RESEARCH REPORT

Mining Association Rules with Item Constraints

Authors:	Student ID:
Yao, Rui	A1680375
Wan, Sichen	A1698799

Table of Contents

1. Introduction

Association rule is one of important research frontier in the field of data mining. Association rule data mining aims to discover unknown rules with potential values by analysing massive data. For instance, consumption habit can be found by discovering associations among different goods(items). According to those rules, goods can be re-categorised to boost consumption whilst the inventory can be rearranged to accommodate specific market requirements.

According to our survey on association rule with item constraints, there are two main problems we should tackle with. The first obstacle is the limited constraint type, which is due to the data complexity in this complicated world. The second hindrance is that constraint cannot integrate well with algorithm. Since different constraints require different professional knowledges, there is not general integrated method for current efficient algorithms to maintain the high performance on different tasks. Therefore, this research attempts to find out better methods to improve the performance on integration between professional knowledges and algorithms. In this research report, boolean mathematics expressions on whether items exist or not are suitable for association rules.

In order to see further, we should stand upon the shoulders of giants. Two prevalent and excellent algorithms on association rules are discussed and analysed in section 2. One optimised algorithm is proposed and described with detailed analysis in section 3. In section 4, expectation on future work is presented. On the basis of previous sections, a summary is concluded in the last section.

2. Previous Work

In this section, we are going to conduct a comprehensive review on two well-known algorithms, namely, Apriori algorithm and FP-growth algorithm. They are traditional algorithms whose aim is to calculate frequent item sets with different methods.

We can decompose the mining association rules into two subsections: Firstly, we find out all possible combinations of items whose support outnumber the minimum support (threshold). And those eligible combinations are called as frequent item sets. Secondly, we search for desired association rules by computing those frequent item sets. [1] Since all of them are designed to mine frequent item sets, corresponding association rules need be found from frequent item sets. To be more specific, If EFGH and EF are both frequent item sets, we can judge whether the rule EF $\Rightarrow$ GH is justifiable or not by getting ratio between former's support and latter's support. If that ratio is greater than or equal to the minimum confidence, this rule is justifiable. [1]

In this section, we will provide overview of given algorithm. We will introduce how given algorithm works. What kind of underlying algorithmic thinking given algorithm uses. The detailed implementation process and efficiency analysis will be also presented. We will try to conduct a comparative analysis for two algorithms on the basis of our current knowledge and cognition.

2.1 Apriori Algorithm

2.1.1 Algorithm Theory

Apriori algorithm was firstly proposed by Rakesh Agrawal & Ramakrishnan Srikant in VLDB (Very Large Data Bases Conference). They found that previous algorithms like AIS and SETM algorithms are inefficient in candidate generation due to too many unnecessary candidate items sets whose supports are lower than minimum support, which produces redundancy counting consequentially. Therefore, Apriori algorithm they proposed has two main goals:

- first goal is to shrink the amount of candidate item sets to fit the proper size.
- Secondly goal is to reduce the repetitive computations which have tiny contribution to find out the frequent item sets and determine the association rules.

In order to achieve those goals, Rakesh Agrawal & Ramakrishnan Srikant published their study results on Apriori algorithm in [1], [2], which can be summarised into one sentence that if some item set is frequent item set, all subsets of this item set are frequent item set as well. This principle can be also comprehended as if some item set is non-frequent item set, all super sets of this item set are non-frequent item sets. With this principle, we can effectively prune the unwanted candidate item sets when we search for the frequent item sets.

Apriori algorithm reduces the insignificant item sets on the basis of reliable prediction which comes from objective observation (prior knowledge). The name of this algorithm reveals the essence of this algorithm: “Apriori”, it can be considered as some knowledge can be acquired before having certain experiences. According to our logic and rationality, the item sets which have higher support tend to have sub item sets which also have higher support. Likewise, the item sets which have low support tend to have super item sets which have low support.

We will present the actual process of this algorithm in the following subsection.

2.1.2 Implementation and Pseudocode

In this subsection, we will display how Apriori algorithm works. As we mentioned above, there are two main steps: find the frequent item and then determine the association rules. We prepare a sample data for demonstration:

Transaction ID	List of Items
0000	{Apple}, {Banana}, {Cake}
1000	{Apple}, {Cake}

2000	{Apple}, {Donut}
3000	{Banana}, {Egg}, {Fish}
4000	{Cake}, {Donut}, {Egg}, {Apple}
5000	{Egg}, {Fish}

Premise: Minimum Support Ratio is 50%; Minimum Confidence Ratio is 50%.

