Performance study of Association Rule Mining Algorithms for Dyeing Processing System

Saravanan .M.S

Assistant Professor in the Dept. of I.T in Vel Tech Dr. RR & Dr. SR Technical University, Chennai, INDIA.

Research Scholar in R & D Centre, Bharathiar University, Coimbatore, Tamil Nadu, INDIA.

Email: saranenadu@yahoo.co.in

Rama Sree .R.J

Professor & Head in the Department of Computer Science, Rashtriya Sanskrit Vidyapeetha,

Tirupati, Andhra Pradesh, INDIA.

Email: rjramasree@yahoo.com

Abstract

In data mining, association rule mining is a popular and well researched area for discovering interesting relations between variables in large databases. In this paper, we compare the performance of association rule mining algorithms, which describes the different issues of mining process. A distinct feature of these algorithms is that it has a very limited and precisely predictable main memory cost and runs very quickly in memory-based settings. Moreover, it can be scaled up to very large databases using database partitioning. When the data set becomes dense, (conditional) FP-trees can be constructed dynamically as part of the mining process. These association rule mining algorithms were implemented using Weka Library with Java language. The database used in the development of processes contains series of transactions or event logs belonging to a dyeing unit. This paper contributes to analyze the coloring process of dyeing unit using association rule mining algorithms using frequent patterns. These frequent patterns have a confidence for different treatments of the dyeing process. These confidences help the dyeing unit expert called dyer to predict better combination or association of treatments. Therefore, this article also proposes to implement association rule mining algorithms to the dyeing process of dyeing unit, which may have a major impact on the coloring process of dyeing industry to process their colors effectively without any dyeing problems, such as shading, dark spots on the colored yarn and etc. This article shows that LinkRuleMiner (LRM) has an excellent performance for various kinds of data to create frequent patterns, outperforms currently available algorithms in dyeing processing systems, and is highly scalable to mining large databases. This paper shows that HMine and LRM has an excellent performance for various kinds of data, outperforms currently available algorithms in different settings, and is highly scalable to mining large databases. These studies have major impact on the future development of efficient and scalable data mining methods.

Keywords: Performance, predictable, main memory, large databases, partitioning, Weka Library

1. Introduction

The purpose of the association rules is to find correlations between the different processes of any application. Knowing the associations between these processes, it helps to take decisions and to use the process methods effectively.

The various association rule mining algorithms were used to different applications to determine interesting frequent patterns. One of the association rule mining algorithm such as Apriori algorithm used the property of support and confidence to generate frequent patterns. Another measure is Predictive Accuracy (Agrawal 1994) it is an indicator of a rule's accuracy in future over unseen data. Confidence of a rule is the ratio of the correct predictions over all records for which a prediction is made but it is measured with respect to the database that is used for training. This confidence on the training data is only an estimate of the rule's accuracy in the future, and since the user searches the space of association rules to maximize the confidence, the estimate is optimistically biased (Scheffer 2001). Thus, the measure predictive accuracy is introduced. It gives for an association rule its probability of a correct prediction (Srikant 1999) with respect to the process underlying the database.

The Weka Library software is used to generate frequent patterns, which has a sequence of association rules. Hence, Weka application can be used to design several processes which generate frequent item sets.

The dyeing processing system is called coloring process of cotton yarn. The coloring process has various treatments, each treatment involved different sub process. Therefore the association rule mining algorithms were applied in the domain of the dyeing process to predict better treatments to produce perfect shade of the color. A shade is a color which should match with the original color.

In this paper the section 2 deals about the related works of this paper. The importance of association rule mining algorithms such as Apriori, Frequent Pattern (FP) Growth and Hyper structure Mining (HMine) and LinkRuleMiner (LRM) algorithms were analyzed and discussed in section 3. The section 4 described the performance analysis of these association rule mining algorithms. The last section 5 concludes the paper with the brief note.

2. Related Work

In many cases, the Apriori algorithm significantly reduces the size of candidate sets using the Apriori principle. However, it can suffer from two-nontrivial costs: (1) generating a huge number of candidate sets, and (2) repeatedly scanning the database and checking the candidates by pattern matching. (Han 2000) devised an FPGrowth method that mines the complete set of frequent itemsets without candidate generation.

FPGrowth works in a *divide-and-conquer* way. The first scan of the database derives a list of frequent items in which items are ordered by frequency in descending order (Herbst 2000). According to the frequency descending list, the database is compressed into a frequent-pattern tree, or *FP-tree*, which retains the itemset association information. The FP-tree is mined by starting from each frequent length-1 pattern (as an initial suffix pattern), constructing its *conditional pattern base* (a "sub database", which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. The FPGrowth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity.

In the next section the performance studies demonstrate that the method substantially reduces search time. There are many alternatives and extensions to the FPGrowth approach, including depth-first generation of frequent itemsets by Agarwal (Agarwal 2001); H-Mine, by Pei (Pei 2001) which explores a hyper-structure mining of frequent patterns.

The FPGrowth method will never need to build up matrices and compute 2-itemset frequency since it avoids the generation of any candidate k-itemsets for any k by applying a pattern growth method. Pattern growth can be viewed as successive computation of frequent 1-itemset (of the database and conditional pattern bases) and assembling them into longer patterns. Since computing frequent 1-itemsets is much less expensive than computing frequent 2-itemsets, the cost is substantially reduced.

The FPGrowth method constructs FP-tree which is a highly compact form of transaction database. Thus both the size and the cost of computation of conditional pattern bases, which corresponds roughly to the compact form of projected transaction databases, are substantially reduced.

