Hash Algorithm for Finding Associations between Genes

P. Asha¹ and S. Srinivasan²

¹Research Scholar, Computer Science and Engineering Department, Sathyabama University, Chennai, Tamilnadu, India.
²Professor and Head of the Department, Computer Science & Engineering, Anna University, Regional Centre, Madurai, Tamilnadu, India.

DOI: http://dx.doi.org/10.13005/bbra/1679

(Received: 10 February 2015; accepted: 13 March 2015)

Association rules are those that narrate the relationships prevailing between attributes present in the database. Every rule mining algorithm generate promising items (frequent items) from which, the rules are framed. These rules try to state the items that are most related and how much one item is closer and depending on the other item. But the rules generated are enormous in number. Filtering out the useful patterns becomes difficult. The paper proposes a Hash based algorithm for extracting only the fruitful patterns at a faster rate. The work has been done using R language, and executed in R data mining Toolkit. Comparative study of Hash algorithm with respect to other algorithms shows that the Hash algorithm behaves better than all the other existing algorithms. It has been tested against various benchmark datasets like Adult, Genome, Cancer datasets using various rule interestingness measures like Lift, Confidence, Interest, Support etc.

Key words: Data mining, Associations, Rule Filtration, Interestingness Measures, Genes.

Association rules for example in case of marketing analysis, measures the closeness or relationships that exists between various products bought by some customer. Experimentation on finding the frequent items and then their close associations seems to be difficult and especially when the database size is large⁷,⁶⁹. The algorithm that is used for finding associations⁵ between items is Apriori Algorithm. Multi-scans is the negative side of Apriori and even with other algorithms⁶,²⁷.

The objective of the work is to propose an effective and efficient mining³³,³⁶ of closely associated patterns. This explains on how the technique can find a better place in the area of medicine and how effectively it has operated over it and proved itself to be better than others. It aims at finding the associations between the human genes, means when one gene gets affected, which are the other genes that do get affected along with it.

Association Rule mining finds wider application in the field of Genomics³³, DNA Analysis³¹, and Bioinformatics, Clone identifications in Software Engineering field, Marketing, and Financial Analysis etc. Both positive and negative rule mining is done, because sometimes the rarest occurring element becomes more important than the positive ones. Finding of effective rules depends on various interestingness measures like Lift, Support, Leverage, Conviction, Confidence etc.

Comparative study of various algorithms with respect to Hash algorithm using benchmark datasets has been done. Various parameters

* To whom all correspondence should be addressed. E-mail: ashapandian225@gmail.com
considered are support, confidence, and computation time, number of Rules and Efficiency. Review on existing system

Farah Hanna AL-Zawaidah, Yosef and Hasan proposed that the repeated disk overhead can be reduced by reducing candidate itemset (size related). Associated rules were generated from relational databases and data warehouses using the basic Apriori. It doesn’t tell about the processing speed and the efficiency of the rules generated.

Jia Ronga, Huy QuanVua, Rob Lawb, Gang Lia said that different filtering methods were used for rule grouping. Rules generation is applied only to smaller database and especially only for tourism dataset and not for other datasets. Yang Xiang et al. proposed rule grouping through which we can identify best rules. After which, various distributed and parallel algorithms rooted up.

Xindong et al. worked out BigData concept. They gave HACE algorithm for handling such a higher volume and which id heterogeneous in nature. Chuang H et al. started up with basic hash definitions and its applicability to bigger datasets.

Confidence and support based cropping of items were put forth by Shinji et al. They focused only on the reduction in transaction and not efficient rule generation. Huan Wu et al. used count method, which followed <itemset, Tids> structure for storing the data. The method counts each candidate itemsets only once. The disadvantage of the system is that, it spends more time for building <itemset, Tids> structure which may never be used for further processing except at the initial phase.

Jayalakshmi et al. worked on sequential maximal pattern mining and made better pattern hunting. Kannika et al. proposed a new method for generating rules based on lift ratio. Interestingness measures are controlled with minimum level and the generated rules are filtered. Support and confidence are set and lift factor is used for filtering the rules.

