



Automatic web content organization based on user actions

by

Wenjie Li and Wenjuan Wang

Supervisor: Morten Goodwin Olsen

Project report for distributed systems in spring 2007

Agder University College
Faculty of Engineering and Science

Grimstad, 14 May 2007

Status: Final

Keywords: user actions, MyApriori algorithm, Fp-Growth algorithm, Apriori algorithm, association rules, running time

Abstract:

Rapid increase in data demands that we search a better way to accumulating useful information, when browsing searching engine. This web content organization is based on user experience. So we pay more attention to use association analysis in data mining and establish large distributed web portals between client and user. There are a variety of approaches of association rule, each one has its own merit and demerit. In order to find a new method of association with good quality in this project, we introduce MyApriori algorithm that is advanced on the basis of Apriori algorithm. Furthermore, by the comparison with FP-Growth algorithm, we also present the results of these two algorithms and give better suggestions on doing an automatic web content organization based on user actions.

Version Control

Version¹	Status²	Date³	Change⁴	Author⁵
0.1	Draft	2007-05-02	Testing and draw figures	Wenjuan Wang Wenjie Li
0.2	Draft	2007-05-08	Original creation of the report. Problem description, background, solution and Work plan	Wenjie Li Wenjuan Wang
0.3	Draft	2007-05-08	Problem description edited and solution improved	Wenjie Li Wenjuan Wang
0.4	Draft	2007-05-09	Discussion added and conclusion	Wenjie Li Wenjuan Wang
0.5	Draft	2007-05-11	Background edited	Wenjie Li Wenjuan Wang
0.6	Draft	2007-05-11	Abstract and introduction	Wenjie Li Wenjuan Wang
0.7	Draft	2007-05-13	Appendix and rework all report	Wenjie Li Wenjuan Wang
1.0	Final	2007-05-14		

1 **Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.

2 **Status** is DRAFT, REVIEW or FINAL

3 **Date** is given in ISO format: yyyy-mm-dd

4 **Change** describes the changes carried out since the previous version

5 **Author** is the one who did the change

Table of Contents

1.	Introduction	4
1.1	Acknowledgements	4
1.2	Report outline	4
2.	Background	5
2.1	Data mining	5
2.2	Association rules	5
2.2.1	Apriori algorithm	5
2.2.2	FP-Growth algorithm	6
2.2.3	Previous work	6
2.3	distributed database	6
3.	Problem description	8
3.1	Task description	8
3.2	Assumptive solution	8
3.2.1	Assumption	9
3.2.2	Simple solution	10
3.3	Main problem to solve	11
4.	Solution	13
4.1	How to design MyApriori	13
4.2	How to design FP growth	16
4.2.1	FP-tree	16
4.2.2	Mining in FP-tree	17
4.3	Analysing two algorithms	18
4.4	Validation and Testing	18
4.4.1	Test results	18
4.4.2	Result analysis	20
5.	Discussion	25
6.	Conclusion and further work	26
6.1	Conclusion	26
6.2	Further work	26
	Appendices	27
	Glossary & Abbreviations	27
	References	27

1. Introduction

Nowadays, a large number of information on the Internet makes our lives convenient, but we don't find what exactly we want to as soon as possible. In information world, our attention is paid to how to discover useful knowledge for us in a way of less time consuming. The related links in the website may help you save your time, for instance, users search a topic in wikipedia, and it will give you a lot of contents about this topic. Even there will list some related topics comparing original topic. In a word, this kind of website is very popular. In servers' aspect, good content will increase the flux of its websites. So our most important problem is the amount of time in the project. The task is to manage thousands of data or files in database and look for valuable transactions on user experience by analyzing high related transactions.

As distributed systems were invented, it helps us solve the problem of transferring a great deal of data and media on network. Distributed database and distributed file system both are able to slow the flux of one web portal. These technologies have been searched for several years and there are some companies that supplied good answers to manage distributed database and files, such as Andrew file system(AFS), common internet file system(CIFS) and so on. Therefore, continuing to discuss multiple web portal is not meaning.

We decide to focus on algorithm of association rules in data mining to enhance the performance of algorithm. By putting MyApriori and FP-growth together, we want to know which algorithm is good for generating frequent 2-itemset. In order to reach our goal, test result and conclusion will be give in following section.

1.1 Acknowledgements

Thanks to Morten Goodwin Olsen and Ole-Christoffer Granmo, they guide us in doing this project and give us some trusty suggestions. An enormous thanks goes to our friends for support from them.

1.2 Report outline

The report is organized in 6 sections. We introduce the related information about algorithm and multiple web portals in section 2; we present the problem description of our project in section 3. In section 4, we give the suggestion of this project and explain how to design two algorithms, in addition test results. Section 5 is about discussion based on above results. In the last section, the conclusion is drawn from the results of implement, further work is put forward.

2. Background

In this section, we will describe the background of our project and previous work done. We also look at the basic ideas and arithmetic that we have used in our project in the following content.

2.1 Data mining

Informally speaking, data mining is the process of extracting information or knowledge from a data set for the purpose of decision making[1]. With the rapid increase of large data sets, it becomes an active field in recent years. The technology of data mining is applied to scientific research, business performance management and many other fields.

Data mining supplies different algorithms according to different tasks. There are four of the core data mining tasks, respectively predictive modeling, cluster analysis, association analysis and anomaly detection. All algorithms attempt to fit a model to the data.[2] Model can be either predictive or descriptive in nature.

2.2 Association rules

Association analysis is used for discovering interesting relationships hidden in large data sets. Given a transaction based dataset, association rules specify patterns within that dataset, e.g. 75% of people who users in wikipedia who visit “the second world war” also visit “war ships”. We could consider “the second world war” has a link with “war ships”, this relationship can be expressed as {second world war} \rightarrow {war ships}.

In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k-itemset. Now, we will introduce support and confidence that are the strength of an association rule. X and Y are disjoint itemsets, i.e. $X \cap Y = \Phi$ and they have the relationship like this $X \rightarrow Y$, the equations are like:

$$\text{Support, } s(X \Rightarrow Y) = p(X \cup Y) = \frac{\sigma(X \cup Y)}{N} \quad (2.1)$$

$$\text{Confidence, } c(X \Rightarrow Y) = P(X/Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \quad (2.2)$$

Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in transactions that contain X.[3]

2.2.1 Apriori algorithm

Apriori is a classic algorithm for learning association rules. It is designed to operate on databases containing transactions(for example, collections of items bought by customers, or details of a website frequentation). This algorithm is the first association rule that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets.[4]

Apriori algorithm has two important characteristics. First, it is a level-wise algorithm; it means that it traverses the itemset lattice on level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets. Second, it employs a generate-and-test strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated

from the frequent itemsets found in the previous iteration. The support for each candidate is then counted and tested against the minisup threshold.