Hence, FPGrowth mines frequent itemsets by (1) constructing highly compact FPtrees which share numerous "projected" transactions and hide (or carry) numerous frequent patterns, and (2) applying progressive pattern growth of frequent 1-itemsets which avoids the generation of any potential combinations of candidate itemsets implicitly or explicitly, whereas Apriori must generate more number of candidate itemsets for each projected database. Therefore, FPGrowth is more efficient and more scalable than Apriori, especially when the number of frequent itemsets becomes really large. These observations and analyses are well supported by our experiments reported in this section.

The a major distinction of H-mine from other algorithms is that H-mine readjusts the links when mining different “projected” databases and has very small space overhead, even counting temporary working space; whereas candidate generation-and test has to generate and test a large number of candidate itemsets, and FPGrowth has to generate a good number of conditional (projected) databases and FP-trees.

The structure and space preserving philosophy of H-mine promotes the sharing of the existing structures in mining, reduces the cost of copying a large amount of data and building new data structures on such data, and reduces the cost of updating and checking such data structures as well.

The H-mine absorbs the nice features of FPGrowth. It is essentially a frequent pattern growth approach since it partitions its search space according to both patterns and data based on a divide-and-conquer methodology, without generating and testing candidate patterns. However, unlike FPGrowth, H-mine does not create any physical projected databases or constructing conditional (local) FP-trees. Instead, it builds and adjusts links dynamically among frequent items during mining to achieve the same effect as construction of physical projected databases. It avoids paying the cost of space and time for the (projected) database reconstruction, and thus has better performance than FPGrowth.

The H-mine is not confined itself to H-struct only. Instead, it watches carefully the changes of data characteristics during mining and dynamically switches its data structure from H-struct to FP-tree and its mining algorithm from mining on H-struct to FPGrowth when the data set becomes dense and the number of frequent items becomes small. This absorbs the benefits of FPGrowth which explores data compression and prefix-path shared mining. Since mining on H-struct and mining on FP-tree are built based on the same frequent-pattern growth methodology, such a dynamic algorithm swapping can be performed naturally and easily.

We have done several experiments to assess the performance of Association Rule Mining algorithms. First it is compared the performance of H-mine algorithm with the LinkRuleMiner algorithm and shows that LinkRuleMiner does not suffer any degradation in its performance. Then various experiments were conducted on it and show that LinkRuleMiner can significantly increase mining performance. The data used in all experiments is the following data generated by Weka Library with Java program (Pérez 2005).

Data: T10I2D10.2kL348N600.

T (avg length of transaction): 10

I (avg length of max. freq. pattern): 2

D (number of transaction): 10,200

L (number of max. freq. pattern): 348

N (number of item): 600

Resulted H-struct and pattern number is:

H-struct Number: 9,478

L(1): 11

L(2): 26

L(3): 16

L(4): 1

Total patterns: 12,245

In the next section, the performance comparison between Apriori, FP-Tree, H-mine and LinkRuleMiner algorithms to mine frequent patterns on dyeing process data will be discussed. The main objective of the discussion is about is to compare performance of H-mine and the LinkRuleMiner algorithms. The rest of the algorithms are included only as reference. The time needed for each algorithm to complete mining in various minimum support is also shown using various diagrams. From the result we can see that the LinkRuleMiner algorithm has comparable performance with original H-mine algorithm in all cases of minimum support. There is no performance loss by removing link adjustment from the algorithm.

3. Performance study of Association Rule mining algorithms

To evaluate the efficiency and scalability of LRM, Hence, it is necessary to perform an extensive performance study. In this section, it is reported that the experimental results on the performance of the LRM in comparison with Apriori, FPGrowth and H-Mine. It shows that LRM outperforms Apriori, FPGrowth and H-Mine is efficient and highly scalable for mining very large databases.

All the experiments were performed on a 2.3 GHz Pentium PC machine with 2 GB main memory and 120 GB hard disk, running Microsoft Windows 7. LRM, H-Mine, FPGrowth and Apriori were implemented by us using Java 6.0. All reports of the runtime of LRM include both the time of constructing H-struct and mining frequent-patterns. They also include both CPU time and I/O time. Hence, it is tested with various data sets, and it yield consistent results. Limited by space, only the results on some typical data sets are reported here.

3.1. Mining transaction databases in the main memory

In this sub section, it is reported that the results on mining transaction databases can be held in the main memory. LRM is implemented as stated in Section 2. For FPGrowth, the FP-trees can be held in the main memory in the tests reported in this sub-section. The source code for FPGrowth, H-Mine and LRM transactions are loaded into the main memory and the multiple scans of database are pursued in the main memory. The Jayabala dyeing units (Margaret 2006), hundred shades contains 10,200 transactions i.e. ATEs, with up to 17 items per transaction. Figure 1 shows the minimum support and time in the Y-axis and association rule mining algorithms such as LRM, H-Mine, FPGrowth and Apriori on X-axis. Clearly, LRM has better results than the other algorithms, and the differences (in term of seconds) become larger when the support threshold goes lower.

