Then the mapping functions,

$t: I \rightarrow T$, $t(A) = \{b \square T \mid \forall \ a \square X, a \delta b\}$

$t: T \rightarrow I$, $i(B) = \{a \square I \mid \forall \ b \square B, a \delta b\}$

Chess Piece connection between the power set of I (P (I)) and the power set of T (P (T)). The following properties hold for all $(A,A_1,A_2) \subseteq (I)$ and $(B,B_1,B_2) \subseteq (T)$:

1. $A_1 \subseteq X_2 \square t(A_1) \square t(B_2)$

2. $B_1 \subseteq B_2 \square i(B_1) \square i(B_2)$

3. $A \subseteq i(t(A) \text{ and } B \subseteq t(i(B))$

The support of a concept A × B is the size of CT (i.e. |B|). "I" is frequent when its support is greater than or equal to minsup. For any item set A, its support is equal to the support of its closure

$$(\sigma(A) = \sigma(c_{it}(A))).$$

Given A, its support $\sigma(A) = |t(A)| / |T|$, and the support of its closure

$$\sigma(c_{it}(A)) = |t(c_{it}(A))|/|T|.$$

To prove, it has to be

$t(A) = t(c_{it}(A))$.

Since c_{it} is the closure operator, it satisfies the first property that

$t(A) \subseteq c_{it}(t(A)) = t(i(t(A))) = t(c_{it}(A))$.

Thus $t(A) = t(c_{it}(A))$.

The $c_{it}(A)$ provides the "I" that is the maximal superset of A and has the same support as X.

Then $A \subseteq c_{it}(A)$, and $t(A) \subseteq t(c_{it}(A))$ due to the property concluding that

$t(A) = t(c_{it}(A))$.

Implicitly, the lemma states that all FIs can be uniquely determined by the CFIs since the support of any "I"s will be equal to its closure. In the worst case, the number of CFIs is equal to the number of FIs, but typically it is much smaller.

4 CFIA

All FIs can be enumerated by applying the constraints, subset-superset item sets and parent-child item sets. The proposed algorithm (CFIA) specifies the order of set enumeration by using the aforesaid constraints. The constraints specify a descendant and superset for qualifying item sets and CFIs are generated after the conditional properties are checked. The child item sets are generated by joining $M_1 \times t(M_1)$ with $M_2 \times t(M_2)$.

1. If $t(M_1) = t(M_2)$, then (1) replace X1 and children under M_1 with $M_1 \square M_2$, generate taxonomy- based child itemsets of $M_1 \square M_2$, and remove M_2.

2. If $t(M_1) \subset t(M2)$, then replace M_1 with $M_1 \cup M2$ and generate taxonomy-based child item sets of $M_1 \square M_2$.

3. If $t(M_1) \subset t(M_2)$, then generate join-based child item set of M_1 with $M_1 \square M_2$, add hash table with $M_1 \square M_2$ and remove M_2.

4. If $t(M_1) \neq t(M_2)$ and $t(M_1 \square M_2)$ is not contain in hash, then generate join-based child item set of M1 with $M_1 \square M_2$.

The CFIA algorithm starts with an empty set. Then, adds all frequent items in the second level of the taxonomy that are item V and W, and form the second level of the tree. Each item set has to generate two kinds of child item sets, i.e.taxonomy based and join-based item sets, respectively. The taxonomy-based item set is generated by joining last items in item sets and its child.

One taxonomy-based item set of V is VU. The first property holds for VU, which results in replacing V with VU and then generating VUA and VUB. The second taxonomy-based item set is joined with the current item set (VU), which produces VUC. Again, the first property holds for VUC, which results in replacing VU and the children in tree under VU with VUC. Next, the join-based child item set of V, VW, is generated. The third property holds for VW, which results in removing and then generating VW under V. In the same approach, the process recursively occurs until no new CFIs are generated. Finally, a complete item set tree is constructed without excessive checking cost.

4.1 Chess Set Description

Positional knowledge can be worth as little as one hundred set of a pawn. Big score swings can become inefficient in such programs while dynamically increasing the step size. Instead of using the previous bound, add extra points in the search direction. While searching upward add the bonus and subtract the bonus while searching downward at every second pass. When the bound overshoots the minimax value, make a small search in the opposite direction using the last search bound without an extra to achieve the final convergence point.

The size of the search tree is quite sensitive to the value of the search window. Also, for the transposition table, CFIA re-searches would cause a heavy overhead. Ensure that the transposition table works properly. CFIA does more of re-searches. The size of the search tree can differ significantly from position to position. A large test set in our experiments for different set-ups, such as, without null-move pruning, without extensions, disregarding counting of quiescence nodes, different transposition table sizes and storage schemes disable some parts of the move ordering. As in all debugging, to understand the search better, a clean, noise-less, algorithm is proposed.