2.2.2 FP-Growth algorithm

FP-Growth takes a radically different approach to discovering frequent itemsets. The algorithm does not subscribe to the generate-and-test paradigm of Apriori. Instead, it encodes the data set using a compact data structure called an FP-tree and extracts frequent itemsets directly from this structure. [5]

FP-growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems. For example, suppose we are interested in finding all frequent itemsets ending in e. To do this, we must first check whether the itemset{e} itself is frequent. If it is frequent, we consider the subproblem of finding frequent itemsets ending in de, followed by ce, be, and ae. By merging the solutions obtained from the subproblems, all the frequent itemsets ending in e can be found. This divide-and-conquer approach is the key strategy employed by the FP-growth algorithm.[6]

2.2.3 Previous work

IBM Almaden research center discovered an efficient algorithm that generates all significant association rules between items in a large database of customer transactions. Each transaction consists of items purchased by a customer in a visit. This research introduces the problem of “mining” a large collection of basket data type transactions for association rules between sets of items with some minimum specified confidence, and presents an efficient algorithm for this purpose.

IBM almaden research center also discuss a hybrid algorithm called AprioriHybrid. This algorithm is combined by two proposed algorithms. By empirical evaluations, these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems. AprioriHybrid also has excellent scale-up properties with respect to the transaction size and the number of items in the database.[7]

Three professors, in university of Helsinki, looked for efficient algorithms for discovering association rules. They gave a method based on successive passes over the database. They gave an improved algorithm for the problem. The method is based on careful combinatorial analysis of the information obtained in previous passes; this makes it possible to eliminate unnecessary candidate rules.[8]

There are a lot of person researching advanced algorithms about association rules. In here, we just list above three papers as the example.

2.3 distributed database

A distributed database is a database that is under the control of a central database management system in which storage devices are not all attached to a common CPU. It may be stored in multiple computer located in the same physical location, or may be dispersed over a network of interconnected computers.

Collections of data (in a database) can be distributed across multiple physical locations. A distributed database is distributed into separate partitions. Each partition of a distributed database may be replicated. Care with a distributed database must be taken to ensure that the distribution and transactions are both transparent.

3. Problem description

In this section, the description of the task will be given. We will then give a assumptive solution and analyse out our main focus.

3.1 Task description

This task is to organizing the content of a large web portal. Some of the content is organized by administrators, and several parts of the content could be organized automatically based on users' actions. Our challenge is that the user actions must be distributed through several text streams, we should use techniques to detect which web pages should be linked together to enhance the user experience. Total task could be expressed as follow:

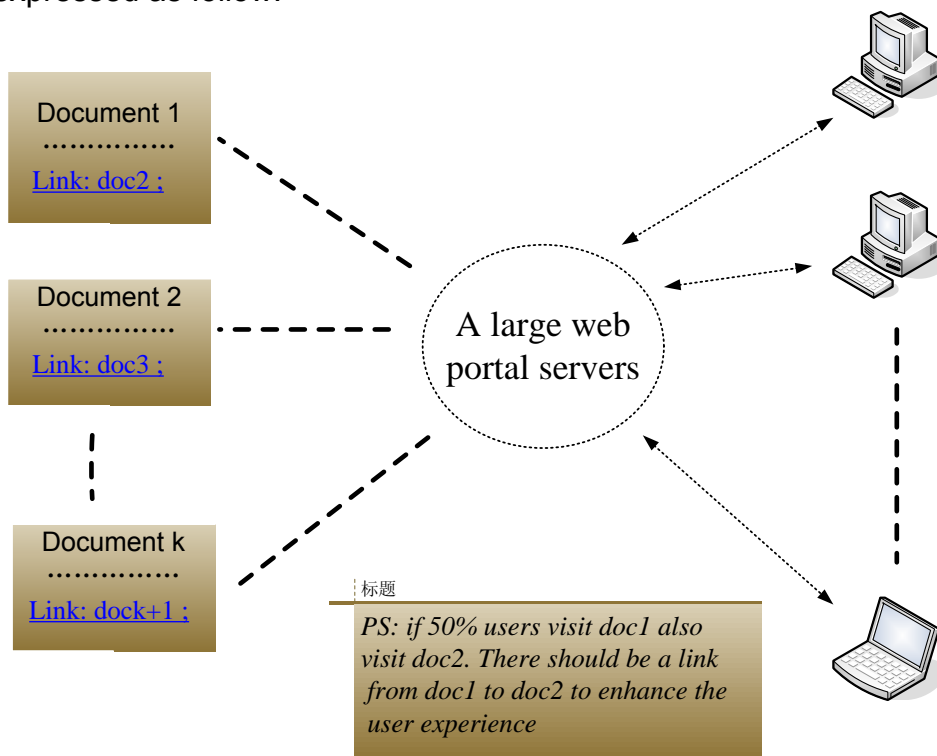


Figure3.1 problem description

Main problem: how to automatically generate content based on users' actions as soon as possible? The most important performance measurement of project is the amount of time needed to satisfy service requests. This problem can be divided into two parts, one is about large web portal servers, the other one is association rules in data mining.

For web portal, the time-consuming includes the time to deliver the request to a server, the time to deliver the response the client, and a CPU overhead of running the communication protocol software. On the other hand, we try to make association rule implement quickly to reach same purpose.

3.2 Assumptive solution

Assumptions of this task will be given in this section. Simple solutions to those given problems will be included too. We can focus on algorithms then.

3.2.1 Assumption

In the beginning of problem analysis, structure of website is mentioned. Since now web sites in most cases is not structured hierarchically. However, we assume that web structure is based on hierarchical document systems, because it is the often use. (e.g document system of www.wikipedia.org).

More details of the structure:

1. Hierarchical system. Documents are sorted already.
2. Each document has a unique identity.
3. Some obvious links already exist.

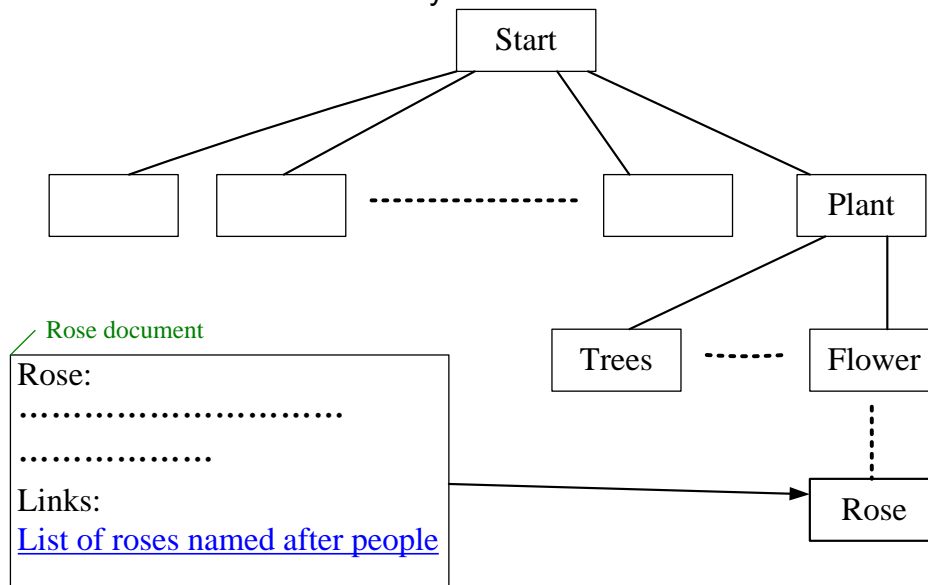


Figure3.2 Assumption of website structure

Now we should format the input and output. Follow the rules below:

1. Each document is an Item. Rose document= Rose. The name of the item is unique, and acts as an identity.
2. A set of user actions will be a Transaction. $T = (I_1, I_2, \dots, I_k)$.
3. A large number of Ts will be included in a txt file. That is the simple input.