Hash algorithm

Let D be the database which holds Tr number of transactions (trans). That is, D= \{Tr₁, Tr₂, Tr₃…Trₘ\}. Every transaction holds few items I₁, I₂, I₃… Iₘ, which could be present in any combination. The item list is denoted by IL = \{I₁, I₂, I₃… Iₘ\}. Having set E, such that E ⊆ IL then it implies that E ⊆ Tr. The association rule is viewed as E => F, where E ⊆ IL, F ⊆ IL, and E ∩ F = ∅. Support up to (E=>F) is given by s% of the transactions in D containing E U F.

Confidence, confid (E=>F) is c% of trans in D that contains E U F.

That is,
Then, E=>F is a association rule if
Confid (E=>F) > minconfid,

Matrix Generation

The matrix S={Sᵢⱼ} = m x n, where p=p₁, p₂,p₃… pₘ and q=q₁, q₂,q₃…qₙ,

Sᵢⱼ = \begin{cases} 
0 & \text{if } I_q \in Tr_p \\
1 & \text{if } I_q \notin Tr_p 
\end{cases}

Example 1:

Let us assume that D={Tr₁,Tr₂,Tr₃,Tr₄} and IL contain 5 items (represented as It) , which are the one-frequent itemsets present in the database D.

Suppose Tr₁ = \{I₅, I₄, I₅\}, Tr₂ = \{I₂, I₁, I₃\}, Tr₃ = \{I₁, I₅\}, Tr₄ = \{I₃\}.

Then the matrix corresponding to the example 1 is,

\[
S = \begin{bmatrix}
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Any item’s supt count is calculated by using,

\[
\text{Supt Count } (Iₖ) = \sum_{k=1}^{n} S_{ₖₙ}
\]

And Cᵢ is the candidate set which consists of 1- Frequent itemsets. Then compare every candidate’s supt count with the minimum (min-sup) support threshold. Supt_count \( \{ Iₖ \} \geq \text{min-sup} \) and if so then we conclude that \( Iₖ \in L₁ \), (Level 1: 1-Freq itemset). The next step is to frame 2-Frequent itemsets and then 3-Frequent itemsets etc… until the candidate satisfies the min-sup specified.

C₂ = L₂ \( \bowtie \) L₂, and this leads to 2-Frequent itemsets. Next,

C₃ = L₂ \( \bowtie \) L₂
and the algorithm proceeds and terminates when no further combinations can be made. Then finally

$$L = \bigcup_{i=1}^{n} L_i$$

where \(L\) contains the complete set of frequent items. After which, the association rules are generated. Remember every sub-set present in the frequent item set should be frequent too. That is,

$$\forall i \subseteq L : L_k - \{i\} \subseteq L$$

**Methodology**

Initially, scan the dataset and create candidate itemset. From that create a hash table for 2 itemsets and then find out large itemsets from the hash table. From the frequent itemsets retrieved now we make association rules. As it may produce many numbers of rules which may be redundant and insignificant, it is a must to remove all those and get only the best rules. Finally we can group similar rules (Fig. 1).

The proposed methodology tries to overcome the problems of the existing system. It contains four modules:

**Frequent itemset generation**

Choose the input dataset and minimum support count, min_sup. Create candidate item-1 and Hash table for generating candidate item-2. Then create large itemset from candidate item-1 and make large item-2 from hash table-1. Next create Hash table-2 from Hash table-1. Repeat the process for creating hash table and large itemset.

**Hash table**

Read the data for candidate item 2 from dataset at the 1st scan. Then we create hash data structure with min_sup and item name (Fig. 2) for 2 itemset. Each item in the separate column and row uses hash function \((x*y) \mod L_n\). Then from the first hash table we generate next hash table (Fig. 3). Generate the hash table that contains a suppt_count > min_sup and discard others possessing support count less than the
min_support count. At the same time truncate the transaction with no frequent item (marked with red color in Fig. 2). Before creating the hash table element, remove the item having less support count from the transaction. (marked with blue circles in Fig. 2).