During re-searches, each pass of CFIA would re-explore most of the nodes. CFIA can be more efficient if the searched nodes are stored. An ordinary transposition table of reasonable size used in our experiments is sufficient. CFIA is a minimax search algorithm, simpler and more efficient. When tested with a number of tournament playing programs of chess, it performed strongly while testing against Negascout, Cilkchess, Starscorates.

CHESS ALGORITHM-I

Function

```
CFIA (root : node_type; f : integer; d : integer)  : integer;
g := f;
upperbound : = + ∞;
lowerbound : = - ∞;
repeat
if   g == lowerbound
then P2 : = g + 1
else P2 : = g; g :=( P1, P2) Memory (root, P2- 1, P2, d);
if g < beta
then upperbound : = g
else lowerbound : = g;
until lowerbound > = upperbound;
return g;
```

The algorithm works with Memory a number of times with a search window of zero size. The search works by zooming in on the minimax value. Each call returns a bound on the minimax value. The bounds are stored in upper bound and lower bound, forming an interval around the true minimax value for that search depth. When both the upper and the lower bound collide, the minimax value is found. CFIA gets its efficiency from doing only zero-window P1, P2 searches, and using a "good" bound variable to do those zero-window searches, called with a wide search window, as in (root, -∞, +∞, depth), making sure that the return value lies between the value of P1, P2. In CFIA a window of zero size is used, so that on each call (P_1,P_2) will either fail high or fail low, returning a lower bound or an upper bound on the minimax value, respectively. Zero window calls cause more cutoffs, but return less information - only a bound on the minimax value. To nevertheless find it, CFIA has to call (P_1,P_2) a number of times, converging towards it.

In order to work, CFIA needs a "first guess" as to where the minimax value will turn out to be. The better than first guess is, the more efficient the algorithm will be, on average, since the better it is, the less passes the repeat-until loop will have to do to converge on the minimax value. If you feed CFIA the minimax value to start with, it will only do two passes, the bare minimum: one to find an upper bound of value x, and one to find a lower bound of the same value.

Typically, one would call CFIA in an iterative deepening framework. A natural choice for a first guess is to use the value of the previous iteration, like this:

CHESS ALGORITHM-II

Function

```
iterative_deepening(root : node_type) : integer;
```

```
firstguess := 0;
for d = 1 to Max_Search_Depth
do
firstguess := CFIA (root, firstguess, d);
if
 times_up()
then
   break;
return
   firstguess;
```

The algorithm CFIA (n, f), which stands for something like Memory-enhanced Test Driver with node n and value f. CFIA is the name of a group of driver-algorithms that search minimax trees using zero window P1, P2 with memory calls. CFIA is simple in that it only does zero window P1, P2 calls, making reasoning about the parts of the tree that get traversed easier than with algorithms that use wide window calls, such as NegaScout and the standard P1, P2. Actually, the difficulty in analyzing ordinary P1, P2 was precisely the reason why Pearl introduced his Test in the first place. Especially in a parallel setting the simplicity of CFIA compared to NegaScout is valuable. Designing and debugging a parallel search routine is a complex affair. CFIA only needs a zero window search, a Test. Instead of two bounds, CFIA needs one. In NegaScout, when new values for the search window become available they have to be communicated asynchronously to the child processes; in CFIA you simply abort an entire sub tree when a cutoff happens. Furthermore, the recursive search code does not spawn re-searches anymore. All re-searching is done at the root, where things are simpler than down in the parallel tree. The large body of research on parallelizing is directly applicable to CFIA instances, since they use zero-window (P1,P2) calls to do the tree searching.

CFIA framework is derived from the K. Coplan's C* algorithm. In CFIA terms the idea of K. Coplan's C* algorithm is to bisect the interval formed by the upper and lower bounds, reducing the number of (P1,P2) with memory calls. On the down side, bisection yields a value for the search window, P2, that turns out to be not as efficient as CFIA's choice. But still, Weill's work indicates that it is worthwhile to experiment with variants on CFIA's choice of pivot value; leaving plenty span for more research.

The transposition table access code is the same as what is used in most tournament chess programs.