Step①: Get the Minimum Support.

6 * 50% = 3, which means each item in the item set must be present at least 3 transactions.

Step②: Scan Total item set to count each candidate and get the support of each candidate. Then compare candidate support with Minimum Support.

The candidate set is {Apple, Banana, Cake, Donut, Egg, Fish}.

Items	Support
{Apple}	4
{Banana}	2
{Cake}	3
{Donut}	2
{Egg}	3
{Fish}	2

We will prune those items whose supports are less than Minimum Support. Therefore, {Banana}, {Donut} and {Fish} should be removed. {Apple}, {Cake} and {Egg} remain.

Step③: Then generate the new candidate based on the remain items.

The candidate set is {{Apple, Cake}, {Apple, Egg}, {Cake, Egg}}.

Step④: Repeat the **Step②**.

Items	Support
{Apple, Cake}	3
{Apple, Egg}	1
{Cake, Egg}	1

We will prune those items whose supports are less than Minimum Support. Therefore, {Apple, Egg}and {Cake, Egg} should be removed. {Apple, Cake} remains.

Step⑤: Get the frequent item set. Then find the association rules.

Review the formulates:

$Support(A \Rightarrow B) = P(A \cup B)$

$Confidence\ (A \Rightarrow B) = Support\ (A \Rightarrow B)/Support\ (A)$

Lift ($A \Rightarrow B$) = $Confidence\ (A \Rightarrow B)/Support\ (B)$ = $P\ (A \cup B)/(P(A)\ \ P(B))$

Association Rule	Support	Confidence	Confidence Ratio	Lift
Apple ⇒ Cake	3	3/4	75%	¼
Cake ⇒ Apple	3	3/3	100%	¼

Since the Minimum Confidence threshold is 50%, the confidences of two rules are both greater than 50%.

Thus, the final rules are:

First Rule: Apple ⇒ Cake

Second Rule: Cake ⇒ Apple

Pseudocode: This pseudocode is written with the inspirations in [1], [2] and [3]

Input	D-Transaction Database; $MinSupport$-Minimum Support Threshold
Output	L-Frequent Item sets

```
$L1 = frequent(1)item\ set(D);$      //Find out all one-item item sets whose frequent is greater than Minimum Support.
$for\ (k = 2;\ L_{k-1}\ != NULL;\ k++)\{$
$C_k = Apriori\_generation(L_{k-1})$ ;  //Generate the Candidate item sets with pruning.
$for(int\ i = 0; i < D.size();\ i++)\{$          //Scan each t in D to count the candidate
$C_t = subset(C_t, t);$    //Get the subset of t.
$for(int\ j = 0; j < candidate.size(); j++)\ \{\ candidate\ \in C_t$
$candidate.counter++;$
}
$L_k = \{candidate.counter \geq MinSupport, candidate\ \in C_k\}$
}
$return\ L = \{all\ frequent\ item\ sets\ L\};$
```

Function 1: $Apriori\_generation\ (L_{k-1}{:}\ frequent(k-1)\_item\ set)$

```
$for\ (each\ item\ set\ I1, I1 \in\ L_{k-1})\{$
$for\ (each\ item\ set\ I2, I2\ \in\ L_{k-1})\{$
$if\ (I1[1] == I2[1]\ \&\&\ I1[2] == I2[2] \ldots \&\&\ I1[k-1] == I2[k-1])\{$
$candidate = Join(I1, I2);$    //Join two item sets to generate candidate.
$if\ Infrequent\_subset\ (candidate, L_{k-1})\{$
$delete\ candidate;\}$   $else\{add\ candidate\ to\ C_k;\}$ //Prune the infrequent candidates.
}
$return\ C_k;$
```

Function 2: $Infrequent\_subset\ (candidate\ k\_item\ set;\ frequent(k-1)\_itemset)$

```
$for(each\ (k-1)\_subset\ sub\ of\ candidate)\{$
$if(sub\ \notin L_{k-1})\{return\ true;\}$        $else\{return\ false;\}\}$
```