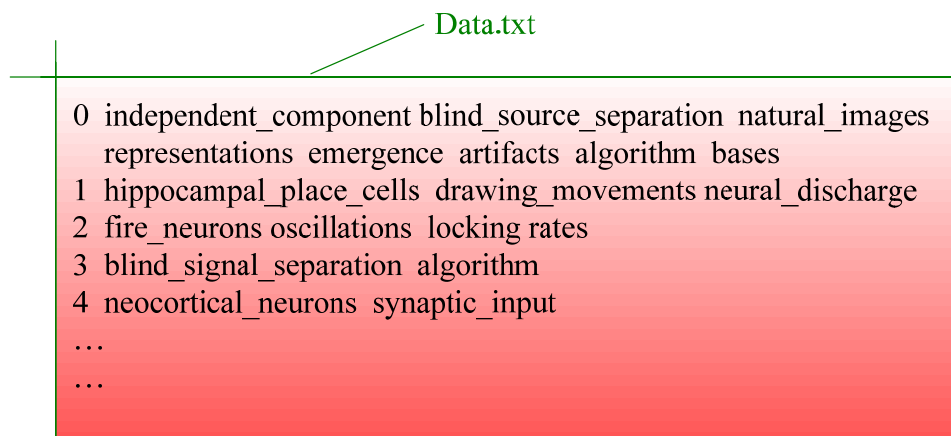


Figure3.3 Input format

4. Output is shows as rules like $I_1 \rightarrow I_2$. It means there should be a relative link to I_1 in document I_2 .

```
Data.txt
----频繁项集----
0:model-79
1:neurons-72
2:neurons-model-15
3:neural_networks-56
4:dynamics-53
5:dynamics-model-18
6:cortex-52
7:visual-cortex-41
8:networks-38
9:algorithm-35
10:responses-34
11:information-33
12:synchronization-24
13:independent_component_analysis-23
14:primary_visual-cortex-23
15:systems-22
16:striate_cortex-20
17:cells-19
18:classification-18
19:neocortical_pyramidal_neurons-18
20:cortical-neurons-18
runingtime is: 204ms
```

Figure 3.4 Output format

3.2.2 Simple solution

Several simple solutions are described to solve the unfocused problems.

1. How to collect user actions? How can we translate the user actions to simple input?
The log file can be used and pumping the actions from the log file would be easy.
2. How to collect the distributed actions?
For large websites, Log file may be different in every database. So we must collect files from all databases, and then combine them together. A figure will help us understand clearly.

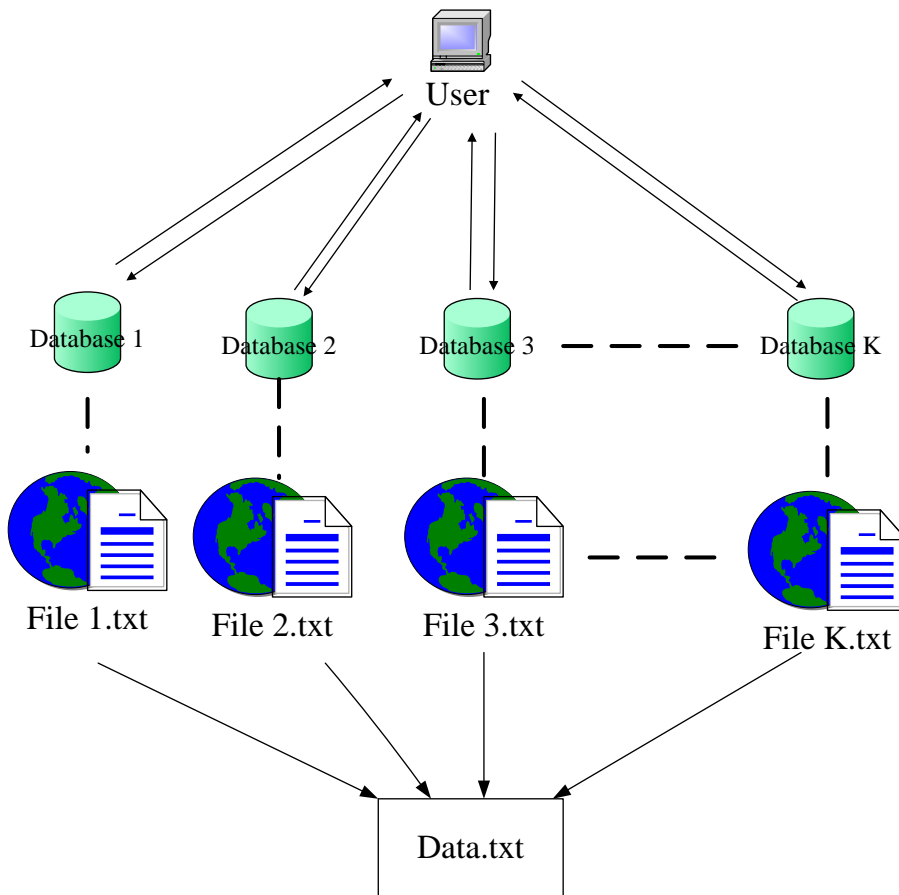


Figure 3.5 Combine inputs

3. Computing online versus offline?

For large servers, thousands of users may connect at same time. If we compute the actions immediately, it will greatly decrease the server performance. In this case, computation will be come later.

I Association rules are limited by mini-support and mini-confidence. Two concepts are mentioned in the background.

II Time of linking depends on the flux of the websites.

3.3 Main problem to solve

Generally, the structure of websites is a basic problem. Considering this, we divide the problem into three parts: Input, Output, and algorithm.

Input:

We collect the actions and automatically generate the relationship of those actions. In addition, a large website may have a lot of mirror sites and replicates of the database. It means user actions are distributed through the whole systems. So we also need to combine the actions. Outline of the problems comes below.

Problems:

1. How to collect user actions?
2. How can we translate the user actions to simple input?
3. What type of document shall we use for inputting? (.txt or something else)
4. How to collect the distributed actions?
5. When we collect the input, should we computing is immediate or do it later?

Output:

The output is the inter relationship of documents. It works as those relative links in the document. Hence, the relation is between two documents. After our computations, we generate the relations and update to the database. What should we follow when we generate the output?

1. What is the limitation of the rule?
2. Why we using those limitations?
3. How long should we keep the output? In another words, how long did the links exist?

Algorithm:

There are a lot of algorithms can be used in this task, we will choose two most main algorithms used widely, respectively MyApriori and FP-growth. MyApriori is adjusted by us, on base of Apriori. Each algorithm has its own merit, hence we want to know which algorithm is suitable in different situations. According to our search, give our suggestions.