**Rule generation**

Then we try to retrieve the rules that were associated based the resultant frequent items. After which, these rules are filtered and refined using various interestingness measures like Confidence, Lift, Leverage, Laplace, Interest Factor etc.

**Grouping rules**

Similar rules are joined together based on the Rule Head. Rule Head contains the same value group that rules into one. Hence we try to reduce the generated rules based on grouping.

**Hash algorithm**

The algorithm is as follows:

\[
\text{Hs Alg}() \\
\{
\text{The algorithm starts with a candidate itemset of one.}
\}
\]

C_k: Candidate item set of size k

L_k <- frequent 1-itemsets

Generate candidate for every L_k do begin

C_{k+1} <- candidate(L_k) #New candidates

Join Step: C_k is generated by joining L_{k-1} with itself

For all transactions t “D” do begin

C_k <- subset [C_k, t] #Candidates contained in t

For all candidates c “C_i do c. count++;

Prune Step: Any (k-1) –Subset of infrequent itemset must be infrequent.

L_k {c “C_i do c. count>= minsup}

Final L_k as frequent item from dataset.

\}

**Grp_rul() // Grouping similar rules**

\{

Rmerge = merge(RevDup(G1), RevDup(G2))

MG = group_by_RHS(Rmerge)

For each G “ MG { agent = find_agent(G)

DR = sort_by_4condition(agent)

Return (DR)
\}

**Rul_fil() //Rules generated after filtering**

\{

R=F_k > {Confidence, Lift, Leverage, Interest Factor}

\}

// Sample Code

colnames(spm)<-c(“x”)

while(j<=m)

{ candidateset<-as.data.frame(itemset[j,])

# print(candidateset)

counttot<-0 i<=1

counttot<-sp(itemset,data,sc)

spm[j,]<-counttot j<j+1 }

rt<-as.data.frame(cbind(itemset,spm))

sorted<-rt[order(-rt$x),]

**Rule Interestingness measures and Grouping**

L_h means the instances present at the left (LHS) hand side of a rule and R_h means the instances at its right (RHS) hand side. Let Proba represents the probability.

1. Confid = P (L_h U R_h) / P (L_h)

2. Supt = P (L_h U R_h) / N_{total}, where N_{total} represents the total number of instances.

3. Lift (L_h U R_h) = count (L_h U R_h) / count (L_h) * supp (R_h)

4. Leverage (L_h->R_h) = supp (L_h U R_h) - supp (L_h)*supp(R_h)

**Example**

Consider Fig. 4 as an example Transaction database. Find the candidate item set -1 using normal Apriori algorithm (Fig. 5). Next generate L1 (Fig. 6) from candidate item -1 with itemset satisfying a supr_count value >= min_sup and other itemsets were removed. After this step create hash table for each itemset, due to the presence of large number of itemset combinations in item set-

<table>
<thead>
<tr>
<th>Tid</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T200</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T300</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T400</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T500</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T600</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Fig. 4. Transactional Data Base**

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supp</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Fig. 5. Candidate item-1**
For the next iteration we need 3 combinations. Remove transaction T500 because it has only one large 2-itemset. Transaction T300 contains A only once, B and D thrice in 3 combination pattern. So we can remove A from the transaction, because we need more than 2 combinations in next step. Similarly, we delete A, F in transaction 200 and 400, E in transaction 100 and hence in this way we can create next step of data easily using the hash table (Fig. 9). Hence the final step would be a simplified database and as a result we achieve data reduction.

**Sparse matrix representation**

Let T1, T2…..Tn represent the patient number and VWF, MSH2 etc., represent the Colon-Cancer infected gene [17], [20]. The value 1 indicates the presence of infected gene for that particular patient. And the input database contains approximately 2 lakh records and hence applying the mining algorithm to it would be too difficult. Hence to reduce that database into a smaller size we try to represent it in a vertical format, stated as, for example take Gene wise listings, say VWF, now find out for which all patients it is present. Proceeding like this would probably reduce the database size as repetitive infected gene would appear in the database.

