A New Approach for Mining Representative Patterns

Abeda Sultana, Hosneara Ahmed, and Chowdhury Farhan Ahmed(✉)

Department of Computer Science and Engineering,
University of Dhaka, Dhaka, Bangladesh
abida1616@gmail.com, hosneara_17@yahoo.com, farhan@du.ac.bd

Abstract. With the revolution of science and technology, we can accumulate huge amount of data which requires to be manipulated efficiently since the amount of data is expanding hence scarcity of knowledge is also increasing. Therefore analysis for more useful and interesting knowledge is on demand. Representative patterns can be a solution to represent data in a more concise way. Different efficient methods for mining frequent and erasable patterns exist in representative pattern mining field that are regarded as significant. We have proposed a new type of pattern called decaying pattern. These patterns are characterized as those patterns that were frequent for a time being and then decayed with time. These patterns of declining nature can give us the opportunity to analyze reasons behind items' decrease such as extinct animals, finding unsolved accidental news, analysis of buying behavior of customers etc. that require further inspection.

Keywords: Frequent pattern · Erasable pattern
Representative pattern · Pattern tree · Decaying pattern

1 Introduction

Data mining is the process of analyzing large amount data to discover knowledge and finding patterns and relationship among them. By means of data mining we can renovate huge amount of information into useful knowledge. This knowledge is immensely important on various applications and research field. Frequent pattern mining is one of the most significant field of data mining. Another important domain of data mining is mining representative patterns. Different kind of frequent patterns can be formed in itemsets, sequences, episodes and substructures and so on. Representative frequent pattern mining refers to finding precise, distinctive and explicable set of patterns of each class that represent their key characteristics among other classes. Representative patterns represent a dataset and extract the significant knowledge from huge amount of data. This representation can be done in many criteria. Many efficient and noble works are already done on these e.g. some patterns are most significant in the database, that are mined as representative frequent patterns, some patterns are so insignificant that they are better to be pruned, these type of patterns are mined as representative erasable patterns [7,9,10]. Again maximal, closed, top rank-k, top-k frequent regular [1,2,6] are also developed for representative pattern mining.

There are some patterns which are frequent in a database for some time being, then they are lost. These patterns could be a part of representative frequent pattern set but due to their decaying nature, they fail to get a place there. This type of patterns are ignored but have importance in various research fields. We focus on this decaying representative patterns which can be a means of mining important knowledge from huge amount of data. Erasable closed pattern set consists of the least frequent patterns and subsets of these patterns are also erasable. On the other hand, decaying pattern has two attributes- at first it needs to be highly frequent for a span of time, secondly its frequency will decay and so the resulting pattern becomes infrequent.

1.1 Motivation

With the existing algorithms we can only find the patterns which were most frequent or too scarce. Patterns having both characteristics- that were frequent for a while but have become infrequent with time are not mined yet. This type of patterns are not observed but have importance in various research fields.

Motivating Example

Suppose a large electronics company sells various types of products like laptops, smartphones, smartwatches, tablets etc. Laptops and smartphones are very popular among the buyers of that company. These two products were sold throughout the year. On the other hand, smartwatches are not greatly welcomed by the customers, so the sale of smartwatches remained below the expectation. However, in case of tablet situation was slightly disparate. Notwithstanding, when company released it first, it was a very welcoming product, after there second release the sale suddenly decreased.

After two or three years when a data analyst of that company mined the most remunerative products, he got laptop and smartphone as the number of sale was higher and so the profit. Contrarily, as least profitable product, he got smartwatch. So company will take action for further development of it. As tablets do not fit in any of the two categories, company will never know the problem why the sale of tablets decreased and it will not take any necessary step. If analyst would observe decaying nature in the trade, this tablet will come to light for finding the reason behind consumers' sudden displeasure toward this product.

There are more important applications of decaying patterns. Everyday accidents or unusual occurrences are happening that appear in newspaper and social media so frequently for some days and then perish of being heedless. In most cases, they remain unresolved. Mining those patterns can help correspondents write follow-ups. Again, we can find this pattern in species data of animals and plants which have become extinct such as Sea Mink, Tasmanian Tiger, West African Black Rhinoceros and so on. In many cases this decaying nature is prevailing but does not come to light. This gradual going off detection is our main purpose of proposing new type of pattern.