4. Solution

In this section, we will describe the algorithms as a part of the solution. There are two parts, one is Apriori and the other is FP growth.

First of all, the main step of the task would be described.

Step 1: Collect and formalize the input.

Step 2: Pick out the frequent_2_itemset.

Step 3: Rule generation.

Step 4: Update to database.

It is evident that methods using in step 1, 3, 4 are very simple. If the input and output is the same, the time cost in step 1, 3, 4 is almost invariableness. We focus on step 2. The improvement in step 2 will greatly increase the performance.

Data structure see appendix.

4.1 How to design MyApriori

A short description is given to illustrate the original Apriori algorithm.

Step 1: Scan the input to get the support S of each 1-items.

Compare the S with min_support , prune the infrequent items, and then generate frequent_1_itemset IS_1 .

Step 2: Generate candidate_k_itemset using IS_{k-1} .

Prune the candidate C_i if C_i has a subset CS_{sub} that don't belong to IS_{k-1} . Find the final candidate_k_itemset.

Step 3: Scan the input, count the support S of C_i .

If C_i belongs to T_k , count ++

Compare S with min_support , prune the infrequent ones, and then get the frequent_k_itemset.

Step 4: If the frequent_k_itemset $CS_k \neq \text{null}$.

Do step 2 again, $k=k+1$. [9]

Changes:

1. To fix our task, we adjust the steps in order to optimize the algorithm.

At the beginning, the output of this algorithm is all frequent itemset. But in our analyzing, only frequent 2_itemsets are interested.

Improvement:

1. Every time, it counts for item of C_i , it should scan the whole input once, if the input contain 10000 transactions, it's a waste of time to do this. So we try to avoid this. We use a bintree here to reduce the time of item and itemset.

```

Scan the input, and building the tree.
While (Ti! =null)
    Kw[]=Ti(translate Ti to a array of kws);
    Search the kw in bintree, ;
    If (kw is already exist)
        Update count S, record the TID;
    Else
        Init a node (kw);
        Count S=1, record TID;

```

Table4.1 building bintree

E.g.

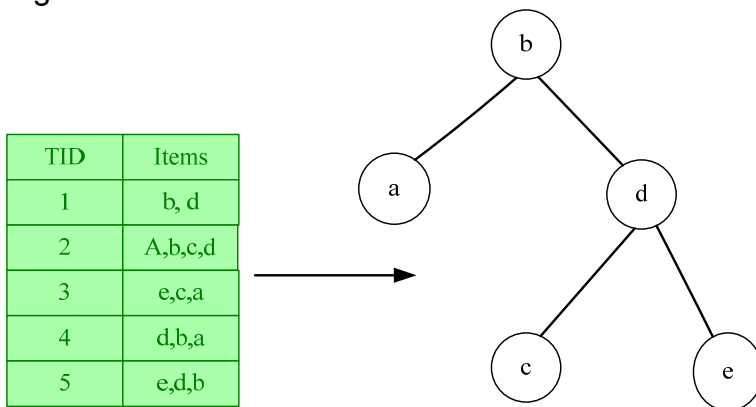


Figure 4.1 example of building bintree

With the figure we can illustrate it easily.

Scan the first transaction T_1 , we start build the tree. Item b is not exist, so we initial the node, $kw=b$, $count=1$ $index="1"$ (index are used to record the TID). $d>b$, we go right child. There isn't a node. So build node d too. Then follow the rules, we build node a as a left child of b. next item in the T_2 is b, and b is already there. Just plus 1 to the count $index="1; 2"$. Then initial node c, and update node d, and so on .after finish scan, we got the bintree.

Comments:

- ① It greatly decrease the time cost of search in database. Search in normal database cost $O[n]$, but in a bintree, the time is $O[\log_2 n]$
- ② for space complexity, the tree will store in memory. So cost of the space is increased.

2. Compare the min-support and prune the infrequent node, then rebuild final tree.

In this case, search in the bintree will take less time. Now go step 3 again. Because of the bintree, we cant using the original way to count the C_i . While the bintree is building, the kw and its' TID is also recorded. So solution comes blew.

```

Kw[] = Ci.split ;           //split Ci to an array of kws);
nIndex[] = bintree.getkw( kw_n ); //return its TIDs;
Compare nIndex[];         // n=0, 1, 2.....n;
Start with 1Index [0], 2Index [0]...nIndex[0]
Do:
  If (all equal)
    Count++;
    all index go the next record.
  Else
    Find the minimal and it goes the next record;
Until one of index[] reach the end;
Return count.

```

Table 4.2 Count key

1Index	Kw1-TIDs	2Index	Kw2-TIDs	3Index	Kw3-TIDs
1Index [0]	1	2Index [0]	1	3Index [0]	1
1Index [1]	3	2Index [1]	2	3Index [1]	4
1Index [2]	4	2Index [2]	4	3Index [2]	5
1Index [3]	6	2Index [3]	5	3Index [3]	7
1Index [4]	7	2Index [4]	7	3Index [4]	9

Table 4.3 Example of count key

Follow this Table, it will show how the improvement is going on. Firstly, we split C_i ; get three items:kw1, kw2, kw3. Second, find the TIDs and split them to nIndex[]. The Table shows the kws and its TIDs. Then begin to compare. 1Index [0], 2Index [0], 3Index [0]. 1=1=1 equal. Count of C_i plus 1.Next step all Indexes go its next record. 1Index [1] =3, 2Index [1] =2, 3Index [1] =4. Not equal, 2 is the minimal. 2Index go its next record. 3,4,4, still not equal. 1Index go next. 4, 4, 4 equal. Count pluses 1. And so on.

Comments:

① It reduces the counting time from $O[n^m]$ to $O[nm]$.(n=number of kws, m= average length of index)

Now we have an overview of Apriori algorithm. As a sum-up a flow chart will present it more clearly