Now apply the Apriori Algorithm. Find 1 (individual) frequently occurring gene from which two closely associated genes are generated. Next from the two-combinations generate the three combinations, that is, the three individual genes that are in close relation and iterate the procedure for multiple combinations and in every stage sum up its presence in the database. Next in every stage filter only the combination of genes which carries
a sup_count >= user specified threshold. Hence, our final resultant would be the best combination, i.e. only the frequently occurring [13], [15], [16], [18], [19] infected genes would sustain as a result of the algorithm. Next generate the association rules from resultant data. Now enormous amount of rules might exist which would be unproductive, redundant or insignificant.

So try to eliminate the unfruitful elements and retain only the most promising rules. Such a technique is termed as Filtration of rules.

**Rule interestingness measures**

There are two basic Interestingness Measures, Subjective and Objective measures. Subjective measures of Interestingness states the belief of the user. It is categorized into two, namely Actionability and Unexpected. Unexpected measures states that the pattern found while discovery may be much astonishing and useful to the user. Actionability measure is that the user act over the pattern to gain advantage of it. Objective measures work on the data and the structure of rule in a discovery procedure. Support and Confidence are objective measures. It generates best rules which may be or may not be much interesting to the user. So, to obtain the best and highly interesting rule, it is important to have a combined format, which is the combination of Subjective and Objective measures.

**Rule Interest Measures**

Finding the best and top rules seems to the biggest motto of rule mining. Basically used measures are: Confidence, Support.

*Discriminability* is one other measure which is used to infer how much the rules are able to distinguish/classify one category from the other.

\[
\text{Discriminability} = 1 - \frac{\text{Proba}(L_h \cup R_\ell) - \text{Proba}(L_h \cup R_\ell)}{\text{N}_{\text{total}} - \text{Proba}(R_\ell)}
\]

If discriminability is 1 then that implies a strong classification has been made. That is \(\text{Proba}(L_h) = \text{Proba}(L_h \cup R_\ell)\).

**Piatetsky-Shapiro Measure**

Gregory Piatetsky-Shapiro has put forth three criteria that every interestingness (Intr) measure of a rule should comply with.

Criteria 1:

Intrst measure should be zero if \(\text{Proba}(L_h \cup R_\ell) = \frac{\text{Proba}(L_h) \times \text{Proba}(R_\ell)}{\text{N}_{\text{total}}}\)

Criteria 2:

Intrst measure must be upgraded monotonically along with \(\text{Proba}(L_h \cup R_\ell)\).

Criteria 3:

Intrst measure must be upgraded monotonically along with \(\text{Proba}(L_h \cup R_\ell)\).

---

**Table 1.** Calculation of execution time, Memory usage - Support

<table>
<thead>
<tr>
<th>Supp Freq.</th>
<th>Items</th>
<th>Ass. Rule Count</th>
<th>Exec Time</th>
<th>Memory Usage</th>
<th>Conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>77</td>
<td>356</td>
<td>62</td>
<td>712</td>
<td>50</td>
</tr>
<tr>
<td>0.4</td>
<td>32</td>
<td>301</td>
<td>54</td>
<td>66</td>
<td>50</td>
</tr>
<tr>
<td>0.6</td>
<td>30</td>
<td>29</td>
<td>27</td>
<td>16</td>
<td>50</td>
</tr>
<tr>
<td>0.8</td>
<td>21</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>50</td>
</tr>
</tbody>
</table>

**Table 2.** Computation time, memory usage - Transactions

<table>
<thead>
<tr>
<th>Trans Freq.</th>
<th>Items</th>
<th>Ass. Rule Count</th>
<th>Exec Time</th>
<th>Memory Usage</th>
<th>Conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>69</td>
<td>437</td>
<td>81</td>
<td>1116</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>1011</td>
<td>138</td>
<td>362</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>65</td>
<td>351</td>
<td>58</td>
<td>647</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>59</td>
<td>290</td>
<td>49</td>
<td>511</td>
<td>50</td>
</tr>
</tbody>
</table>