1.2 Contributions

- We have proposed a concept of new type of representative patterns named “Decaying patterns” which represents those which were once in frequent pattern set but decayed with time. This could be put into representative pattern set but due to fall off nature, they fail to remain there.
- We have developed an algorithm to mine this type of patterns from real life large datasets which are collected from famous data mining repositories FIMI and SPMF.
- Data that we have used to test are considered as data stream so we have divided it into set of windows and for any current window we have observed whether it is frequent or erasable which assures getting recent result always.
- We have run our algorithm on six real life datasets and two synthetic datasets. Further our own web service to collect news from prominent online newspapers provided us with floods of daily news. We have pre-processed that huge data and applied our algorithm. From all of these real life large datasets, significant number of decaying patterns have been observed.

The rest of the paper is organized as follows. In Sect. 2, some overview of related works on representative pattern has been given. Section 3 consists of our proposed approach, algorithm and a small simulation to exemplify it. Section 4 contains the experimental analysis based on different performance metrics using the algorithm on many real life and synthetic datasets. Finally, in Sect. 5, we concluded with discussion on the future scope of robustness of our algorithm.

2 Related Work

Maximal Frequent Itemsets [3]**:** Low min_sup generates large number of patterns. Bayardo [3] proposed for storing long patterns (maximal frequent itemsets) in roughly linear scale. If a pattern is X is frequent, all Y where $Y \subset X$ is frequent.

FPclose [4]**:** Implements another distinction of FP-tree known as CFI-tree (Closed Frequent Itemset Tree). Four fields are necessary for this tree structure - item name, count, node link and level. Subset test of maximality is done with level. Count works for checking if the support count is equal to it’s superset and if it is not, both superset and subset are stored in memory. FP-close is the fastest among the algorithms of that time when minimum support is low but when minimum support is high it becomes slow than Apriori.

TFP [12]**:** For mining top-k frequent closed items, TFP is an efficient algorithm.

The common factor among all approaches of frequent pattern mining is the usage of *min_sup* threshold which ensures generation of accurate and entire set of frequent item sets which leads to two problems stated below -

First, an appropriate *min_sup* is taken as input but this requires detailed knowledge on mining query. Again setting min support is quite problematic in the sense that a too small threshold may produce thousands of itemsets on the other hand a too big threshold may generate no answers.
Second, Due to the downward closure property, when a long itemset is mined, it may generate an exponential number of itemsets.

To solve these problems they proposed a new approach of mining top-k frequent closed itemsets of minimum length *min_l*, where k is user given number of frequent closed itemset that they want to be mined. k is easy to specify and top-k means k most frequent closed itemsets. *min_l* helps to mine long itemset without mining the short ones first.

ECP (Erasable Closed Pattern) [11]**:** In factories, for the optimization of production plans erasable pattern (EP) mining plays an important role. For efficient mining of these patterns various algorithms have been proposed. Nevertheless, number of EPs becomes numerous because of large threshold values which cause memory usage overhead. Hence it becomes requisite to mine compressed EPs representation. This paper first came up with the concept of erasable closed patterns (ECPs). These ECPs can be represented without losing information. They at first gave a theory to detect ECPs based on a structure name dPidset and proved it. Then two efficient algorithms, ECPat and dNC-ECPM are proposed. Their result of experiment on these two algorithm shows that for sparse datasets ECPat performs the best but ECPM algorithm is more efficient in the case of memory usage and runtime for rest of the datasets.

DSTree (Data Stream Tree) [8]**:** In this paper, Leung et. al. proposed the concept of data stream tree. Transactions are sorted in any canonical order chosen by user. Each node keeps a list for frequency count. With the appearance of a new batch of transaction, it is appended to the list to each of the node and frequency count of that node in the current batch. The last entry of a certain node N is the frequency count of that node in the current batch. When the next batch of transactions arrives after fulfilling the batches in a window, the list is shifted to left to place the newest batch to be added as the most recent one. In DSTree costly deletion is not required, only shifting and updating the frequency list will suffice to update tree.