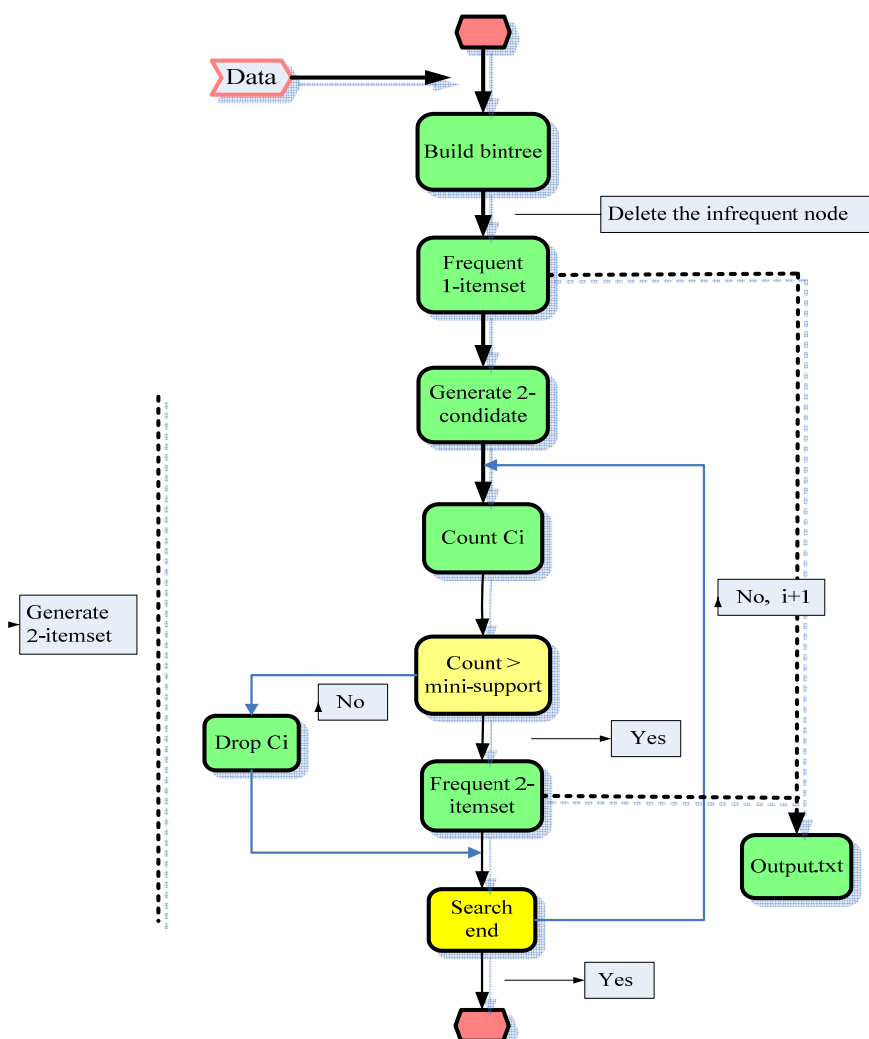


Figure 4.2 Actual Apriori algorithm

Source code see appendix

4.2 How to design FP growth

This section presents another algorithm called FP-growth. It takes a different way to find the frequent k-itemset. It avoid the cost while generate the candidates. Instead, it encodes the input using a compact data structure called an FP-tree. Then pick out the frequent k-itemset from the structure directly. [10]

4.2.1 FP-tree

FP-Tree is the compressed representation of the input. It farthest reserves the composing of transaction, so frequent itemset can be extracted directly from the trees. It assumes that the items are already sorted by support. It construct by reading the input, mapping the transactions to a branch of the tree, merging the branch if foreside is the same. For instance, path a-b-d-e and a-b-e-f, they have same foreside a-b; we can merge this two to one branch, a-b- and have a right branch e-f and a left branch d-e. After scan the whole input data, FP-tree is built.

The main steps present blew:

1. Scan the input data once; collect the frequent items and count support. Sorted the in decreasing support counts in itemset L.
2. Initially, the tree contains only the root node, and present by **null** symbol.
3. for each transaction T_i .

```

Prune the infrequent item, and sorted the items follow the sequence in L;
Define p is the first element  $T_i$ ,  $P=T_i-p$ ;
  Insert (p, P, Tree);
    If T has children n that satisfies  $n.item-name==p.item-name$ 
      N.count++;
      p= first item of P, and  $P= P-p$ ;
      If p! = null
        Insert (p, P, n);
    Else
      Initial a new child node N,
        N.item-name=p;
        Count=1;
      Initial a link to the node n that  $n.item-name=p$ ;
      p= first item of P, and  $P= P-p$ ;
      If p! =null;
        Insert (p, P, n);

```

Table 4.4 Building FP-tree

4. Finish.

Here is the figure to present the final FP-tree.

More examples will present in a Table.

Item	Prefix paths	Conditional fp-tree	Frequent itemsets
I5	{{(I2 I1:1),(I2 I1 I3:1)}}	<I2:2,I1:2>	I2 I5:2,I1 I5:2,I2 I1I5:2
I4	{{(I2 I1:1),(I2:1)}}	<I2:2>	I2 I4:2
I3	{{(I2 I1:1),(I2:2),(I1:2)}}	<I2:4,I2:2> <I1:2>	I2 I3:4,I1 I3:4,I2 I1 I3:2
I1	{{(I2:4)}}	<I2:4>	I2 I1:4

Table 4.5 Example of fp-growth in items

Source code see appendix

4.3 Analysing two algorithms

In this section, we discuss the time complexity and space complexity of these two algorithms. Furthermore, some comments in theoretic aspect are included.

Time complexity:

There is a bottleneck in MyApriori algorithm. In most situations, the prune strategy using in MyApriori algorithm greatly decrease the cost of candidates' generation, and it already works very well. But we still can't ignore the cost, for instance, the number of frequent items is over 10,000, it will generate 10^7 candidates, count and check for all of them. When the Itemsets grow bigger, the cost will be unthinkable.

For FP-growth, it avoids candidates' generation. The running time of algorithm its depend on the support. In the author points out. When is support $>1.5\%$, FP-growth is fast then MyApriori. If the support $<1.5\%$, the running times are almost the same.[a]

Space complexity:

In MyApriori the size of bintree depend on how many items there are. The cost is $O[n]$. But the FP-tree is based on whether the data is convergence. If the data is a really dispersive data, the FP-tree may not be able to store in the memory. For different real task, the applications of these two algorithms are different too. Which one is fit for our task, more discussion about the detail is given after testing.

4.4 Validation and Testing

4.4.1 Test results

The whole testing is to search an efficient algorithm that could generate frequent 2 itemsets quickly. Running time and the number of frequent 2 itemsets are both our comparative parameters. The reason of choosing 2 itemsets as the test factor is that we just need to find great association from one item to another item. However, more 2 itemsets and 2 itemsets with low association don't relate with our project. On the other hand, according to our project's task, user actions have different possibility, so we divided the users' information into two groups, one is a dispersive group, the other one is a concentrated group.

Firstly, let's look three different dispersive data:

Mini-support	2	4	5	7	8	9	10	13	14	15
MyApriori's Running time (ms)	985	344	250	141	125	109	94	78	78	79
Frequent 2 itemsets in MyApriori	854	294	194	84	69	58	49	31	27	22
FP-Growth's Running time (ms)	328	281	250	203	203	188	187	203	203	204
Frequent 2 itemsets in FP-growth	907	230	220	119	89	75	63	37	32	29

Table 4.6: comparison of running time and number of frequent 2-itemset of two algorithms when 570 transactions are read.

Mini-support	20	30	50	70	100	200	300	400	500	700
MyApriori's Running time (ms)	17390	16547	8863	5438	3234	1344	1047	984	984	938
FpGrowth's Running time (ms)	2234	2188	2063	2063	2078	2047	2031	2047	2031	2031

Table 4.7: comparison of running time and number of frequent 2-itemset of two algorithms when 10000 transactions are read.

Mini-support	20	40	50	70	100	150	300	600	800	1000
MyApriori's Running time (ms)	141047	37688	36562	22688	20234	10063	5078	3422	3256	3188
FpGrowth's Running time (ms)	5609	4188	4016	4141	3953	3938	3906	3890	3890	3907

Table 4.8: comparison of running time and number of frequent 2-itemset of two algorithms when 20000 transactions are read.