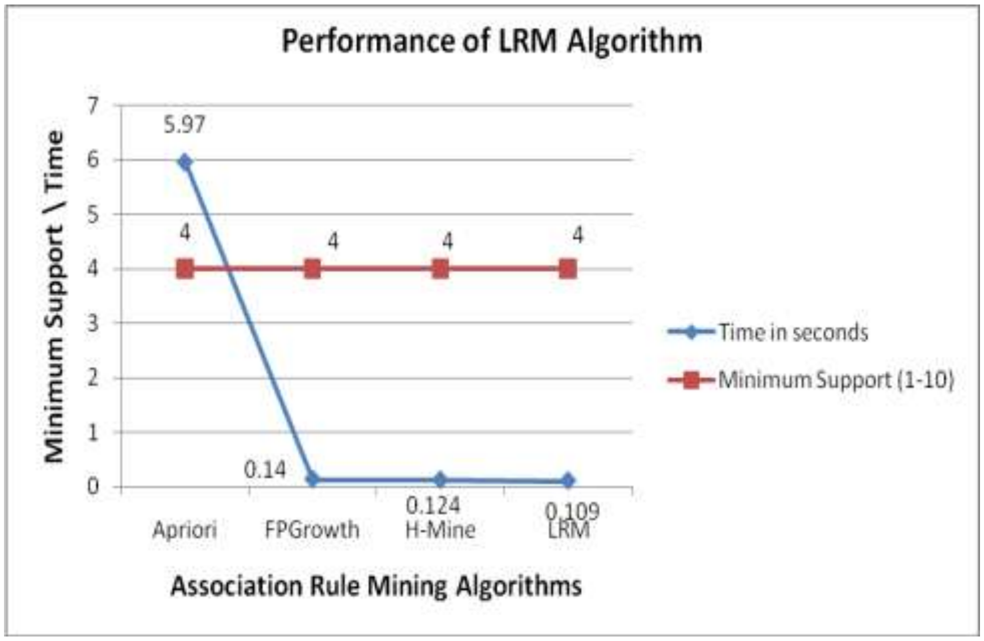


Figure 1. Comparison of Minimum Support, Runtime with the Association Rule Mining Algorithms on dataset of hundred shades of Jayabala dyeing unit

Apriori works well for sparse data sets since most of the candidates that Apriori generates turn out to be frequent patterns. However, it has to construct a hashing tree for the candidates and match them in the tree and update their counts each time when scanning a transaction that contains the candidates. That is the major cost for Apriori. FPGrowth has a similar performance to that of Apriori and sometimes it is even slightly worse.

This is because when the database is sparse, the FP-tree cannot compress data as effectively as it can for dense data sets. Constructing FP-trees over sparse data sets recursively has its overhead. Figure 2 plots the high water mark of space usage for LRM, FPGrowth and H-Mine in the mining procedure.

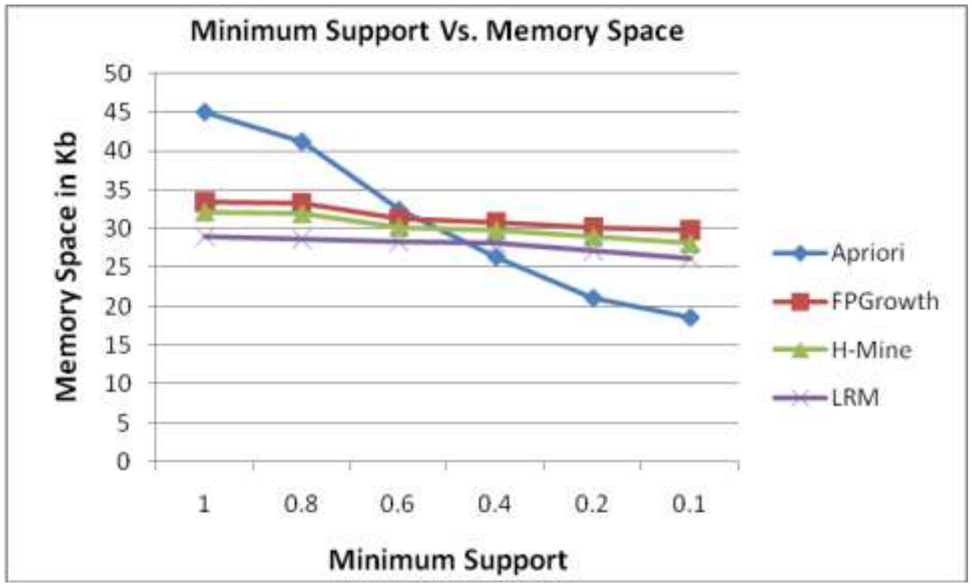


Figure 2. Space usage on data set of Jayabala dyeing unit Y-Axis represents the space usage in thousands and X-Axis represents Support threshold in %

To make the comparison clear, the space usage (Y axis) has a thousand scales. From the figure, it can see that LRM and H-Mine have similar space requirements and are very scalable in term of space usage with respect to the support threshold. Even when the support threshold reduces to very low levels, the memory usage is still stable and moderate. The memory usage of Apriori does not scale well as the support threshold goes down. Apriori has to store level wise frequent patterns and generate next level candidates. When the support threshold is low, the numbers of frequent patterns as well as that of candidates are non-trivial.

In contrast, pattern-growth methods, including LRM, H-Mine and FPGrowth, do not need to store any frequent patterns or candidates. Once a pattern is found, it is output immediately and never read back. What are the performances of these algorithms when applied to dense data sets? it uses the synthetic data set generator described in Agrawal and Srikant (1994) to generate a data set for hundred shades dyeing process of Jayabala dyeing process. This data set generator has been used in many studies on frequent pattern mining. Hence, it is also refereed the contributions from the various authors namely Agrawal and Srikant (1994) for more details on the data set generation. Data set hundred shades dyeing process contains 10,200 transactions and each transaction has up to 17 items. There are 1000 items in the data set and the average longest potentially frequent itemset has 15 items.

In dense data sets, such as hundred shades and forty eight shades dyeing process of H-Mine has very small number of frequent items. Thus, it has the same performance as LRM. Previous studies, e.g., Bayardo (1998), show that Apriori is incapable of mining such data sets. It is also tested with the scalability of the algorithms with respect to the average number of items in transactions in the dense data sets. The experimental results are consistent with the results reported in Agrawal and Srikant (1995): as the average size goes up, the runtime goes up linearly. H-Mine and LRM have a similar trend.