2.1.3 Algorithmic Thinking & Efficiency Analysis

❖ **Backtracking**

Different algorithms are designed under the guidance of characteristic algorithmic thinking. Apriori algorithm aims to effectively find out the frequent item sets by joining desirable candidate item sets and pruning undesirable candidate item sets.
In this case, *backtracking* [4] is employed with constraint (Minimum Support). Apriori algorithm prunes the item set once such item set does not meet the support constraint whilst joins the item set whose support is greater than or equal to Minimum Support. In backtracking thinking, once eligible item set occurs, algorithm will continually search the subset of this item set. Contrarily speaking, further search will stop without second visit when the current item set fails to meet the constrain. In Figure 1, it is manifest that, when algorithm visits {Apple} whose support meets the constraint, subsets of {Apple} will be visited again. In the same principle, algorithm will stop any further actions on {Banana} since this item set is infrequent, which implies subsets of {Banana} do not belong to frequent item sets. Backtracking plays a pivotal role in Apriori Algorithm.

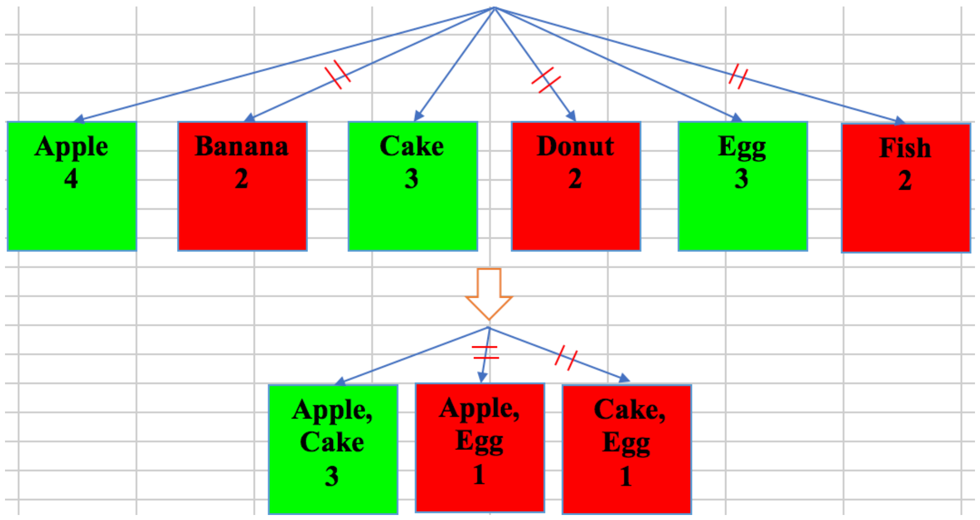

Figure 1.** Graph for Apriori with example data in **section 2.1.2

❖ **Quadratic Growth-Time Complexity**

According to the pseudocode provided in section 2.1.2, we can calculate the time complexity of Apriori algorithm.

$\Rightarrow O(n + n + n^2 + n + c) = O(\boldsymbol{n^2})$(Omit the linear time. Thus, only quadratic time stands for the time complexity)

2.2 FP-growth Algorithm

Since the Apriori Algorithm requires frequent scans on transaction item sets and candidate generations, the expense on input/output soars when the target database is huge. In order to get a better alternative algorithm, FP-growth Algorithm was proposed by Han and his teammates to break through bottleneck of Apriori Algorithm. [6]

2.2.1 Algorithm Theory

FP-growth Algorithm discovers a new way to mine the frequent item sets by changing the data structure instead of maintaining the generate-and-test paradigm that Apriori Algorithm adopts. The obvious novelty of this algorithm is on the frequent pattern tree

(FP-Tree), which is a derivation of Prefix tree. [6] The main idea of this algorithm has two steps:

- Construct FP tree and store item sets with necessary information.
- Get the frequent pattern by recursively mining frequent item sets from the tree.

Although the main idea is easy to be understood, the implementation of FP-growth is fairly difficult. In next subsection, we will try our best to reveal how FP-growth algorithm works with example.

2.2.2 Implementation and Pseudocode

In this subsection, we will show how this algorithm works. As we mentioned above, there are two main steps: Construct FP tree with item sets then recursively mine the FP tree. A sample data is prepared to demonstrate this part:

Transaction ID	List of Items
0000	{Apple, Banana, Cake, Egg, Fish, Oil}
1000	{Apple, Cake, Ginger}
2000	{Egg, Ice}
3000	{Apple, Cake, Donut, Egg, Ginger}
4000	{Apple, Cake, Egg, Ginger, Lamb}
5000	{Egg, Juice}
6000	{Apple, Banana, Cake, Egg, Fish, Pasta}
7000	{Apple, Cake, Donut}
8000	{Apple, Cake, Egg, Ginger, Milk}
9000	{Apple, Cake, Egg, Ginger, Noodle}

Premise:

Premise: Minimum Support Ratio is 20%; Minimum Confidence Ratio is 50%.