Then comparing the three concentrated data:

Mini-support	2	4	5	7	10	30	50	100	200	400
MyApriori's Running time (ms)	125	110	125	109	109	125	109	125	125	47
FpGrowth's Running time (ms)	281	235	210	204	172	141	125	110	78	47

Table 4.9: comparison of running time and number of frequent 2-itemset of two algorithms

when 1000 transactions are read.

Transaction number	10	20	50	70	100	150	200	300	400	500
MyApriori's Running time (ms)	3000	2937	2875	2859	2907	2844	2437	2391	2391	2406
FpGrowth's Running time (ms)	594	563	483	469	469	438	391	375	313	318

Table 4.10: comparison of running time and number of frequent 2-itemset of two algorithms when 5000 transactions are read.

Mini-support	1	10	50	100	300	500	1000	2000	2500	3000
MyApriori's Running time (ms)	12094	12047	12438	12641	12344	11813	11703	12000	12265	12407
FpGrowth's Running time (ms)	875	828	735	703	625	563	469	360	359	344

Table 4.11: comparison of running time and number of frequent 2-itemset of two algorithms when 10000 transactions are read.

4.4.2 Result analysis

According to the tables above, we infer to some conclusions as follows:

1. These three figures indicate that running time of FP-growth algorithm presents stability in dispersive database, even if mini-support varies. However, that of MyApriori algorithm decreases greatly as mini-support increases. On reaching a certain mini-support, MyApriori shows better performance than FP-growth.

