Recent Frequent Item Mining Algorithm in a Data Stream Based on Flexible Counter Windows

Yanyang Guo
School of Information Engineering, Yangzhou Polytechnic College, Yangzhou,China
gyy197966@163.com

Gang Wang[1,a], Fengmei Hou[1,b], Qingling Mei[2,c]
[1]School of Information Engineering, Yangzhou Polytechnic College, Yangzhou,China
[2]Department of Computer Science, Yangzhou University, Yangzhou,China
[a]yzwg001@163.com, [b]yzhfmjs@163.com, [c]mql859@163.com

***Abstract*—In the paper the author introduces FCW_MRFI, which is a streaming data frequent item mining algorithm based on variable window. The FCW_MRFI algorithm can mine frequent item in any window of recent streaming data, whose given length is L. Meanwhile, it divides recent streaming data into several windows of variable length according to m, which is the number of the counter array. This algorithm can achieve smaller query error in recent windows, and can minimize the maximum query error in the whole recent streaming data.**

***Index Terms*—streaming data, counter array, data mining, most recent frequent item**

I. INTRODUCTION

Although there are many algorithms concerning of frequent item mining in streaming data[1,3,5], many of them don't put emphasis on current data. The existing researches of frequent item mining in the most recent streaming data are mainly algorithms based on slide window technology. L. Golab et al. introduced an algorithm based on hopping windows[2], which requires a specified a threshold 1/*m*. Recently, they introduced several algorithms utilizing slide window model. Lee and Ting[7] put forward an algorithm, which can realize space complexity O(ε^{-1}), and processing time of updating and querying O(ε^{-1}). L. Zhang and Y. Guan[6] proposed an Estimate of streaming data frequent value based on slide window, which requires a memory space O(ε^{-1}) , and the processing and querying time of each data item O(ε^{-1}). H.T.Lam,T.Calders[8] presented to mine the first K maximum frequent item in slide windows with dynamic-Change lengths. I.T.Ferry et al. proposed an algorithm which divides the most recent streaming data based on time-inclined method[4]. However, this algorithm demands that the number of counter array must be equal to the number of windows divided, which is not applicable when the number of counter array is already given.

In practical application, it is required that the querying error of recent data be relatively smaller, while errors produced by most existing algorithms are all the same for data at all times[9,10,11]. Aiming at this problem, *FCW_MRFI*, which is a streaming data frequent item mining algorithm based on variable window, is introduced in this paper. The *FCW_MRFI* algorithm can mine frequent item in any window of recent streaming data, whose given length is *L*. Meanwhile, it divides recent streaming data into several windows of variable length according to *m*, which is the number of the counter array. This algorithm can achieve smaller query error in recent windows, and can minimize the maximum query error in the whole recent streaming data. In order to compare its accuracy and recall rate with other existing methods, experiments with real data sets and synthetic data sets are conducted, which shows that *FCW_MRFI* algorithm offers much improved accuracy in data stream recent frequent item mining.

II. DEFINITION

If the maximum length of the most recent streaming data allowed to be queried is *L*, and current time is *t*, then the frequent data item in any window during the period from *t-L* to *t* can be queried. If the particular window to be queried is $w = [w_{\min}, w_{\max}]$ (as illustrated in Figure 1), in which $w_{\min}$ refers to the farthest point, while $w_{\max}$ refers to the nearest one, then it can be seen from Figure 1 that the query window *w* should satisfy: $t - w_{\min} \le L$, to wit, $t - L \le w_{\min} < w_{\max} < t$.

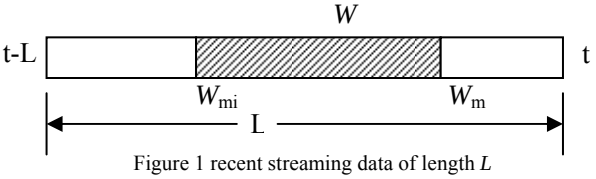


Figure 1 recent streaming data of length *L*

If the particular window to be queried is $w = [w_{\min}, w_{\max}]$, then the length of query window *w* is $|w| = w_{\max} - w_{\min}$. If is a constant defined by the system between [0,1], and if the support of a particular *x* in *w* is equal to or larger than $\phi|w|$ (), then *x* is the frequent item

doi:10.4304/jsw.9.1.258-263

of the window. We are supposed to query the frequent item designated by the user in the most recent streaming data.

If there are *m* counter units available in the system, we are supposed to perform frequent item query against any window in the most recent streaming data of length *L* using only *m* counter units, and at the same time minimize the maximum query error. For data of different time, query errors of those nearer to the current time are relatively smaller. So, the streaming data are divided into several time spans, which are called basic windows. Statistic information of item number in each basic window is stored in an array of counter units. Here Hash function is adopted to get approximate numbers, that is to say, set up a Hash function *H* with *h.count*, and set up H(x) as the counter of each data item x. If the value range of *H*(*x*) is [1, *h*], then there are *h* units. For a particular window *w* to be queried, some window span overlapping with the queried window *w* can be chosen to achieve minimum error between the length of w' which are composed of these spans and the length of *w* which are composed of the queried windows.