Step①: Get the Minimum Support.

10 * 20% = 2, which means each item in the item set must be present at least 2 transactions.

Step②: First Scan to get the support of each item, then erase items whose supports are less than Minimum Support. Then insert the eligible items into Header Table after sorting them in descending order.

Since {Ice}, {Juice}, {Lamb}, {Milk}, {Noodle}, {Pasta} and {Oil} only appear once, these items need to be removed. Therefore, we can construct a Header Table.

{Apple}	8
{Cake}	8
{Egg}	8
{Ginger}	5
{Banana}	2
{Donut}	2
{Fish}	2

Step③: Second Scan to Transaction Items again to remove the infrequent item, then reorder the items in each item set in the descending order.

Transaction ID	List of Items
0000	{Apple, Cake, Egg, Banana, Fish}
1000	{Apple, Cake, Ginger}
2000	{Egg}
3000	{Apple, Cake, Egg, Ginger, Donut}
4000	{Apple, Cake, Egg, Ginger}
5000	{Egg}
6000	{Apple, Cake, Egg, Banana, Fish}
7000	{Apple, Cake, Donut}
8000	{Apple, Cake, Egg, Ginger}
9000	{Apple, Cake, Egg, Ginger}

Step④: Map the item sets in Header Table to FP Tree in order to construct the Tree.