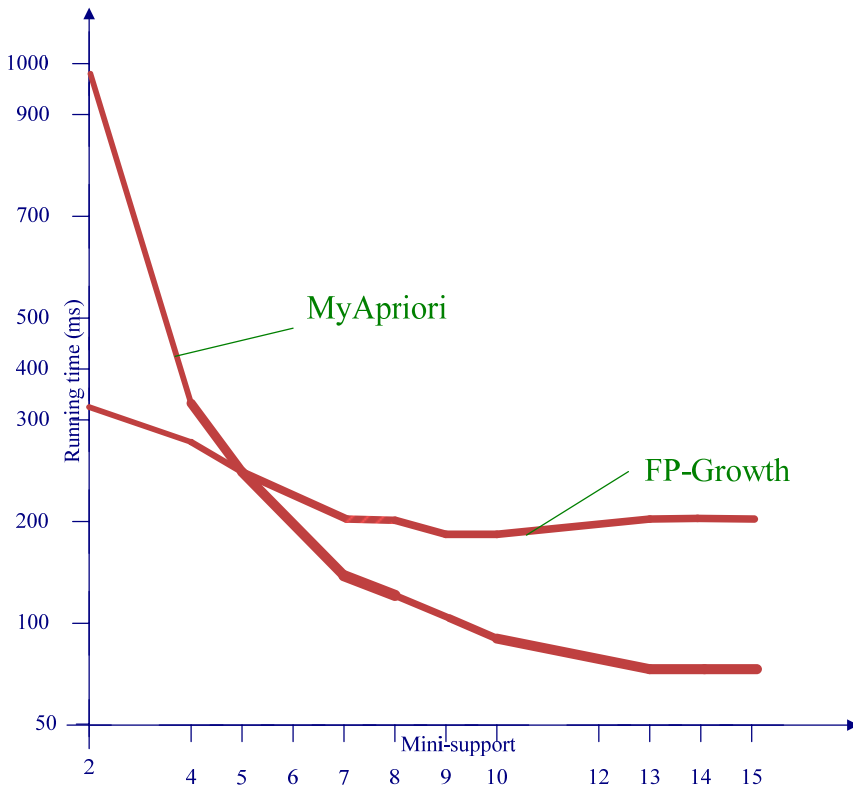


Figure4.5: 570 dispersive data

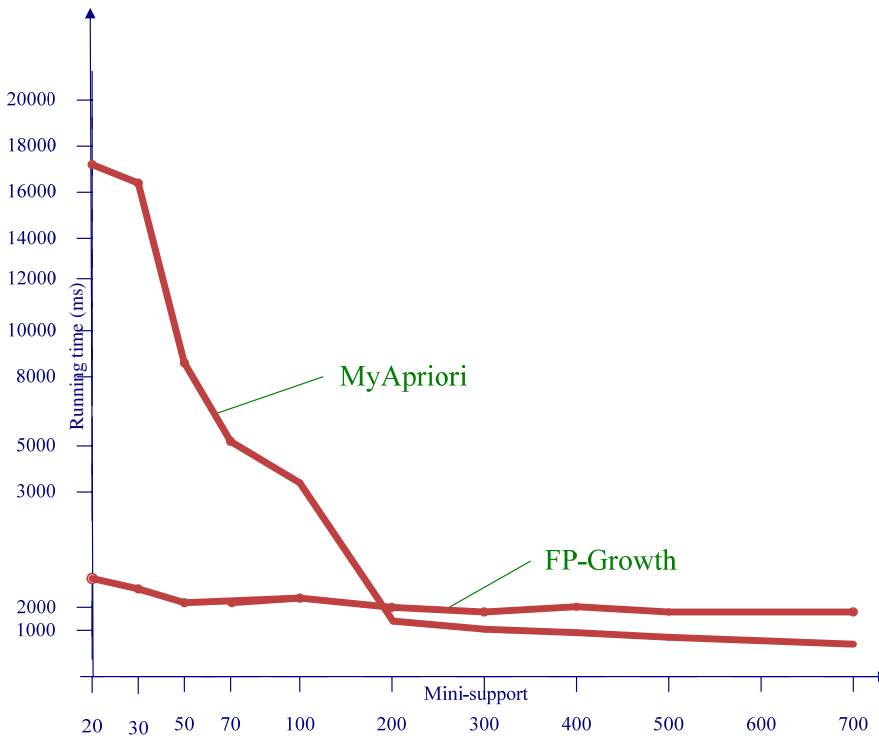


Figure 4.6: 10000 dispersive data

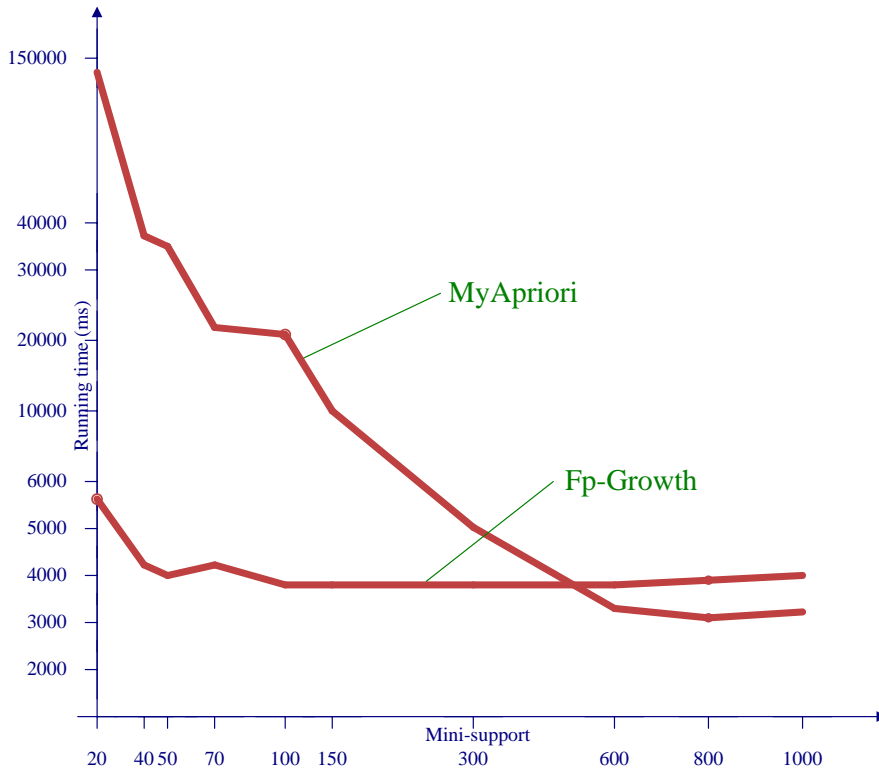


Figure 4.7: 20000 dispersive data

- When data is concentrated and has repeated transactions, running time of FP-growth is lower than MyApriori's. By the increase of number of data, the effect that FP-growth runs less time than MyApriori is more obvious.