A simple way for this problem is to averagely allocate the most recent streaming data *L* into *m* parts, with each part of fixed length *L/m*. With this way, errors between the queried window *w* and the chosen window span w' are less than *L/m*. However, this method sets very high demands for space complexity. Moreover, it treats all data items in recent streaming data equally, which results in much loss of the newer data information. However, the newer data information is often commonly employed and tends to carry more valuable information than historical data.

Another way is to divide recent streaming data with the length of individual window span 1,2,4,…2^{l-1}. The tilted time frame method can estimate the frequency of recent data item more accurately, and decrease the accuracy of the historical data gradually. However, with this method, the error between *w* and w' can reach *L/2*, and it demands the number of counter array be equal to that of divided window span, to wit, $l = \log L = m$, which is not applicable when the value of counter array *m* is give n.

To counter the problems occurred in the two methods, a compromise, *FCW_MRFI*, is proposed in this paper. *FCW_MRFI* tries to preserve as much newer data information as possible, and at the same time minimize maximum and ensemble error.

III. Main Idea of FCW_MRFI Algorithm

3.1 If $m \le \log_2 L$

If the length of slide window is L, the number of counter array defined by the system is m, and the size of basic window is b=$\frac{L}{2^m}$, then the number basic window in current slide window is L'=L/b=2^{l-m}.

The most recent streaming data in slide window is divided into m spans by logarithmic time-inclined of length L, and the length of these spans are respectively b, b, 2b, 4b, …, 2m-2b, 2m-1b. The first window stores b data item from the basic window that comes first, and the length of the following window is twice that of the former clip. Generally, if the size of the first window is represented by w0, then, and the ith window following it is $w_i = 2^{i-1}b, \quad 0 < i < m$. One array of counter is adopted to counter the individual data item in each span, and mark them in turn as C_0, C_1, C_2, C_3, C_4,…. A buffer array, which is marked as C_{-1}, is set up by the system to receive streaming data.

Hcount is adopted to estimate the counting of the individual data item in the most recent basic windows. The counting is added to counter array C_{-1}, and the original value of C_{-1} is transferred to C_0. The counter array correspondent to the window of 2i length is marked as C_i,(i=0,1,2,…m-1), and a counter array C_{-1} is set up to receive the latest data. The counter array C-1 should transfer its data in order to receive the latest data every time when it has received b data item.

Figure 2 is taken as a simple example to illustrate how the counter array conducts *hcount* calculation and transfer operation. In figure 2, *b*=1, *m*=5, the counter arrays correspondent to the individual windows are C_{-1},C_0,C_1,C_2,C_3,C_4: , among which C_{-1} is to counter the latest data item.

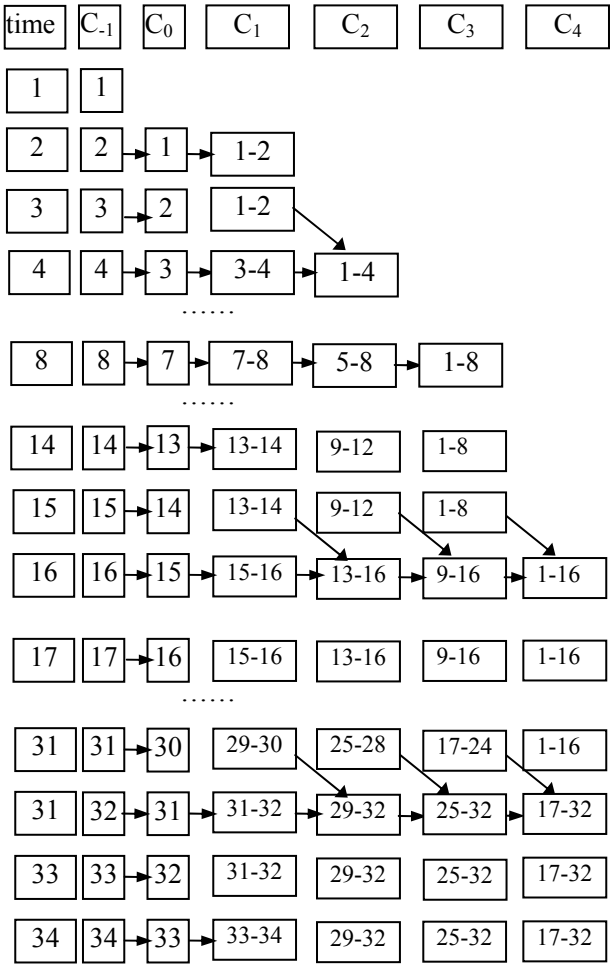


Figure 2 transfer operation f counting windows

Figure 2 shows that counter array C_{-1} always counter the latest data items, and every time when it has received b data item it perform data-transfer. Apart from transferring the values in C_{-1} to C_0, it adds the values of former several counter arrays and transfers them to the following counter arrays. The time of transferring differs in different time t. For example, when t=2, it transfers 2 times to the following counter array, when t=3, it transfers 1 time, and when t=4, it does 3 times. Generally, G(t) is marked as the location of "1" (the low-order bit is bit0), which is the lowest in binary representation of integer t. The times of transferring to be conducted by counter arrays following C_{-1} is identified by G(t). At t, the times of transferring to be conducted is $G(t)$+1. For example: if $2=(00010)_2$, then $G(2)=1$, to wit, when t=2, the times of transferring is 2. Similarly, as $G(3)=0$ $G(4)=2$, when t=3, the times to be transferred should be 1, and when t=4, the times should be 3. After summing and transferring, the correspondent counter arrays should refresh their records.