3 Our Proposed Approach

Decaying patterns are important and useful for analysis and many other purposes in many data repository. To the best of our knowledge, it is the first approach for mining decaying pattern. By observing the enormous application in many sectors, we hope these patterns will be effective and useful for gaining knowledge.

As we proposed our algorithm to work on stream of data, we used DSTree based algorithm which serves pretty well for the data stream. Again we have proposed a pattern tree approach to mine decaying patterns and sub patterns of different length, which is also a significant part of our algorithm. We consider sliding window technique for capturing data effectively. We also thought about including dynamic future of window and batch which we'll adopt as our future contribution.

3.1 Preliminaries

A window is comprised of a set of batches and a batch consists of a set of transactions.

- *win_F* (Frequent window length in decaying pattern) The number of consecutive windows where a pattern has to be frequent.
- *win_IF* The number of consecutive windows where a pattern needs to be infrequent. (Difference between total window size & *win_F*).
- *min_WT* (Minimum window support threshold) A threshold that is to be crossed by a pattern by occurrence to be frequent in a window.
- *ET* (Error threshold in *win_F*) The maximum number of batches in *win_F* where a pattern can be infrequent.
- ET' (Error threshold in *win_IF*) The maximum number of batches in *win_IF* where pattern can be frequent.

Definition 1 (Decaying pattern)**.** Consider a set of batches $B = \{b_1, b_2, b_3, \ldots, b_n\}$ where each batch consists of a number of transactions $T = \{t_1, t_2, t_3, \ldots, t_m\}$. If a pattern is frequent (meets *min_WT*) in frequent windows (each window of *win_F*) and then becomes infrequent in decay windows (each window of *win_IF*), it is called decaying pattern.

e.g. a pattern with a window length 9 and the frequency list of the pattern in these windows is [9, 5, 10, 4, 3, 3, 2, 0, 0]. Let the *min_WT* is 4, frequent window length *win_F* is 4. Here the for first four window [9, 5, 10, 4] the pattern meets *min_WT*(4). The pattern remains infrequent for last five windows *win_IF* [3, 3, 2, 0, 0]. So this pattern is our desired decaying pattern.

- Batch size and window size generation
  The size of batch and windows will be based on the number of transaction in a database. In a database with small transactions this value should be kept as small as possible. If user wants to observe the decaying characteristics intensely then he should keep batch size small because the window will move slowly in that case. In general case, as the batch size increases, number of decaying patterns also increases. For sparse dataset, batch size should be as large as possible for this reason.
- *win_F* and *win_IF* generation
  For dense dataset

$$win\_F = \frac{Total\ window\ size}{2}$$

or

$$win\_F = \frac{2 \times Total\ window\ size}{3}$$

For sparse dataset

$$win\_F = \frac{Total\ window\ size}{3}$$

$$win\_IF = Total\ window\ size - win\_F$$

But this can be tuned by user according to his demand.

- Minimum window support min_WT
  We represent the minimum window support as a percentage value. A minimum window support threshold of 60% means a pattern has to appear in at least 60% of all the transactions in current window of the data stream. The min_WT value is calculated as follows:

$$min\_WT = min\_sup\ \% \times\ number\ of\ transactions\ in\ current\ window$$

- ET and ET' generation
  The error threshold in win_F and win_IF should be 1% for dense dataset which refers that a pattern which remains frequent in at least 99% windows of win_F and remains infrequent in at least 99% windows of win_IF, patterns will be accepted as decaying pattern. For sparse dataset ET and ET' can vary from 10% to 20%.

3.2 Tree Construction and Mining

We have used FP-Growth [5] for mining frequent patterns in each window, additionally, as data is considered as stream, mechanism of DSTree [8] is followed for sliding window formation which has been modified according to our purpose.

The transactions in the tree are sorted in frequency descending order for static database. In case of data stream any canonical order can be followed such as alphabetic order or order based on any specific property. With every node in the tree a frequency list is added which contains the frequency of current batches of that item. A batch of transactions is inserted at a time and frequency of each item is appended to the frequency list of each node. When new batch is added, the list is shifted to left and the oldest batch frequency is removed. This has the same effect as deleting the transactions from the oldest batch in a window.