CHESS ALGORITHM

Function

```
Memory(nt : node_type; P1, P2, s : integer) : integer;
if  retrieve(nt) = = OK
then /* Transposition table lookup */
if
nt.lowerbound >= P2
then return
nt.lowerbound;
if nt.upperbound <= P1
then return
nt.upperbound;
P1: = max(P1, nt.lowerbound);
P2: = min(P2, nt.upperbound);
if
d = 0
then
g : = evaluate(nt); /* leaf node */
else if
nt = = Maxnode
then
g : = -∞; a := P1; /* save P1 value */
c : = firstchild(nt);
while (g < P2) and (c != nochild)
 do
g : = max(g, P1, P2)Memory(c, a, P2, d - 1));
a : = max(a, g);
c : = next(c);
else
```

```
/* nt is a minnode */
g : = +∞; b := P2; /* save P2 value */
c : = firstchild(nt);
while (g > P1)
and
(c != nochild)
do
g : = min(g, P1, P2)Memory (c, P1, b, s - 1));
b : = min(b, g);
c : = next(c);
/* Transposition table storing of bounds */
/* Fail low result implies an upper bound */
if g <= P1
then
 nt.upperbound : = g; store nt.upperbound;
/* Found an accurate minimax value - will not occur if
called with zero window */
if g > P1 and g < P2
then
nt.lowerbound : = g; nt.upperbound : = g; store
nt.lowerbound, nt.upperbound;
/* Fail high result implies a lower bound */
if
g >= P2
Then
 nt.lowerbound : = g; store nt.lowerbound;
return g;
```

Transposition table access takes place in the retrieve and store calls. The lines around retrieve make sure that if a value is present in the table, it is used, instead of continuing the search.

The main procedure is CFIA -MAIN and a function, called CFIA -EXTEND, creates a sub tree followed by a proposed set enumeration. CFIA-EXTEND are executed recursively to create all item sets under the root item sets. The New Child function creates a child item set. For instance, NewChild(V,U) creates a child item set VU of a parent item set V, and adds the new child in a hash table. The TaxChild function returns the taxonomy-based child itemsets of that GI. CFIA generates the join-based child itemsets. The function called CFIA-PROPERTY, checks for conditional properties of CIs and makes the operations with the generated itemset. Following the CFIA algorithm, it will result in a tree of all CFIs.

CFIA-MAIN (Database,Taxonomy,minsup)

1. Root = Null Tree //Root node of set enum
2. NewChild (Root, FIs from second level of taxonomy)
3. CFIA -EXTEND(Root) CFIA - EXTEND(Father)
4. for each F_i in Father.Child
5. C = TaxChild(Fi) // taxonomy-based child itemset
6. if supp(C), ≥ minsup then
7. CFIA -PROPERTY(Nodes,C)
8. for j = i+1 to |Father.Child| // join-based child itemset
9. $C = F_i \cup F_j$
10. if supp(C) , minsup then
11. CFIA -PROPERTY(Nodes,C)
12. if F/ Child ≠ NULL then CFIA -EXTEND(Fi) CFIA -PROPERTY(Node,C)
13. if $t(F_i) = t(F_j)$ and Child(Fi)= θ; then
14. Remove(F_j); Replace(F_i) with C
15. else if $t(F_i)$. $t(F_j)$ and Child(F_i)= ; then
16. Replace(Fi) with C
17. else if $t(F_i)$. $t(F_j)$ then
18. Remove(Fj); if Hash(t(C)) then NewChild(F_i,C)
19. else if Hash(t(C)) then NewChild(F_i,C)

5. Experimental Results

In our experiment, the CFIA algorithm is coded in JAVA language and the experiment was done on a Pentium IV with 640Mb of main memory running Windows XP operating system. The real datasets are used in our experiment. Real dataset from the Machine Learning Database Repository, i.e. chess, are used with our own generated taxonomies. 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 minsup lowers. The higher the ratio is, 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 CFIA in real situations.

Using the data generator, generate chess datasets as described below.

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

Moreover, in the index, each axis is divided into a number of equal intervals that is fixed by t.

First compare the query performance of the algorithm CFIA and path tree, pruning. Then, focus on evaluating the efficiency of the pruning technique.

Implement the first set of experiments in the following experimental setting:

1) The cardinality of datasets is set to 1×10^n and 8×10^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 = 1×10^n.
(b) Cardinality = 8×10^n.

Anti-correlated datasets

(a) Cardinality = 1×10^n.
(b) Cardinality = 8×10^n.

The results of experiments observed in this case where the datasets are independent; each algorithm needs to spend more query time. It is mainly because there is more space set in anti-correlated datasets.

In the series of experiments, the impact of the pruning technique on the query performance of CFIA. The pruning technique can optimize the CFIA algorithm in most cases and the pruning effect becomes more evident as the cardinality of datasets increases.

The superiority of CFIA over the naive solutions is more marked for anti-correlated datasets than for independent ones. The main reason being naive solutions directly run CFIA on datasets and requires more query time for anti-correlated datasets. The efficiency of Query Time Reduction Ratio (QTRR) in CFIA can further be investigated against naive solutions.