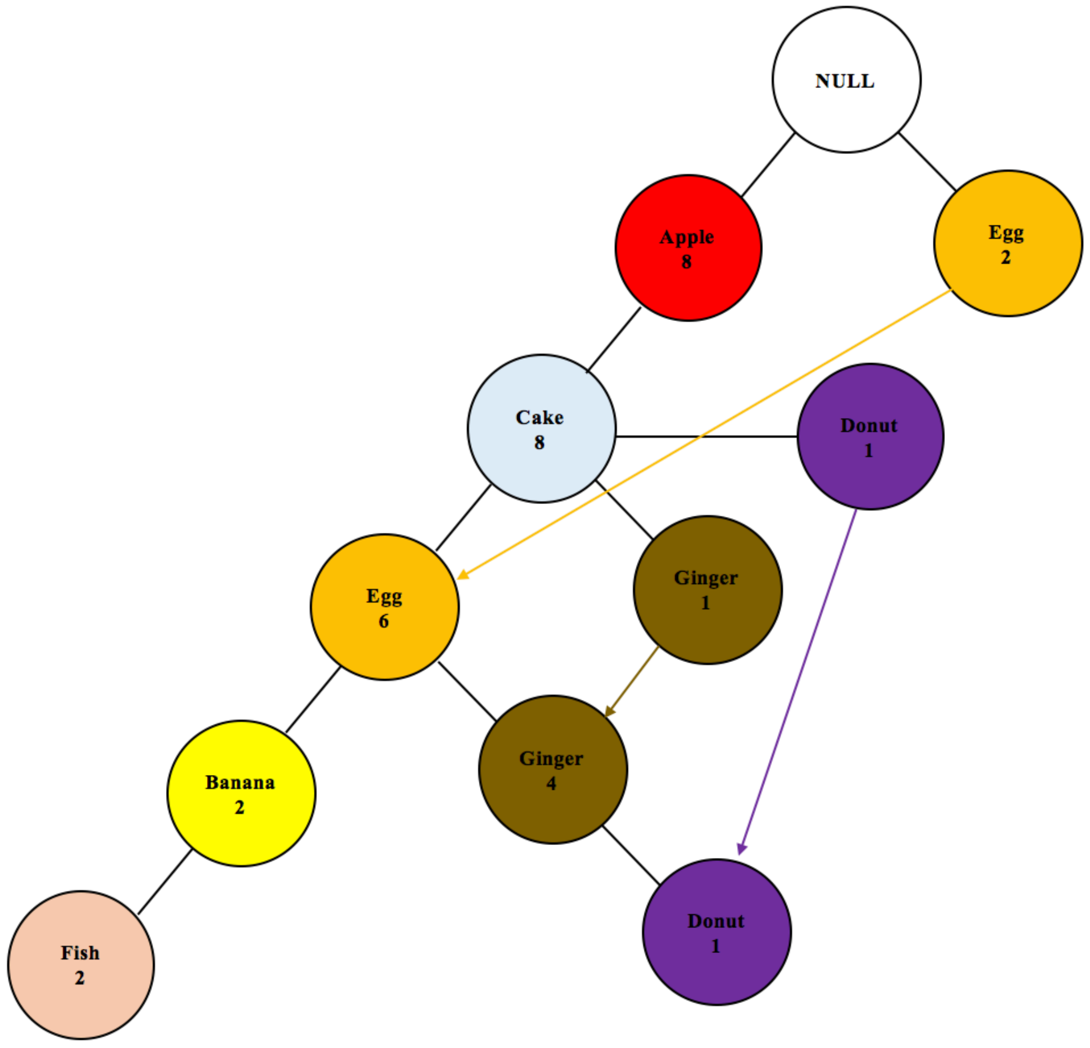

Figure 2. *FP-Tree with the example data in* ***section 2.2.2***

Step⑤ Mine the FP-Tree recursively to get frequent item sets.

According to the instructions in [6], "conditional pattern base" is the key to find out the underlying frequent item sets. The frequency of all items in the whole link need to be synchronised with the frequency of analysed item.

As for {Fish}, there is only one link: {Fish 2} - {Banana 2} - {Egg 2} - {Cake 2} - {Apple 2}. Therefore, we can figure out the frequent item sets of {Fish} include: {Apple 2, Fish 2}, {Cake 2, Fish 2}, {Egg 2, Fish 2}, {Banana 2, Fish 2}, {Apple 2, Cake 2, Fish 2}, {Apple 2, Egg 2, Fish 2}, {Apple 2, Banana 2, Fish 2}, {Cake 2, Egg 2, Fish 2}, {Cake 2, Banana 2, Fish 2}, {Egg 2, Banana 2, Fish 2}, {Apple 2, Cake 2, Egg 2, Fish 2}, {Apple 2, Cake 2, Banana 2, Fish 2}, {Apple 2, Egg 2, Banana 2, Fish 2}, {Cake 2, Egg 2, Banana 2, Fish 2} and {Apple 2, Cake 2, Egg 2, Banana 2, Fish 2}.

Frequent item sets for other item can be determined in the same way.

Step⑥ Find the association rules.

The association rules can be discovered in the same way shown in **Step⑤** of Apriori Algorithm in section 2.1.2.

Pseudocode: This pseudocode is written with the guidance in [6].

Input	D-Transaction Database; $MinSupport$-Minimum Support Threshold
Output	Milestone Output: fp-tree; Final Output: L-Frequent Item sets

```
***Function 1***: *FP-Tree-Construction (D, Child) {*
$L1 = frequent\ item(1)\ set(D);$     //Find out all one-item item sets whose frequent is greater than Minimum Support. Store eligible item sets into $L1$.
$L2 = L1.sort(descending\_order);$     //Sort the $L1$ into $L2$ in descending order in terms of Support.
$FP\_TREE * Root\_Node = NULL;$    //Initialise FP-Tree with NULL Root Node.
$for(int\ i = 0; i < D.size();\ i++)\{$//Reorder eligible item sets in D in $L2$ order.
$D[i] = D[i].sort(order\_of\_L2);\}$
$if((Root\_Node \rightarrow LeftChild) != NULL \mid\mid (Root\_Node \rightarrow RightChild) != NULL)$
$\{if\ (Child[i].ID == D[j].ID)$//If Root Node has child whose ID is the same as current item's ID.
$\{Child[i].Counter++;\}$
}
$else$    //If not, create a new node for current item.
$\{New\ New\_Child;$
$New\_Child.ID = D[j].ID;\ \}$
$New\_Child.Counter++;$
$D[j].Parent = Root\_Node;$
$New\_Child.Node\_Link \rightarrow Node(Node.ID == New\_Child.ID);$*//Create Node Link between items whose ID are identical in order to sum up the frequency of this item.*
$return\ FP\_Tree\_Construction\ (D, Child)$ ;}
```

```
***Function 2***: $FP - Tree - Mining\ (FP\_TREE\ myTree, Item\_Set)$
$int\ Temp\_Minimum\_Support\ = 0;$
$if(myTree.path < 2$ && $myTree.empty()\ ! = 0)$ { //Case for only one path exists.
$for(int\ i = 0; i <\ myTree.path.size(); i + +)\{$
$if(Node[j].Counter < Temp\_Minimum\_Support)\ \ (Node[j] \subseteq myTree.path)$
$\{Temp\_Minimum\_Support\ = Node[j].Counter;\}$
$return\ L3(All\ Node.Counter \geq Temp\_Minimum\_Support);$ }//Generate item sets $L3$ which consists of all nodes whose support are greater than or equal to temp mini support.
$if(myTree.path > 1)\{$        //Case for multiple paths exist.
//Construct conditional pattern base, according to this base, then construct FP-Subtree
return $FP - Tree - Mining\ (FP\_TREE\ myTree, Item\_Set);\}$
```