Frequent patterns from each window are added to a new pattern tree. Each node in a pattern tree also consists of a frequency list. Frequent patterns generated from each window of data stream tree are considered as a batch of transactions for the pattern tree. The frequency list of each node contains the frequency of that node in a particular batch. Each batch of frequent patterns are inserted at a time into tree.

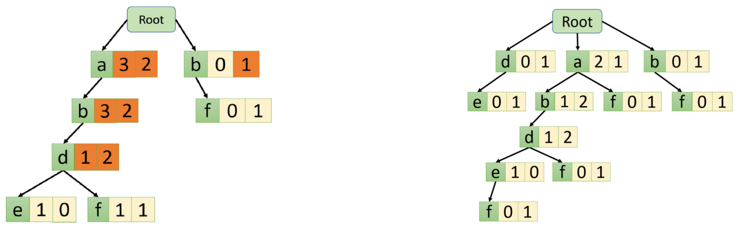
When a complete pattern tree is generated, we extract the patterns of our interest by checking only the leaf node. If a leaf node meets the condition of being a decaying pattern the whole branch and the sub-branch of a path including the leaf node will be considered as decaying pattern.

Example Workout

- In Table 1 a small transaction dataset is shown. There are seven items {a, b, c, d, e, f, g} and fifteen transactions. We have divided the transactions into five batches.
  Each batch consists of three transactions, batch length BS is 3. Each window consists of 2 batches, window size WS is 2. We consider our minimum window support threshold min_WT 3 which means a pattern will be frequent in a window if the total frequency of a pattern in the batches of that window is at least 3.
  Here, frequent window length in decaying pattern, win_F is 2. So a decaying pattern will be frequent in consecutive 2 frequent windows.
- Now we have to construct a DSTree with these batches. In Fig. 1a the tree is constructed with batch-1 and batch-2. The frequency of each batch is inserted in the frequency list with each node. e.g. 'a' has value 3 and 2 in its list which refers to the frequency of 'a' in first batch is 3 and in second batch is 2. In each window, we mine frequent pattern from the tree with FP-growth algorithm. The total frequency of 'a' in window 1 is 5 which is greater than min_WT, so 'a' is a frequent pattern for window 1.
  In this window, we have found {a}, {b}, {d}, {a, b}, {b, d}, {a, d}, {a, b, d} as frequent patterns.

Table 1. Transactions are arranged in frequency descending order

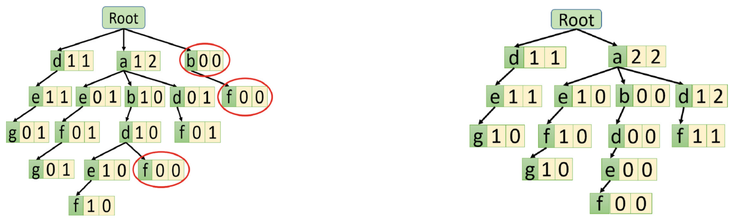
Batch	Transactions	Contents
First	t_1	{a, b, d, e}
	t_2	{a, b, f, d}
	t_3	{a, b}
Second	t_4	{a, b, d, f}
	t_5	{b, f}
	t_6	{a, b, d}
Third	t_7	{a, b, d, e, f}
	t_8	{d, e}
	t_9	{a, f}
Fourth	t_{10}	{a, d, f}
	t_{11}	{d, e, g}
	t_{12}	{a, e, f, g}
Fifth	t_{13}	{a, d}
	t_{14}	{a, d, f}
	t_{15}	{d, e, g}



(a) Tree after including first two batch (b) Tree after inserting 3rd batch

Fig. 1. Window 1 & 2

– For second window, we have to insert third batch by removing the oldest batch from the tree. We shifted the frequency list of each node to left and added the frequency of new batch to the right. Again by mining frequent patterns we got {a}, {b}, {d}, {f}, {a, b}, {a, d}, {b, d}, {a, b, d} (Fig. 1b).



(a) Tree after inserting 4th batch (b) Tree after inserting 5th batch

Fig. 2. Window 3 & 4

– For third window (Fig. 2a), after inserting fourth batch we got {a}, {e} as frequent patterns.
– On fourth window (Fig. 2b) we got {a} and {g} as frequent patterns.