3.2. Mining very large databases

To test the efficiency and scalability of the algorithms to mine very large databases (Agarwal 1993), Hence, the it is generated that the data set of Emerald dyeing unit three shades dyeing process using the synthetic data generator. It has 12294 transactions i.e. ATEs with similar statistical features to the data set Jayabala dyeing unit's hundred shades dyeing process . Hence, it is enforced the memory constraints on LRM so that the total memory available is limited to 2, 4, 8 and 16 Mb, respectively. The memory covers the space for H-struct and all the header tables, as well as the related mechanisms. Since the FP-tree built for the data set is too big to fit into the main memory, hence, the performances of FPGrowth on this data set is not reported and also not explicitly impose any memory constraint on Apriori.

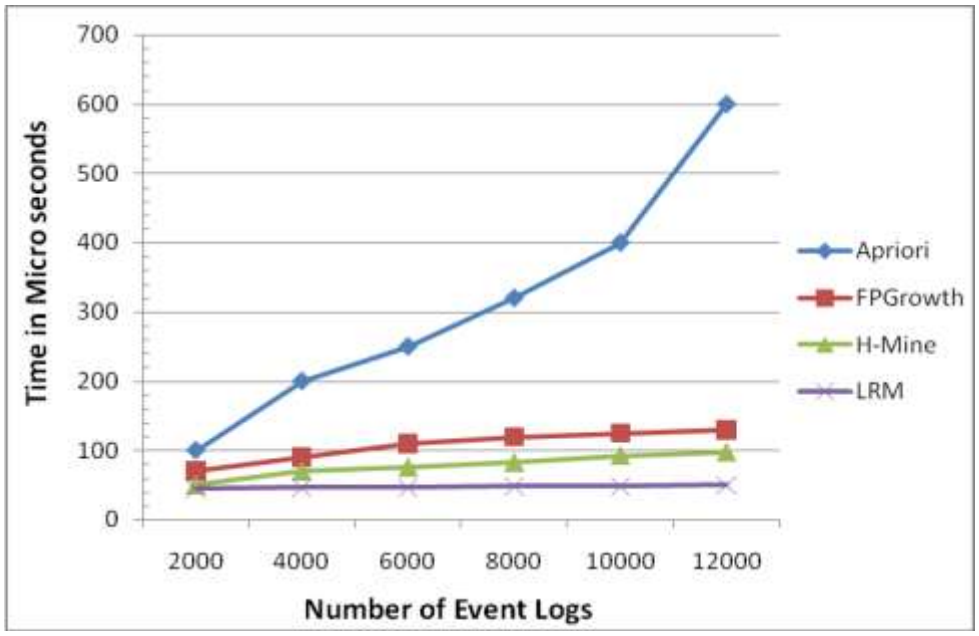


Figure 3. Scalability with respect to the number of transactions on data set of Jayabala dyeing unit

Figure 3 shows the scalability of both LRM (with the main memory size constrained to be 2 Mb) and Apriori with respect to the number of transactions in the database. In figure 3 the Y-axis the values has runtime in seconds and in X-axis number of transactions in thousands. Hence various support threshold settings were tested. Both algorithms have a linear scalability and LRM is a clear winner. From the figure, it can identify that the LRM is more efficient and scalable at mining very large databases. To study the effect of the memory size constraints on the mining efficiency and scalability of LRM in large databases, hence it is plotted the Figure 4, shows that the scalability of LRM with respect to the support threshold with various memory constraints, i.e., 2, 4, 8 and 16 Mb, respectively.

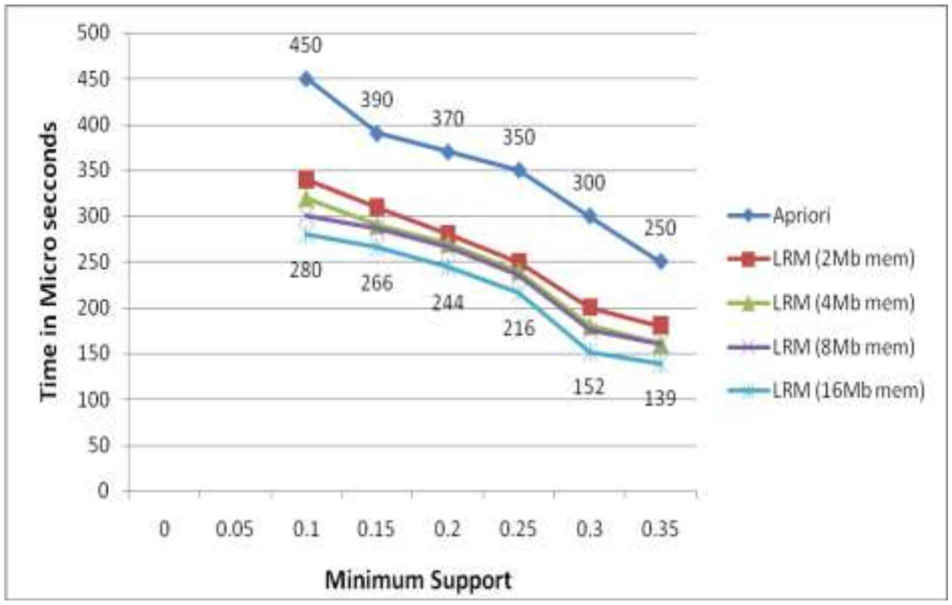


Figure 4. Scalability of LRM on large data set of Jayabala dyeing unit

As shown in the figure, the runtime is not sensitive to the memory limitation when the support threshold is high. When the support threshold goes down, as available space increases, the performance improves. Figure 4 shows the effect of available memory size on mining large data sets. At high support levels, the performance is not sensitive to the available memory size and thus the number of partitions. When the support threshold is low, the memory size plays an important role in determining performance. With a high support threshold, the number of frequent patterns is small and most frequent patterns are short. The dominant cost is the I/O cost and thus it is insensitive to the size of the available memory. When the support threshold is low, with a larger available memory, LRM has less partitions and thus generates fewer locally frequent patterns, i.e., the locally frequent patterns contain more globally frequent ones and less noise. Therefore, LRM can run faster with more memory.