2.2.3 Algorithmic Thinking & Efficiency Analysis

❖ **Divide and Conquer** [6]

FP-growth Algorithm uses *Divide and Conquer* to avoid the redundant workload in scan and candidate generation. The algorithm decomposes the problem into several small subsidiary problems to find out all frequent item sets which share the same specific "suffix" frequent item. For instance, in order to find out all frequent item sets which contain item {Cat}, we should check whether {Cat} is frequent item or not. If it is frequent, then search and check whether those 2-item item sets which contain {Cat} are frequent or not. In the same way, each subsidiary problem can be divided into smaller subsidiary problems. In the end, results of all subsidiary problems can be merged together to get all frequent item sets which contain {Cat}. Divide and Conquer is the key strategy in FP-growth Algorithm.

❖ **Quasilinear Growth-Time Complexity**

According to pseudocode, it takes *O(nm)* to for item insertion. n is the sum of transactions while m is the sum of items in a transaction. As for restructure the tree, it takes $O(nlog_2n)$. If the database is not so huge, this algorithm has a better performance over Apriori Algorithm. However, FP-growth Algorithm might have same or even worse performance when mega data need to be mined and analysed.

3 One Proposed Algorithm

On the top of particular algorithm, the principle of data analysis is always objectivity and Since data mining is to discover the underlying special rules among items without other noisy data or irrelevant constraints, we should adhere to “pure” Apriori fashion instead of “plausible” empirical fashion in data analysis. [7]

In this era of data explosion, better data analysis algorithm need to be designed to meet the buoyant demand. In order to the huge demand on data analysis, data analyser need revolutionary changes in computation capacity and storage capacity. We have pondered over a better algorithm for mining the association rules. Several cutting-in points, including better structure (like FP-Tree), better computer and better computational strategy, etc. We think Apriori Algorithm can be improved under the parallel computing to overcome the performance bottleneck. The first reason for the preference in Apriori Algorithm instead of other excellent Algorithms like FP-growth Algorithm is relatively easy implementation and its good applicability in parallel computing ecosystem. The second reason for favour on parallel computing over concurrent computing or serial computing is less waiting time, faster response and greater throughput. In this algorithm we propose, MapReduce [9] instead of Spark is chosen because of its inexpensiveness in hardware and labour cost and robust maturity in pattern and technicians. [10]