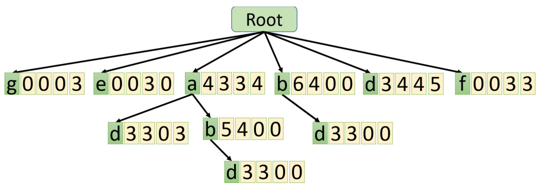


Fig. 3. Pattern tree with the frequent patterns from all windows

– We have built a pattern tree with all the patterns found above, considering the patterns from each window as a batch of transactions to insert in the tree. Each node also consists of a frequency list (Fig. 3).
– Now from this pattern tree, we can easily find out desired pattern only by checking the leaf nodes. The decaying patters are - {a, b, d}, {b, d}, {a, d} (Fig. 3).

Algorithm 1: Algorithm for Mining Decaying Patterns

```
Input  : transactions[] where each transaction consists of items, $min\_sup$,
         $win\_F$ and $win\_IF$, $ET$ and $ET'$
Output: List of decaying patterns in structure named $Decay\_Patterns$
1  Sort transactions[] in lexicographic order
2  $root \leftarrow$ Add_Batch_to_Tree(Batch$_0$,...,Batch$_m$)
3  PatternSet $\leftarrow$ FPgrowth(tree)
4  foreach Remaining batch Batch$_i$ do
5      Add_batch_to_tree(Batch_i, root)
6      Pattern_set[window++] $\leftarrow$ FP_growth(tree)
7      Pattern_tree(pattern_set, root)
8  end
9  Decay_Patterns $\leftarrow$ Extract_Pattern(root)
10 Function Extract_Pattern(root)
11     foreach leaf node in tree do
12         if ItemFreq in win_F $\geq$ min_WT and ItemFreq in each batch of win_IF
           == 0 then
13             keep the node in tree
14             Decay_Patterns[count++] = $\beta \cup$ leaf node
15             Prune the leaf node upto root;
16         else
17     end
18     return Decay_Patterns
```

4 Experimental Results

We tested our algorithm on six real life and two synthetic datasets- chess, mushroom, pumsb, accidents, connect, c73d10k, c20d10k. Also we have a web service where we fetch data from some prominent online news portal of Bangladesh. We collected around 59,033 news in 4 months (August to November). We processed the data and applying our algorithm, got our desired result. The algorithm is implemented in JAVA and experiments are performed in Linux environment (Ubuntu 16.04), on a PC with Intel(R) Core-i3-4005U 1.70 GHz processor 4 GB main memory. As there is no literature on finding decaying pattern, we could not compare our result with any other algorithm. For this reason we have shown our result on five metrics.

Algorithm 2: Algorithm for Mining Decaying Patterns

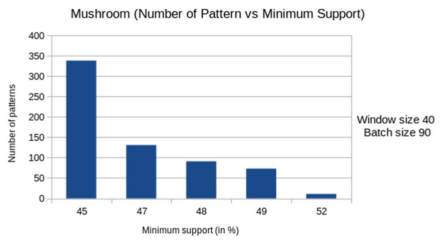
```
19 Function Add_Batch_to_Tree(batch[], root)
20     if root is NULL then
21         foreach $batch_i$ in batch[] and transaction t in $batch_i$ do
22             add t to tree and nodeFrequency to list
23         end
24     else
25         Shift left each node in tree
26         foreach $batch_i$ in batch[] and each transaction t in $batch_0$ do
27             add t to tree
28             add frequency of node to list
29         end
30     end
31     if frequency of each window is 0 of any node in tree then
32         delete node and its successors
33     end
34     return root
35 Function Pattern_Tree(patterns[])
36     foreach $pattern_i$ in patterns[] do
37         Add $pattern_i$ to tree
38         insert frequency of node to freq_list
39     end
```

– Number of patterns with varying minimum window support
– Number of patterns with varying window size
– Number of patterns with varying batch size
– Runtime with varying minimum support
– Maximum memory usage with varying minimum support.