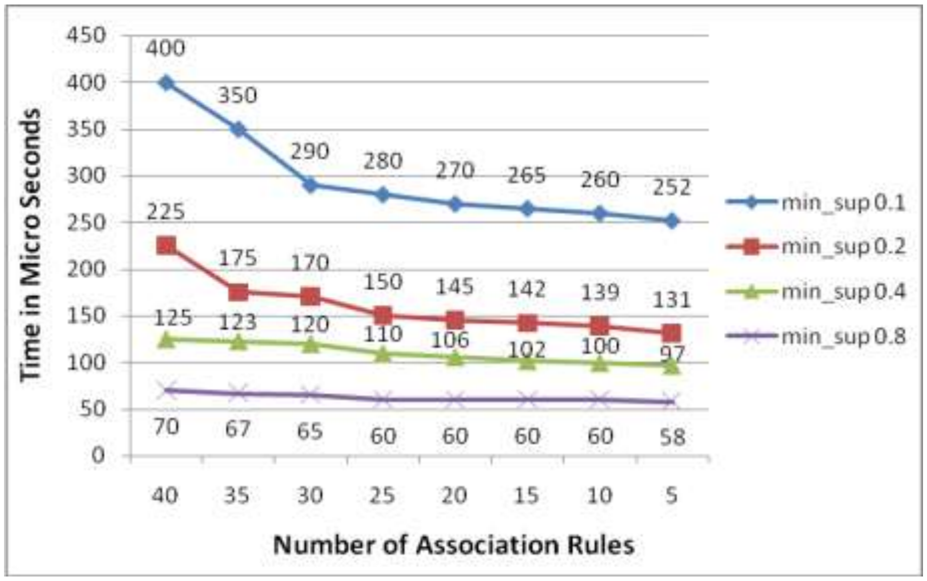


Figure 5. Result of LRM algorithm generated association rules and time using various minimum support values on dataset of Jayabala dyeing unit

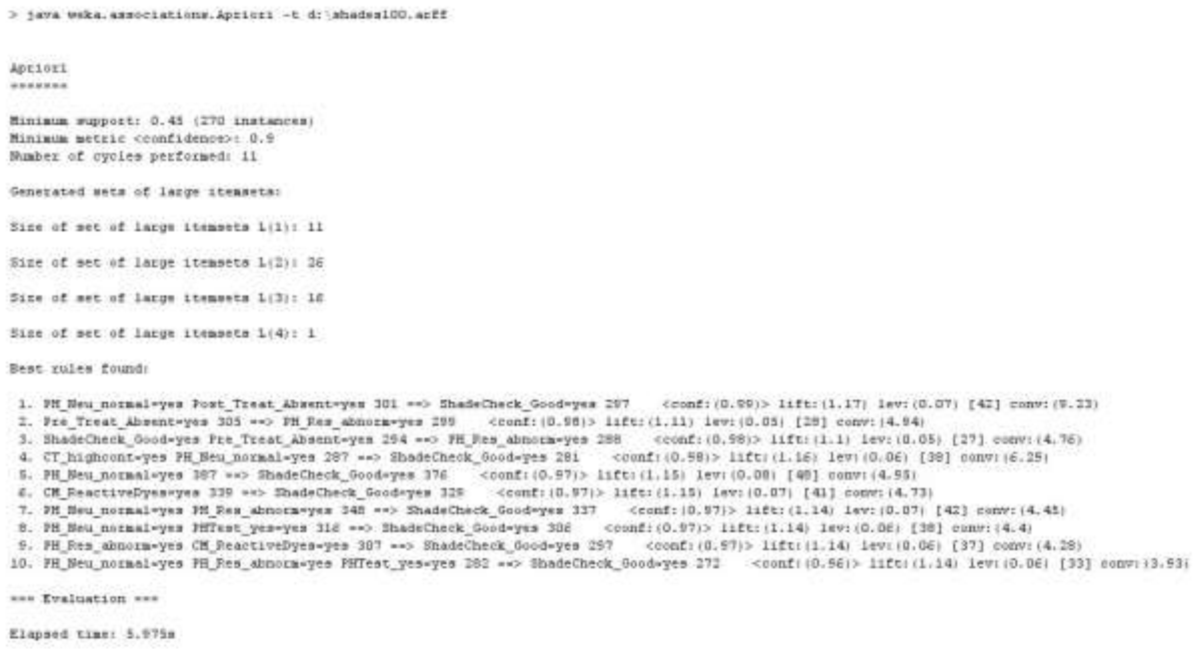
The results show that LRM can fully utilize the available memory to scale up the mining process. In general, as the support threshold goes down, the ratio goes up. This means that mining with a low support threshold may lead to some patterns being more frequent in certain partitions. On the other hand, less memory (small partition) leads to more partitions and also increases the ratio.

The figure 6 has the performance issue between association rule and different minimum supports. The figure 6 has minimum support values from 1 to 8 in the X-axis and time duration in micro seconds in Y-axis, the result was generated using LRM algorithm.

The performance of these algorithms were shown in the following diagrams figure 6, 7,8 and 9 respectively for Apriori, FPGrowth, H-Mine and LinkRuleMiner algorithms. These figures show that the execution time of each algorithm in Weka library with Java. The input parameters for these algorithms set to minimum threshold = 0.9, and minimum support = 0.1 and maximum support = 1.0.