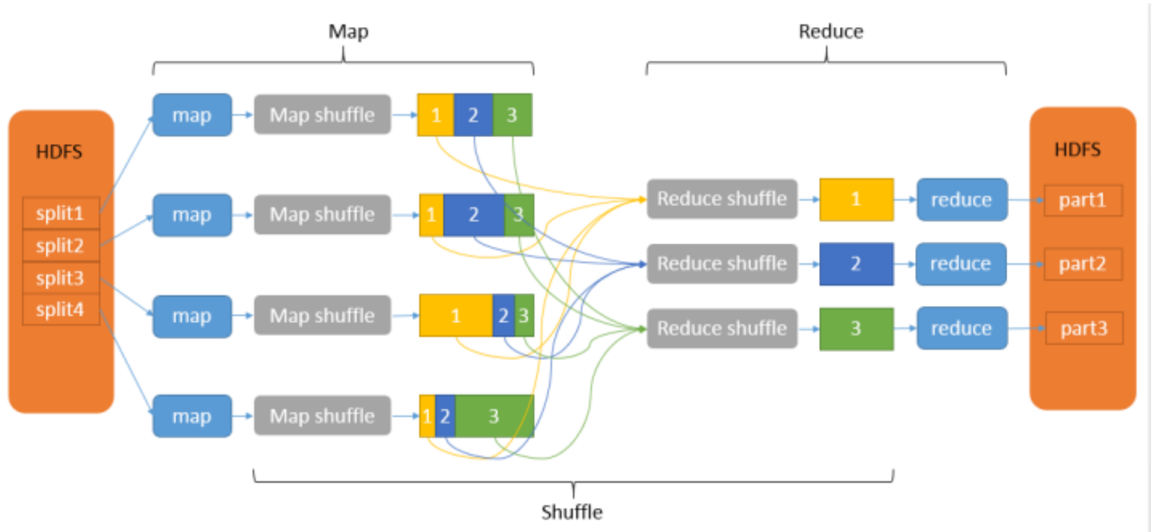

Figure 3. MapReduce Data Flow Diagram

3.1 Algorithm Theory

The basic idea of this algorithm is that, under the MapReduce model, original transaction item sets will be divided into several groups before actual data processing proceeds by Mappers. Then those groups will be allocated to computers in Hadoop Cluster. [11] Then 3 steps will be made to analyse those data groups:

- Generate 1-item candidate item sets by comparing original transaction item sets with item sets in each data group.
- Calculate the support of each item set. Then get the 1-item frequent item sets by comparing those support with Minimum Support.
- Generate the frequent item sets iteratively until the reaching maximum size of item sets.

3.2 Implementation

In this subsection, we will try best to explain how this algorithm works. We reckon that 5 steps should be made.

Step①: Divide Transaction Database into X subsets which similar size of items. Then send subsets to X computers. (Map)

Step②: As for each computer, the subset it received should be scanned to generate all local candidate item sets, marked as Candidate_1. Counter increment for support of each candidate item set is 1.

Step③: Send same item sets with corresponding supports in Candidate_1 from X computers to Y computers by using the hash functions. (Shuffle)

Step④: Sum up counter of all same item sets in Y computers. Then generate the total supports accordingly. Then determine the local frequent item sets, marked as FreqSet_1 by comparing actual supports with Minimum Support. (Shuffle)

Step⑤: Accumulate support counters of all same candidate item sets to update the actual support counters in Transaction Database. Then all frequent item sets can be secured. (Reduce)

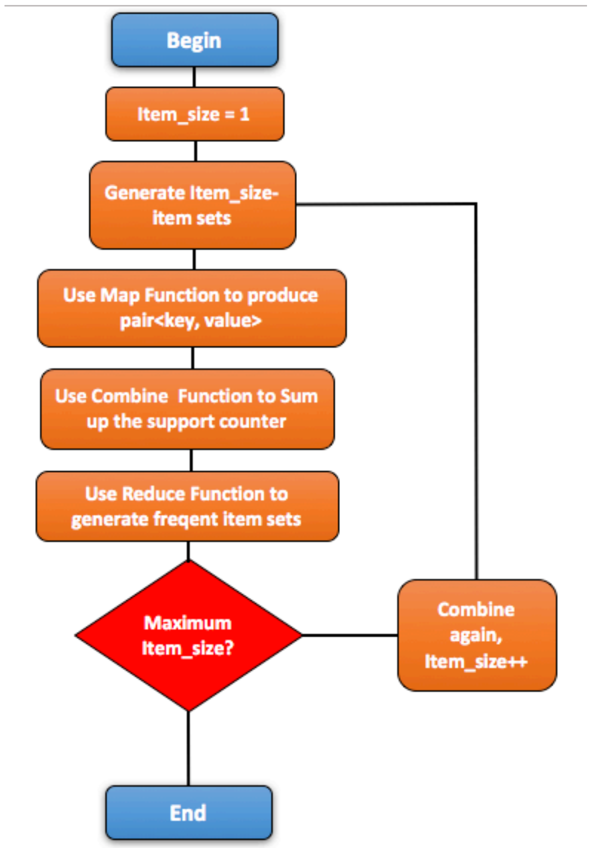
Begin
Item_size = 1
Generate Item_size-item sets
Use Map Function to produce pair<key, value>
Use Combine Function to Sum up the support counter
Use Reduce Function to generate freqent item sets
Maximum Item_size?
Combine again, Item_size++
End

Figure 4. Frequent Item Sets Generation by MapReduce

3.3 Algorithmic Thinking & Efficiency Analysis

❖ **Divide and Conquer** [9]

Undoubtedly, Divide and Conquer is employed in Apriori Algorithm with MapReduce. However, it does not simply minify item sets size when dividing the problem into small problems. The dependency (Support Counter) among item sets has been considered when making decisions on grouping different item sets. Item sets which are strongly dependent on each other tend to be marshalled to same group, whereas those item sets whose dependency are low are likely to be sent to different groups. Then item sets will be calculated in separate computers in different groups until local computation are finished. All local results will be gathered together to determine the final result in the end.