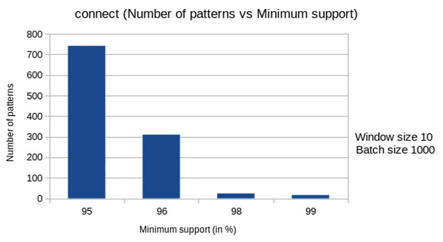
4.1 Pattern Count w.r.t. Minimum Window Support

With varying minimum support value, number of decaying patterns also varies. Number of patterns and minimum support change proportionally to each other but in case of decaying pattern this trend does not hold always. Plausibly because when minimum support is low, the patterns tend to be frequent in the decaying window (win_IF) and those patterns are rejected as per the definition of decaying pattern. When minimum support is higher those rejected patterns are added in decaying pattern list. From Fig. 4, result of two dense datasets (Fig. 4b and a) connect and mushroom are shown where pattern number decreases with increasing min_WT while results of sparse dataset accident and c20d10k (Fig. 5b and a respectively) are different. Number of patterns tends to increase with increasing min_WT for accidents. Nearly reverse nature is noticed for c20d10k. So this variation actually depends on dataset.

Our news dataset was highly sparse as we got news of four months only. For better output we splitted the dataset by taking two months' news in a

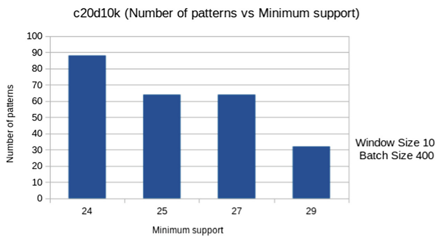


(a) Mushroom dataset

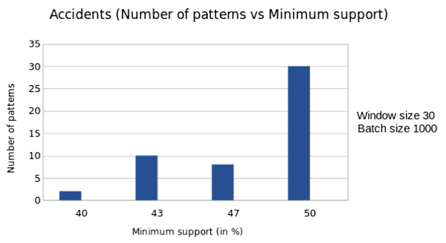


(b) Connect dataset

Fig. 4. Pattern number vs Minimum window support (min_WT)



(a) c20d10k dataset



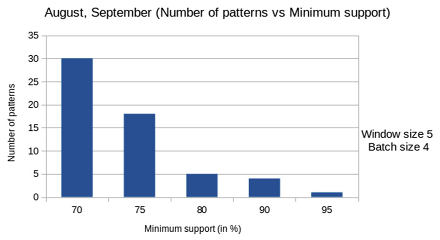
(b) Accident dataset

Fig. 5. Pattern number vs Minimum window support (min_WT)

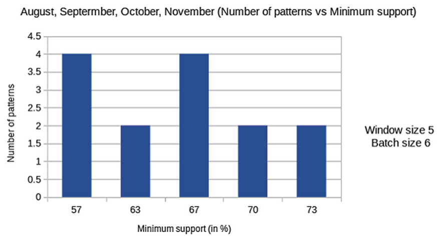
group. From the news of August and September, we observed several decaying patterns (Fig. 6a). Most of them are murder incidents including 16th amendments of Bangladesh constitution, floods in northern part of Bangladesh. From August to November (Fig. 6b), the important decaying news mostly are rape and murder case including some international matters like Rohinga issue, issue of mosque Al-Aqsa in Palestine etc.

4.2 Pattern Count w.r.t. Window Size

In case of dense dataset generally pattern number tends to increase with increasing window size (Fig. 7a) because in larger window, longer decaying pattern can be generated and in that case there will be lot of sub patterns. But in case of sparse dataset the opposite tendency is noticed (Fig. 7b) because sparse dataset contains small number of decaying patterns and with increase of window size possibility of being frequent in a large set of transaction decreases. For this reason, many patterns are rejected as they are infrequent in the frequent windows. This characteristic varies with dataset.