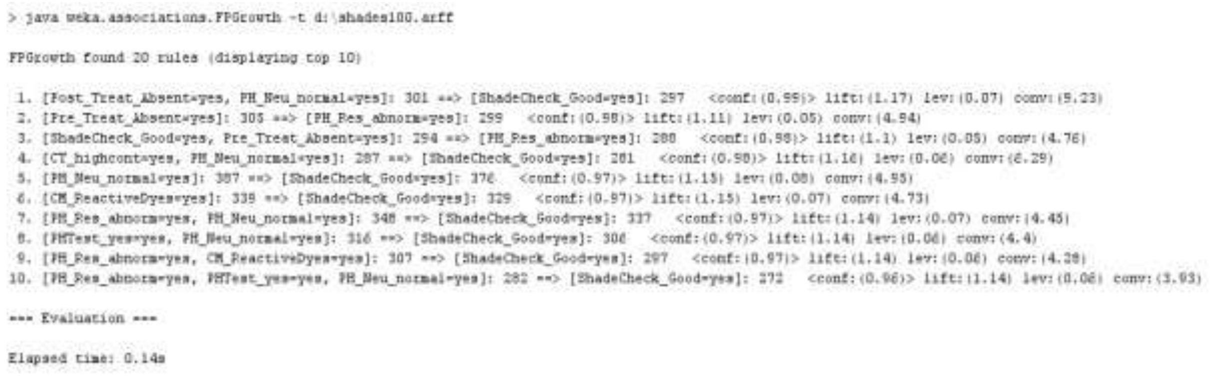
The processing time for hundred shades dyeing process of Jayabala dyeing unit using Apriori association rule mining algorithm is 5.975 seconds and it is shown in figure 6. The processing time for hundred shades dyeing process of Jayabala dyeing unit using FPGrowth association rule mining algorithm is 0.140 seconds and it is shown in figure 7. The processing time for hundred shades dyeing process of Jayabala dyeing unit using H-Mine association rule mining algorithm is 0.124 seconds and it is shown in figure 8. The processing time for hundred shades dyeing process of Jayabala dyeing unit using LinkRuleMiner association rule mining algorithm is 0.109 seconds and it is shown in figure 9.

It is clearly identified that each algorithm has its own processing time, here the H-Mine and LRM algorithms considerably minimizes the time to execute these algorithms. So the minimum time taken for executing the association rule mining algorithm is LinkRuleMiner, which requires only 0.109 seconds to execute 10,200 audit trail entries with 600 process instances.



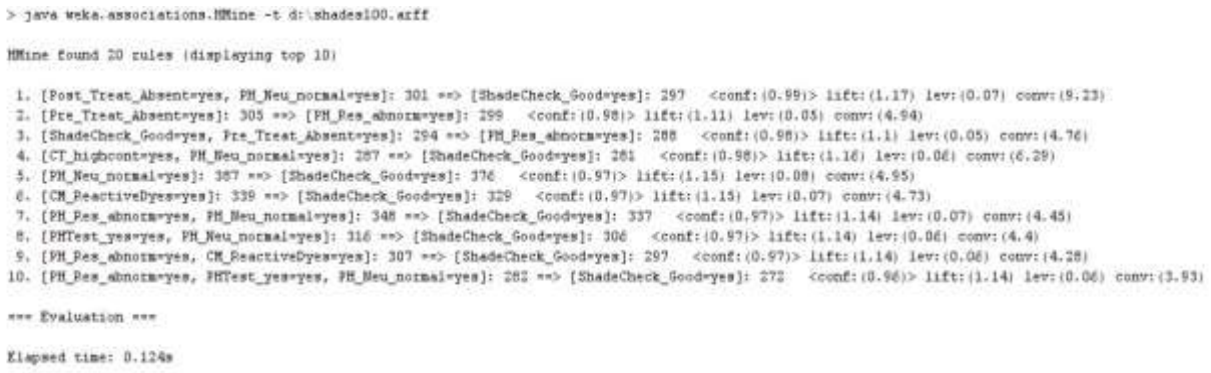
```
> java weka.associations.Apriori -t d:\shades100.arff


Apriori
=======

Minimum support: 0.45 (270 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 11

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 26

Size of set of large itemsets L(3): 16

Size of set of large itemsets L(4): 1

Best rules found:

 1. PH_Neu_normal=yes Post_Treat_Absent=yes 301 ==> ShadeCheck_Good=yes 297    <conf:(0.99)> lift:(1.17) lev:(0.07) [42] conv:(9.23)
 2. Pre_Treat_Absent=yes 305 ==> PH_Res_abnorm=yes 299    <conf:(0.98)> lift:(1.11) lev:(0.05) [29] conv:(4.94)
 3. ShadeCheck_Good=yes Pre_Treat_Absent=yes 294 ==> PH_Res_abnorm=yes 288    <conf:(0.98)> lift:(1.1) lev:(0.05) [27] conv:(4.76)
 4. CT_highcont=yes PH_Neu_normal=yes 287 ==> ShadeCheck_Good=yes 281    <conf:(0.98)> lift:(1.16) lev:(0.06) [38] conv:(6.25)
 5. PH_Neu_normal=yes 387 ==> ShadeCheck_Good=yes 376    <conf:(0.97)> lift:(1.15) lev:(0.08) [48] conv:(4.95)
 6. CM_ReactiveDyes=yes 339 ==> ShadeCheck_Good=yes 329    <conf:(0.97)> lift:(1.15) lev:(0.07) [41] conv:(4.73)
 7. PH_Neu_normal=yes PH_Res_abnorm=yes 348 ==> ShadeCheck_Good=yes 337    <conf:(0.97)> lift:(1.14) lev:(0.07) [42] conv:(4.45)
 8. PH_Neu_normal=yes PHTest_yes=yes 316 ==> ShadeCheck_Good=yes 306    <conf:(0.97)> lift:(1.14) lev:(0.06) [38] conv:(4.4)
 9. PH_Res_abnorm=yes CM_ReactiveDyes=yes 307 ==> ShadeCheck_Good=yes 297    <conf:(0.97)> lift:(1.14) lev:(0.06) [37] conv:(4.28)
10. PH_Neu_normal=yes PH_Res_abnorm=yes PHTest_yes=yes 282 ==> ShadeCheck_Good=yes 272    <conf:(0.96)> lift:(1.14) lev:(0.06) [33] conv:(3.93)

=== Evaluation ===

Elapsed time: 5.975s
```