6. Conclusions And Further Research

CFIA can mine only a small set of closed frequent item sets and results in reducing computational time. The advantage of CFIA 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. Further research intends to propose a method to extract only a set of important rules from these closed frequent item sets. The results at pilot level also need much more experiments with new methodologies.

References

[1] Agrawal. R, Imielinski. T, Swami. A.N, *"Mining association rules between sets of items in large databases"*, In Buneman, P., Jaodia, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C. 207-216, 1993.

[2] Srikant. R, Agrawal. R, "Mining generalized association rules", Future Generation Computer Systems, 161-180, 1997.

[3] Hipp. J, Myka. A, Wirth. R, Guntzer. U, *"A new algorithm for faster mining of generalized association rules"*, In: Proceedings of the 2nd European Conference on Principles of Data mining and Knowledge Discovery (PKDD '98), Nantes, France 74-82, 1998.

[4] Chung. F, Lui. C, *"A post-analysis Framework for mining generalized association rules with multiple minimum supports"*, Workshop Notes of KDD'2000 Workshop on Post Processing in Machine Learning and Data Mining, 2000.

[5] Han. J, Fu. Y, *"Mining multiple-level association rules in large databases"*, Knowledge and Data Engineering 11, 798-804, 1999.

[6] Lui. C.L, Chung. F.L, *"Discovery of generalized association rules with multiple minimum supports"*, In : Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD '2000), Lyon, France 510-515, 2000.

[7] Shintani. T, Kitsuregawa. M, *"Parallel mining algorithms for generalized association rules with classification hierarchy"*, In Proceedings of the 1998 ACM-SIGMOD,International Conference on Management of Data, 25-36, 1998.

[8] Michail. A, *"Data mining library reuse patterns using generalized association rules"*, In International Conference on Software Engineering, 167-176, 2000.

[9] Hwang. S.Y, Lim. E.P, *"A data mining approach to new library book recommendations"*, In Lecture Notes in Computer Science ICADL 2002, Singapore, 229-240, 2002.

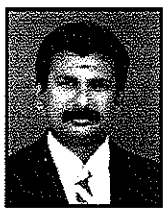
[10] Sriphaew. K, Theeramunkong. T, *"A new method for finding generalized frequent itemsets in generalized association rule mining"*, In Corradi, A., Daneshmand, M., eds.: Proc. of the Seventh International Symposium on Computers and Communications, Taormina- Giardini Naxos, Italy 1040-1045, 2002.

[11] Pasquier. N, Bastide. Y, Taouil. R, Lakhal. L, *"Discovering frequent closed itemsets for association rules"*, Lecture Notes in Computer Science 1540, 398-416, 1999.

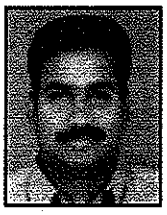
[12] Pasquier. N, Bastide. Y, Taouil. R, Lakhal. L, *"Efficient mining of association rules using closed itemset lattices"*, Information Systems 24 25-46, 1999.

[13] Zaki. M.J, Hsiao. C.J, *"CHARM: An efficient algorithm for closed itemset mining"*, In Grossman, R. Han, J. Kumar, V. Mannila, H. Motwani, R.Eds. Proceedings of the Second SIAM International Conference on Data Mining, Arlington VA (2002)

Author's Biography

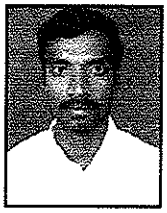


Dr. E. Ramaraj *is presently working as a Director, Computer Centre at Alagappa University, Karaikudi, Tamilnadu, India. He has 20 years of teaching experience and 5 years research experience. He has presented research papers in more than 20 national and international conferences and published more than 30 papers in national and international journals. His research areas include Data mining and Network security.*



R.Gokulakrishnan *is a Ph.D part time candidate in the Department of Computer Science and Engineering, Alagappa University, Karaikudi,*

Tamilnadu. He received his Master degree in Computer Applications in 1996. He has more than 4 years of industrial experience. Currently he is working as a Lecturer in Information Technology at J.J.College of Arts and Science, Pudukkottai, Tamilnadu. He has participated and presented the research papers in various National and International level conferences. His research interests include Data mining and Information system. University Grants Commission, New Delhi, recommended his candidature for International Commonwealth Scholarship in the year 2007.



***K.Rameshkumar** is presently doing Ph.D Fulltime at Department of Computer Science and Engineering, Alagappa University, Karaikudi. Tamilnadu. He has more than 3 years of industrial experience. He has participated and presented the research papers in various national and international level conferences. His research interests are Data mining, E-security and Quantum computing.*