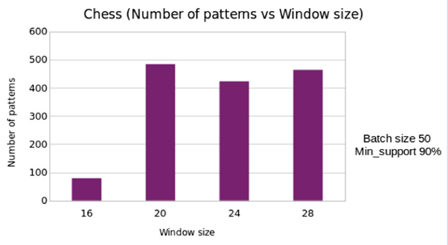


(a) News dataset
(August and Semptember)

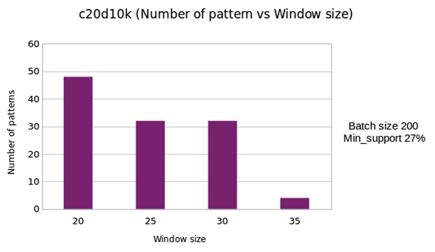


(b) News dataset
(August to November)

Fig. 6. Pattern number vs Minimum window support (min_WT)



(a) Chess dataset

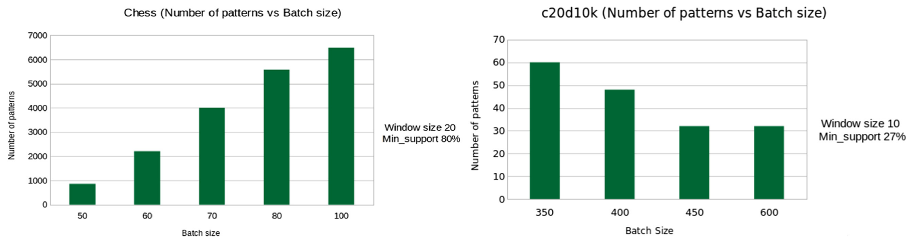


(b) C20d10k dataset

Fig. 7. Pattern number vs Window size

4.3 Pattern Count w.r.t. Batch Size

The next metric is number of patterns w.r.t batch size. Similar characteristic is observed as to previous metric. In case of dense dataset (Fig. 8a) number of patterns increase with increasing batch size because of longer pattern generation and when batch size increases total number of window decreases. So a pattern has to be frequent and infrequent in small number of windows. This increases the probability of getting decaying pattern with increasing batch size. Again, for sparse dataset (Fig. 8b) number of patterns tends to decrease because with batch size is larger window slides faster. In case of sparse dataset, frequent patterns in one window tend to become infrequent in subsequent windows which result in small number of patterns. This characteristic also varies with dataset as depicted in the figures. So we cannot conclude a rigid relationship between pattern count and batch size in mining decaying patterns.

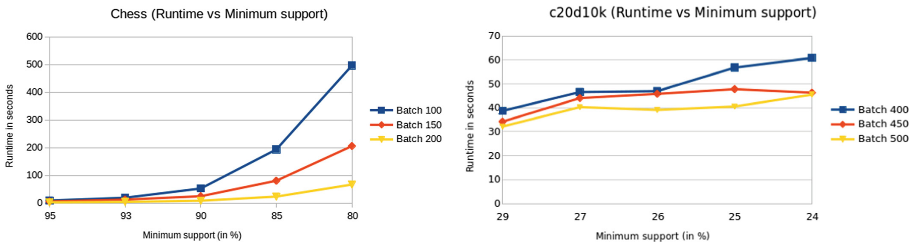


(a) Chess dataset (b) C20d10k dataset

Fig. 8. Pattern number vs Batch size

4.4 Runtime Evaluation

Runtime graphs are shown with varying *min_WT* on three batches for a dense [Chess] (Fig. 9a) and a sparse dataset [c20d10k] (Fig. 9b). Run time depends on the number of patterns generated and total number of windows to calculate. From the graph, it is comprehensible that run time increases with decreasing minimum support and batch size. For sparse dataset, result is a bit different. With increasing batch size, number of patterns decreases as before which requires greater runtime. In case of sparse dataset, pattern number changes differently with varying *win_WT* so as the runtime.