Figure 6. Association rules using Apriori algorithm for Jayabala dyeing process of hundred shades whole log



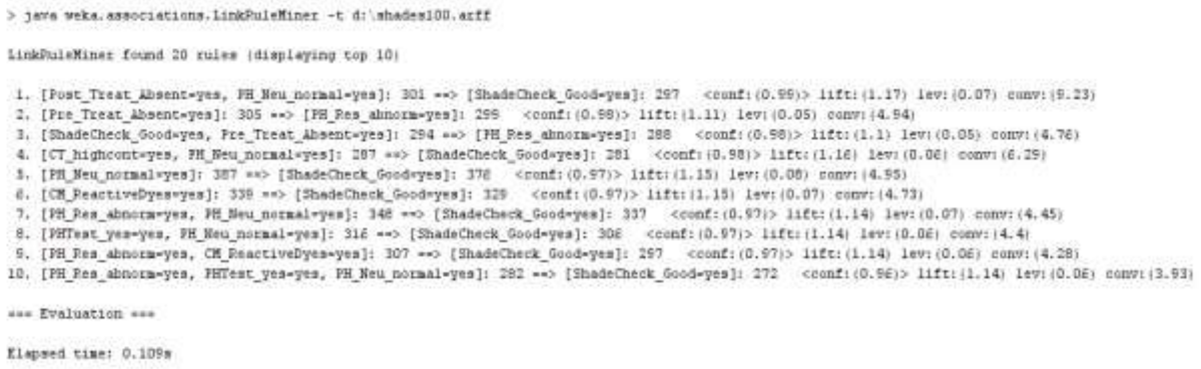
```
> java weka.associations.FPGrowth -t d:\shades100.arff

FPGrowth found 20 rules (displaying top 10)

 1. [Post_Treat_Absent=yes, PH_Neu_normal=yes]: 301 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.99)> lift:(1.17) lev:(0.07) conv:(9.23)
 2. [Pre_Treat_Absent=yes]: 305 ==> [PH_Res_abnorm=yes]: 299   <conf:(0.98)> lift:(1.11) lev:(0.05) conv:(4.94)
 3. [ShadeCheck_Good=yes, Pre_Treat_Absent=yes]: 294 ==> [PH_Res_abnorm=yes]: 288   <conf:(0.98)> lift:(1.1) lev:(0.05) conv:(4.76)
 4. [CT_highcont=yes, PH_Neu_normal=yes]: 287 ==> [ShadeCheck_Good=yes]: 281   <conf:(0.98)> lift:(1.16) lev:(0.06) conv:(6.29)
 5. [PH_Neu_normal=yes]: 387 ==> [ShadeCheck_Good=yes]: 376   <conf:(0.97)> lift:(1.15) lev:(0.08) conv:(4.95)
 6. [CM_ReactiveDyes=yes]: 339 ==> [ShadeCheck_Good=yes]: 329   <conf:(0.97)> lift:(1.15) lev:(0.07) conv:(4.73)
 7. [PH_Res_abnorm=yes, PH_Neu_normal=yes]: 348 ==> [ShadeCheck_Good=yes]: 337   <conf:(0.97)> lift:(1.14) lev:(0.07) conv:(4.45)
 8. [PHTest_yes=yes, PH_Neu_normal=yes]: 316 ==> [ShadeCheck_Good=yes]: 306   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.4)
 9. [PH_Res_abnorm=yes, CM_ReactiveDyes=yes]: 307 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.28)
10. [PH_Res_abnorm=yes, PHTest_yes=yes, PH_Neu_normal=yes]: 282 ==> [ShadeCheck_Good=yes]: 272   <conf:(0.96)> lift:(1.14) lev:(0.06) conv:(3.93)

=== Evaluation ===

Elapsed time: 0.14s
```

Figure 7. Association rules using FPGrowth algorithm for Jayabala dyeing process of hundred shades whole log



```
> java weka.associations.HMine -t d:\shades100.arff

HMine found 20 rules (displaying top 10)

 1. [Post_Treat_Absent=yes, PH_Neu_normal=yes]: 301 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.99)> lift:(1.17) lev:(0.07) conv:(9.23)
 2. [Pre_Treat_Absent=yes]: 305 ==> [PH_Res_abnorm=yes]: 299   <conf:(0.98)> lift:(1.11) lev:(0.05) conv:(4.94)
 3. [ShadeCheck_Good=yes, Pre_Treat_Absent=yes]: 294 ==> [PH_Res_abnorm=yes]: 288   <conf:(0.98)> lift:(1.1) lev:(0.05) conv:(4.76)
 4. [CT_highcont=yes, PH_Neu_normal=yes]: 287 ==> [ShadeCheck_Good=yes]: 281   <conf:(0.98)> lift:(1.16) lev:(0.06) conv:(6.29)
 5. [PH_Neu_normal=yes]: 387 ==> [ShadeCheck_Good=yes]: 376   <conf:(0.97)> lift:(1.15) lev:(0.08) conv:(4.95)
 6. [CM_ReactiveDyes=yes]: 339 ==> [ShadeCheck_Good=yes]: 329   <conf:(0.97)> lift:(1.15) lev:(0.07) conv:(4.73)
 7. [PH_Res_abnorm=yes, PH_Neu_normal=yes]: 348 ==> [ShadeCheck_Good=yes]: 337   <conf:(0.97)> lift:(1.14) lev:(0.07) conv:(4.45)
 8. [PHTest_yes=yes, PH_Neu_normal=yes]: 316 ==> [ShadeCheck_Good=yes]: 306   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.4)
 9. [PH_Res_abnorm=yes, CM_ReactiveDyes=yes]: 307 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.28)
10. [PH_Res_abnorm=yes, PHTest_yes=yes, PH_Neu_normal=yes]: 282 ==> [ShadeCheck_Good=yes]: 272   <conf:(0.96)> lift:(1.14) lev:(0.06) conv:(3.93)

=== Evaluation ===

Elapsed time: 0.124s
```