Distinctiveness in Divide and Conquer this algorithm uses is on the introduction of dependency among item sets. Credit should be given to this distinctive trick because of decrease in both cost and complexity.

❖ **Larger Data, Faster Time**

As one of prevalent algorithms in parallel computing, it is manifest that the advantageous performance when the mega data need to be analysed. MapReduce provides a right direction for data analysts to lessen limitation of Apriori in single-machine without expensive expense.

4 Future Work

Mining association rules with item constraint contains infinite values to be dug out. Although many studies have been conducted on this field, there are still enormous needs in different industries. After studying Distributed Databases and Data Mining, many practical methods and innovative ideas have enlightened us in data analysis. When we conducted survey and research, we get relatively in-depth comprehensions in many knowledges we studied in this course. We are attracted by data analysis when we get familiar with the techniques and importance of data analysis. In order to become a competitive data analyst, three work we should keep on:

- **Programming Ability.** Programming is always the priority task for use to commit to. Efficient algorithms can create value when we implement them in reality. We should continue the study on data mining. Source code are good materials for us to learn from. From the basic algorithms to advanced algorithms, we need to practice them until we digest them and get fully understanding.

- **Working Practice.** Working in team in company provides a good platform for us to apply our knowledge and techniques into real business to create value. This is also an ideal way for us to reinforce our ability and get insight in professional field. Since only those data analysts who have excellent programming ability, insightful ideas and professional knowledges can gain comparative advantage, which is also the cornerstone of our future career.

- **Lifelong Learning.** Continuous learning is the most important attitude we need to maintain. Techniques and technology develop rapidly. Therefore, we need to continue learning and study to update our knowledges and ideas. Open source ethos facilitates and encourages us to learn continually no matter we are at university or in the workplace.

5 Conclusion

The main purpose of this research is to try to find out one better algorithm which can effectively mining association rules with item constraint by calculating frequent item sets. In this research, Apriori Algorithm and FP-growth Algorithm have been introduced and analysed with specific case study. Main idea, implementation and analysis in algorithmic thinking and efficiency have been presented and illustrated. In conclusion, Apriori Algorithm is universal and easily-implemented but costly due to frequent scans and prolific candidate item sets. FP-growth Algorithm discovers a shortcut to avoid massive workload Apriori Algorithm has to undertake, which produces improved performance. However, the FP-Tree fail to maintain its high efficiency once data are pretty huge. In order to break the bottlenecks, we suggest a solution which combines Apriori Algorithm with MapReduce parallel computing. This method can overcome drawbacks Apriori Algorithm and FP-growth Algorithm have. Further study on this proposed algorithm need to be done since there are few study on this title.

6 Reference

[1] Ramakrishnan Srikant & Quoc Vu & Rakesh Agrawal. *Mining Association Rules with Item Constraints.*

[2] Rakesh Agrawal & Ramakrishnan Srikant. *Fast Algorithm for Mining Association Rules.*

[3] Rakesh Agrawal & Tomasz Imielinski* & Arun Swami. *Mining Association Rules Between Sets of Items in Large Databases.*

[4] Francesca Rossi & Peter van Beek & Toby Walsh. *Handbook of Constraint Programming. Chapter 4-Backtracking Search Algorithms.*

[5] Pang-Ning Tan & Michael Steinbach & Vipin Kumar. *Introduction to Data Mining. Chapter 6-Association Analysis: Basic Concepts and Algorithms*

[6] Jiawei Han & Jian Pei & Yiwen Yin. *Mining Frequent Patterns without Candidate Generation.*

[7] K.Prasanna Lakshmi & Dr.C.R.K.Reddy. *Compact Tree for Associative Classification of Data Stream Mining.*
[8] Immanuel Kant. *Critique of Pure Reason.*
[9] Jeffrey Dean & Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.*
[10] Juwei Shi & Yunjie Qiu & Umar Farooq Minhas & Limeri Jiao & Chen Wang & Berthold Reinwald & Fatma Özcan. *Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics.*
[11] Yue Zhao & YonggongRen & Yang Liu. *Improved Apriori Algorithm and Its Application Based on MapReduce.*