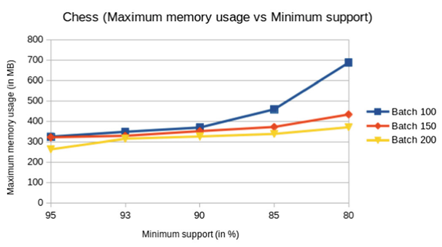


(a) Chess dataset (b) C20d10k dataset

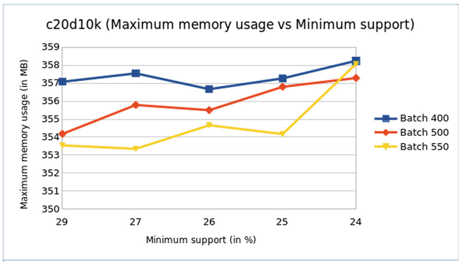
Fig. 9. Runtime (in second) vs *win_WT*

4.5 Maximum Memory Usage Evaluation

Memory usage also depends on the number of patterns generated. With varying minimum support, the variation of maximum memory usage during run time is shown. We determined maximum memory usage for any instance. During the execution of the code, we have kept the maximum value of memory usage. If at any instance more memory is used than the value, we have updated it. Here we have shown memory consumption of two dataset- chess (Fig. 10a), the dense one and c23d10k (Fig. 10b), the sparse one.



(a) Chess dataset



(b) C20d10k dataset

Fig. 10. Maximum memory usage (in MB) vs win_WT

5 Conclusions

Many significant patterns are stale with time which should be in representative pattern and need proper attention. This type of patterns are important because if we only focus on those which are always in representative set, some promising patterns that suddenly started decaying will remain neglected. For this, mining this type of patterns are important for different contexts. We have developed an algorithm for mining decaying patterns and applied it by merging the concept of data stream. We have constructed pattern tree structure which speeds up the mining process. As we are dealing with data stream, more interesting knowledge can be found at any instance of time from the patterns.

The application field of this algorithm is huge beginning from market basket data to find new characteristics in dataset. We applied the work on many real life dataset and have got expected results. Our work is highly applicable for mining decaying news and we have depicted significant result from our own processed news data. As a future work, we are planning to develop more compressed structure of tree, applying more efficient mining methodology. We will also carry out more research for generalizing the algorithm so that it can be performed with dynamic window adjusting feature to get the best result without user's input.

References

1. Amphawan, K., Lenca, P., Surarerks, A.: Efficient mining top-k regular-frequent itemset using compressed tidsets. In: Cao, L., Huang, J.Z., Bailey, J., Koh, Y.S., Luo, J. (eds.) PAKDD 2011. LNCS (LNAI), vol. 7104, pp. 124–135. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28320-8_11
2. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-k regular-frequent itemsets using database partitioning and support estimation. Expert Syst. Appl. **39**(2), 1924–1936 (2012)
3. Bayardo, R.J. : Efficiently mining long patterns from databases. In: Proceeding of the ACM-SIGMOD International Conference on Management of Data, pp. 85–93 (1998)

4. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-trees. IEEE Trans. Know. Data Eng. **17**(10), 1347–1362 (2005)
5. Han, J., Pei, J., Yin, J.: Frequent patterns without candidate generation a frequent-pattern tree approach. Data Min. Knowl. Disc. **8**(1), 53–87 (2004)
6. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining top-k frequent closed pat- terns without minimum support. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), Maebashi City, Japan, 9–12 December, pp. 211–218 (2002)
7. Lee, G., Yun, U., Ryang, H.: Mining weighted erasable patterns by using underestimated constraint-based pruning technique. J. Intell. Fuzzy Syst. **28**(3), 1145–1157 (2014)
8. Leung, C.K.S., Khan, Q.I.: DSTree: a tree structure for the mining of frequent sets from data streams. In: Proceedings of the Sixth International Conference on Data Mining (ICDM 2006), pp. 928–932. IEEE Computer Society, Washington, DC (2006)
9. Nguyen, G., Le, T., Vo, B., Le, B.: Discovering erasable closed patterns. In: Nguyen, N.T., Trawiński, B., Kosala, R. (eds.) ACIIDS 2015. LNCS (LNAI), vol. 9011, pp. 368–376. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15702-3_36
10. Nguyen, G., Le, T., Vo, B., Le, B.: EIFDD: an efficient approach for erasable itemset mining of very dense datasets. Appl. Intell. **43**(1), 85–94 (2015)
11. Vo, B., Le, T., Nguyen, G., Hong, T.: Efficient algorithms for mining erasable closed patterns from product datasets. In: IEEE Access, p. 1 (2017)
12. Wang, J., Han, J., Lu, Y., Tzvetkov, P.: TFP: an efficient algorithm for mining top-k frequent closed itemsets. IEEE Trans. Knowl. Data Eng. **17**(5), 652–664 (2005)