Figure 8. Association rules using H-Mine algorithm for Jayabala dyeing process of hundred shades whole log



```
> java weka.associations.LinkRuleMiner -t d:\shades100.arff

LinkRuleMiner found 20 rules (displaying top 10)

 1. [Post_Treat_Absent=yes, PH_Neu_normal=yes]: 301 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.99)> lift:(1.17) lev:(0.07) conv:(9.23)
 2. [Pre_Treat_Absent=yes]: 305 ==> [PH_Res_abnorm=yes]: 299   <conf:(0.98)> lift:(1.11) lev:(0.05) conv:(4.94)
 3. [ShadeCheck_Good=yes, Pre_Treat_Absent=yes]: 294 ==> [PH_Res_abnorm=yes]: 288   <conf:(0.98)> lift:(1.1) lev:(0.05) conv:(4.76)
 4. [CT_highcont=yes, PH_Neu_normal=yes]: 287 ==> [ShadeCheck_Good=yes]: 281   <conf:(0.98)> lift:(1.16) lev:(0.06) conv:(6.29)
 5. [PH_Neu_normal=yes]: 387 ==> [ShadeCheck_Good=yes]: 376   <conf:(0.97)> lift:(1.15) lev:(0.08) conv:(4.95)
 6. [CM_ReactiveDyes=yes]: 339 ==> [ShadeCheck_Good=yes]: 329   <conf:(0.97)> lift:(1.15) lev:(0.07) conv:(4.73)
 7. [PH_Res_abnorm=yes, PH_Neu_normal=yes]: 348 ==> [ShadeCheck_Good=yes]: 337   <conf:(0.97)> lift:(1.14) lev:(0.07) conv:(4.45)
 8. [PHTest_yes=yes, PH_Neu_normal=yes]: 316 ==> [ShadeCheck_Good=yes]: 306   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.4)
 9. [PH_Res_abnorm=yes, CM_ReactiveDyes=yes]: 307 ==> [ShadeCheck_Good=yes]: 297   <conf:(0.97)> lift:(1.14) lev:(0.06) conv:(4.28)
10. [PH_Res_abnorm=yes, PHTest_yes=yes, PH_Neu_normal=yes]: 282 ==> [ShadeCheck_Good=yes]: 272   <conf:(0.96)> lift:(1.14) lev:(0.06) conv:(3.93)

=== Evaluation ===

Elapsed time: 0.109s
```

Figure 9. Association rules using LinkRuleMiner algorithm for Jayabala dyeing process of hundred shades whole log

4. Conclusion

In this paper the performance of association mining algorithms such as Apriori, FPGrowth, H-Mine and LRM were discussed using dyeing processing system. From these algorithms, the one can analyze any type of the dyeing process to predict better treatment process to color the yarn effectively. Particularly the large databases can be executed efficiently in H-Mine and LRM algorithms, whereas the Apriori and FPGrowth are good when the size of the database is small. It is also compared with the time taken to process each algorithms, it concludes that H-Mine and LRM are the one which requires least time to execute. So the basic performance of each algorithm time and memory with various minimum support thresholds were discussed in this paper. Hence the complex and less-structured processes of any dyeing process can use H-Mine or LRM algorithms to identify the better dyeing process of Jayabala dyeing unit.

References

Agrawal. R and R. Srikant (1994). “Fast algorithms for mining association rules in large databases”. In J. Bocca, M. Jarke, and C. Zaniolo, editors, Proc. Int. Conf. on *Very Large Data Bases*, pages 478–499, Santiago, Chile.

Scheffer. T (2001). “Finding Association Rules That Trade Support Optimally against Confidence”. In Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (September 03 - 05, 2001). L. D. Raedt and A. Siebes, Eds. *Lecture Notes in Computer Science*, vol. 2168. Springer-Verlag, London, 424-435.

Srikant. R and R. Agrawal (1999), “Mining generalized association rules’. 1999, pg. 407–419.

Han J, Pei J, Yin Y (2000). “Mining frequent patterns without candidate generation”. In: Proceeding of the 2000 ACM-SIGMOD international conference on management of data (SIGMOD’00), Dallas, TX, pp 1–12.

Herbst. J. (2000). “A Machine Learning Approach to Workflow Management”. In Proceedings 11th European Conference on Machine Learning, volume 1810 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 183–194.

Agarwal R, Aggarwal CC, Prasad VVV (2001). “A tree projection algorithm for generation of frequent itemsets”. J Parallel Distribut Comput 61:350–371.

Pei J, Han J, Lakshmanan LVS (2001). “Mining frequent itemsets with convertible constraints”. In: Proceeding of the 2001 international conference on data engineering (ICDE’01), Heidelberg, Germany, pp 433–332.

Pérez, M. S., Sànchez, A., Herrero, P., Robles, V., and Pena, J. M (2005). “Adapting the weka data mining toolkit to a grid based environment”, In 3rd Atlantic Web Intelligence Conference, pp. 492-497.

Margaret. C Perivoliotis (2006), Wax Resist Decoration, “An Ancient Mediterranean Art”, Published by on line journal antciencia.com, ISSN 1646-3463, Vol. 2, No. 4, August-October’ 2006.

Agrawal, R., Imielinski, T and Swami, A. N. “Mining association rules between sets of items in large databases”. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 207-216.

.