**Table 3.** Rules after removal of Redundancy

<table>
<thead>
<tr>
<th>Total no of Rules when Support = 31% Before Filtering</th>
<th>After Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>26</td>
</tr>
</tbody>
</table>

**Fig. 11.** Sample set of generated rules
Criteria 3: Intrust measure must be degraded monotonically along with every $\text{Proba}(L_h)$ and $\text{Proba}(R_h)$. Intrust is defined as,

$$\text{Intrust} = \text{Proba}(L_h \cup R_h) - \frac{(\text{Proba}(L_h) \cdot \text{Proba}(R_h))}{N_{\text{total}}}$$

Usually the value of Intrust is >0. If Intrust = 0, then the chance is better than the rule obtained and if Intrust = <0, then the chance is somewhat (quiet) better than the rule obtained.

**Performance analysis**

The work has been implemented using R language. Table I shows the characteristics of our infected-gene dataset, which displays the set of association rules retrieved, req-items generated, total execution time, memory usage with respect to different support thresholds and Table II with respect to Transactions.

Table III shows the total number of rules generated when the support=0.09. Here, we can observe that the number of rules has been reduced from 120 to 26 after redundancy removal and refinement by the various rule interestingness measures. That is, if support count increases, the number of rules generated also increases. But after certain range, the number of rules generated can also be nil. For the support value > 4/6/8, no rules were generated.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>APRIORI Rules</th>
<th>ECLAT Rules</th>
<th>PVARM Rules</th>
<th>NRRM Rules</th>
<th>Hash Based Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groceries</td>
<td>80856</td>
<td>60246</td>
<td>10026</td>
<td>20126</td>
<td>7432</td>
</tr>
<tr>
<td>Ser Prediction</td>
<td>90032</td>
<td>80125</td>
<td>18001</td>
<td>33511</td>
<td>11235</td>
</tr>
<tr>
<td>Genome</td>
<td>14256</td>
<td>5012</td>
<td>700</td>
<td>3148</td>
<td>471</td>
</tr>
<tr>
<td>Adult</td>
<td>48000</td>
<td>39256</td>
<td>12456</td>
<td>28356</td>
<td>7800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Hash Based APRIORI (sec)</th>
<th>APRIORI Algorithm (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groceries</td>
<td>29</td>
<td>0.07</td>
</tr>
<tr>
<td>Ser Prediction</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>Genome</td>
<td>10</td>
<td>0.01</td>
</tr>
<tr>
<td>Adult</td>
<td>23</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Fig. 12. Rule with conclusion = "VWF"

Fig. 13. Comparison of Various Algorithms
We can even list out the rules, whose RHS is a particular gene, (actual/required gene we expect to know about) that is the resultant gene received as a result of various infected gene combinations. In Fig. 12, we try to list the rules where RHS=VWF. The resultant is only one rule that has satisfied the specified constraint.

Comparative study of Apriori, Éclat mining algorithm, PVARM (Partition based Validation for Association Rule Mining), NRRM (non-redundant rules method) and Hash Based with respect to various datasets like adult, genome, groceries and SER prediction are done. The reason behind these dataset selection is that, these dataset have different transaction size, item size (Table IV).

The Fig.13 shows bar chart of comparison of various algorithms. Horizontal axis has each algorithm in its side and y axis has support level 0 to 1 Lakh. In this study we fixed minimum confidence=50% and lift=20% and monitored their execution time. From Fig 13, we infer that the Hash algorithm performs well when compared to all others, with most interesting rules and non-redundant rules.

Table V shows the time taken by the Apriori and Hash based algorithm, for the entire data mining task.

From Fig. 14 we can conclude that Hash algorithm performs better than the Apriori, that is the entire computation with very less time and excels even with larger datasets (Table V).

CONCLUSION

We conclude that, the Hash algorithm has performed well based on the performance analysis stated with various parameters. The algorithm scales well for bigger databases too. We have made a thorough analysis of gene associations and with lesser time and accurate combinations and frequencies.

REFERENCES