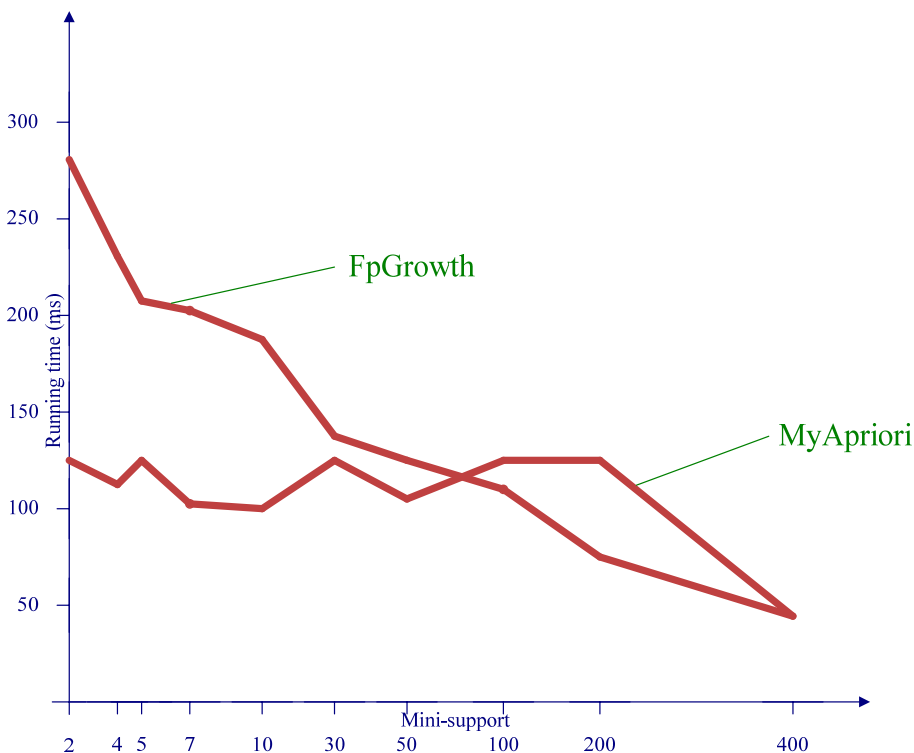


Figure 4.8: 1000 concentrated data

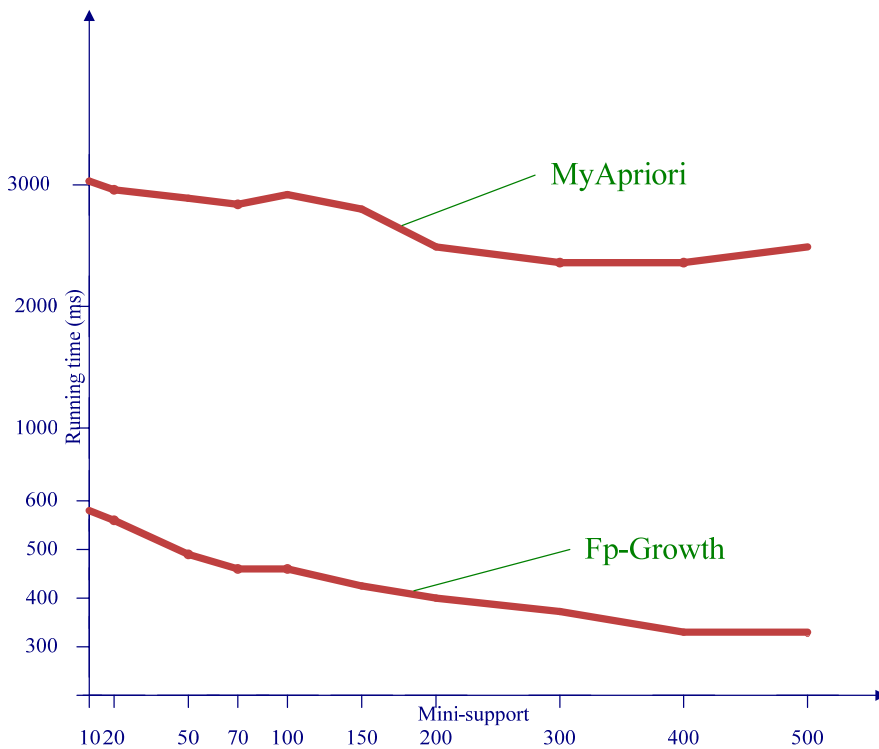


Figure 4.9: 5000 concentrated data

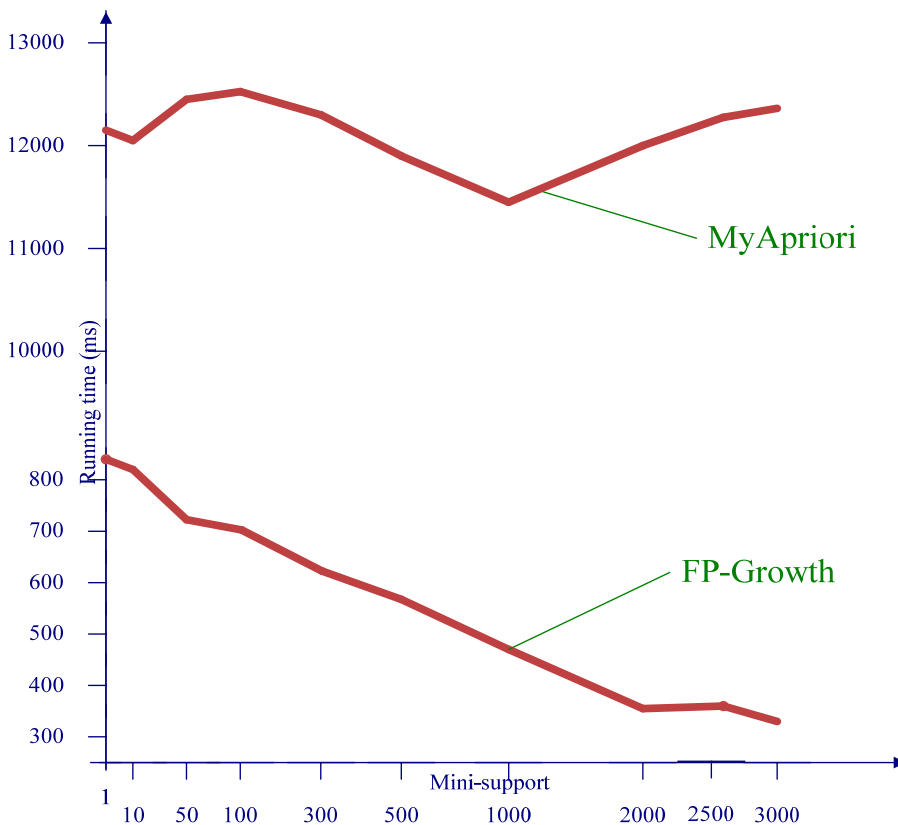


Figure 4.10: 10000 concentrated data

3. Through comparing two kinds of data, consume of concentrated data is lower than that of dispersive data in two algorithm. The performance of FP-growth display superiority over MyApriori.

4. By testing the number of frequent 2-itemset, we found that performance mining the number of frequent 2-itemsets of two algorithms is almost same.

5. Discussion

First of all, we advanced the original Apriori algorithm, that is means decrease the time of scanning the input when generating candidate k itemsets every time. Instead, through building a tree, we could count the support of candidate itemset. It improves the efficiency of all Apriori algorithm. Actually, we have changed algorithm successfully and named it as MyApriori algorithm.

However, later we found out that it didn't illuminate advantage of MyApriori algorithm. Consequently, we decided to compare this method with the method that is already known, FP-growth algorithm. At last, we wrote the codes of two algorithms, and selected two kinds of data as our variables. In fact, the result that we get proved is that two algorithms have their own advantages and disadvantages.

When input data is concentrated, FP-growth algorithm presents better performance. This testing also proved that FP-growth outperforms the standard MyApriori algorithm by several orders of magnitude for certain transaction data sets. I think that could explain why a gap between FP-growth and MyApriori in above figures. When data is concentrated, FP-tree has smaller branches. It also effect whether conditional FP-tree is bushy. So the run-time performance of FP-growth depends on the compaction factor of data set. If the resulting conditional FP-trees are very bushy, the performance of the algorithm degrades significantly.[7]

Nevertheless, the performance of MyApriori shows better, when dispersive data is input and mini-support reaches a certain value. Furthermore, as the mini-support is large enough, the advantages will be more obvious. In total, although FP-growth performs steadily however data and mini-support vary greatly, we find that MyApriori arrive an ideal result by improving on original Apriori algorithm.

6. Conclusion and further work

6.1 Conclusion

In this report, we are supposed that MyApriori improved has more advantages than FP-growth, because advanced Apriori algorithm builds bin tree when scanning database at first time, instead initial method to scan several times. It decreases time consuming greatly.

However, comparing the results, FP-growth has an effective performance in time in whole, when data is concentrated and dispersive. The better performance of MyApriori just happened in space complexity. It doesn't matter that The actual result don't reach our imagination well. We could realize essential distinction of two algorithms by discussing. In my opinion, it is not necessary to compare these two algorithms.

But by analysing above results, we found that is Myapriori has a better performance in space complexity. Furthermore, for our this project, transactions in large web portal are dispersive, this algorithm is favorable. On the other hand, the unstable cost of space will decrease the server's performances, even overflow in running FP-growth at the same precondition.

6.2 Further work

Concerning the further work related to algorithm, it is very important to find a better method that is less time-consuming and ideal frequent itemsets. Ideal frequent itemsets is means that we could not only mine arbitrary frequent itemsets but also find all frequent n-itemset. On the other hand, the code about FP-growth just runs all frequent itemsets, we wish that this code could implement frequent n-itemset automatically. That is to say FP-growth is a recursive process, if all recursion happen, we have all frequent itemset. But we control the time of recursion, e.g. twice, then frequent 2-itemset is took.

To implement large web portal is also challengeable. In the part of solution, we assume that web sites is structured hierarchically. It is different from most cases, so it could be considered as the further work.

Appendices

Glossary & Abbreviations

Support	See section 2.3 association analysis
confidence	See section 2.3 association analysis
Min-sup	Minimal support limiting
Frequent k itemset	Frequent itemsets, every element has k items.
Sup in F4.4	support
Lin in F4.4	Head ink brother nodes with the same item name.
Fp-growth	See section 4.2
Apriori	See section 4.1

References

- [1] U.Fayyad and R.Uthurusamy. **Data mining and knowledge discovery in databases**. Communication of the ACM, 39(11): 24-26, 1996
- [2] Margaret H.Dunham. Data Mining introductory and advanced topics. 4: 17-18.
- [3] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. **Introduction to Data mining**. Chapter 6, 243.
- [4] http://en.wikipedia.org/wiki/Apriori_algorithm
- [5] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. **Introduction to Data mining**. Chapter 6, 268.
- [6] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. **Introduction to Data mining**. Chapter 6, 271.
- [7] Rakesh Agrawal, Ramakrishnan Srikant. **Fast Algorithms for Mining Association Rules**. Introduction.
- [8] Heikki Mannila, Hannu Toivonen and A. Inkeri Verkamo. **Efficient algorithms for discovering association rules**. Abstract and introduction.
- [9] Lan H. Witten & Eibe Frank. **Data mining practical machine learning tools and techniques**. Chapter 6.
- [10] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. **Introduction to Data mining**. Chapter 6
- [11] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. **Introduction to Data mining**. Chapter 6, 273.
- [12] Amitabha Das, Wee-keong Ng, Yew-kwong Woon, "association rule mining:Rapid association rule mining" proceedings of the tenth international conference on information and knowledge management October 2001.
- [13] <association rules:apriori algorithm> andrew kusiak, university of iowa. andrew-kusiak@iowa.edu
- [14] <apriori implementation with C++> wentrue wentrue@sina.com

- [15] <comparison and application of fp_growth and Apriori algorithm> Yang wei, Geng liming, yang wei liaoning technical university.
- [16] <an efficient incremental mining algorithm-QPD> Jen-peng huang, Nan-jie huang, huang-cheng Kuo. department fo information management southern Taiwan university of Technology.
- [17] <an effcient algorithm for continoursly mining of assosiation rules> lv yonghe, Xu yaqi Taiwan university of technology.
- [18] <http://bbs.matwav.com>
- [19] <Mining Strong Affinity Association Patterns in Data Set with skewed Support Distribution> huixiong, pang-Ning Tan, Vipin Kumar