To sum up, FCW_MRFI is given below to illustrate the refresh procedure of individual counter array C_i.

As there are h units in each counter array Ci, $C_i[k]$ is adopted to represent kth unit in Ci, to wit, the counter of data whose Hash value is k. In the following algorithm, $C_i + C_j$ represents the adding up of values of the corresponding units of C_i and C_j. For example, $C_i = C_i + C_j$, represent $C_i[k] = C_i[k] + C_j[k] \quad (k = 1,2,...,h)$

Algorithm 1 *FCW_MRFI*($m \le \log_2 L = l$)

Begin

1. t=2;
2. receive and form *hcount* of the first two windows and save to C_{-1},, C_0, $C_1=C_{-1}+C_0$;
3. While not end condition do
4. $t=t+1$; $C_0=C_{-1}$;
5. form Count for newer group and store to C_{-1}; temp1=C_{-1};
6. q=min(G(t),m) /*m is the number of counter array, $G(t)$ the location of "1", which is the lowest in binary representation of integer t */
7. for j=1 to q do
8. temp2=C_j;
9. $C_j=C_{j-1}$+temp1; temp1=temp2;
10. end for j ;
11. end While ;

End

The counter array produced by the above algorithm can cover all data items in current windows, and for the newer data items, which can offer higher accuracy. If n m=4, L=16, figure 3 shows the range of the individual counter array when t=15, 16, 31, 32. As seen from above, t=15, the range is [1, 15]. As the maximum range of counter array is [1, 8], and the length is 8, the maximum query error is 4, while in range [14, 15], the query error is 0. Generally, at, the query error of range [t-1, t] is 0, and that of [t-2^i+1,t] is 2^{i-1}. One more example, when t=16, the range is [1,16]. As ranges of [1,8] can be achieved by subtracting counting value of [9,16] from that of [1,16], the maximum range is [1,8] or [9,16], and the length of both is 8, the maximum query error is 4. When t=31, the maximum range of counter in the queried windows is [17,24], and the length is 8, the maximum query error is4. When t=32, the maximum range of counter in the queried windows is [17,24] or [25-32], and the length of both is 8, the maximum query error is4. Generally, if t mod 16=g, then the range at t is $[t-16-g+1, t]$, $[t-16-g+1, t]$. The maximum query error of counters within the queried windows is l/2, and minimum query error is 0. Furthermore, for those older data ranges $|w| = (16+g)$, the query errors are greater, while for those newer data ranges, the query errors are smaller.

3.2 *If* $m > \log_2 L$

When $m > \log_2 L$, the recent streaming data L is divided into l windows of length 1, 2, 4, 8, …, 2^{l-1}. Let $m' = m - logL$, mark windows of length 2^i as S_i, and mark the corresponding counter array as C_i. As for those m' unused counter arrays, the length of them is divided according to the following principle: first, from S_{l-1}, delete S_{l-1}, add two S_{l-2}, and subtract 1from the value of m. If m' is not 0, delete one more S_{l-2}, and add two S_{l-3}. And if m' is still not 0, delete one more S_{l-2} , add two S_{l-3} . If all S_{l-2} are deleted, m' is still not 0, delete one S_{l-3}, subtract 1 from m' . The rest follows the same tend till m' is 0.

Mark the number of windows S_i as T_i. According to the method stated above, for given L and m, the maximum number of window k of window S_k which enables $T_k \neq 0$ is defined by the following rule:

Mark: $l = \log_2 L$: first calculate

$$D_i = 2^{l-i+2} - (l - i + 4) \tag{1}$$

Then the maximum number of window k is

$$k = \arg\max_i (D_i \ge m') - 1 \tag{2}$$

T_i, the number of window S_i is defined by

$$T_i = \begin{cases} 0 & i > k \\ D_{i+1} - m' + 1 & i = k \\ 2(2^{l-i-1} - T_{i+1}) - 1 & i = k-1 \\ 1 & k-2 \ge i \ge 1 \end{cases} \tag{3}$$

For example, if L=1024,m=20, l=10, then m′ =10. From (1), by calculation, $D_{10}=0$, $D_9=3$, $D_8=10$, $D_7=25$, $D_6=56$.From (2), by calculation, k=8-1=7. Still, from (3), by calculation:

$T_7=10-10+1=1$,$T_6=2(2^{10-7}-1)-1=13$,

$T_0=T_1=T_2=T_3=T_4=T_5=1$

Thus, the total number of counter array needed is $T_7+T_6+T_5+...+T_1+T_0=1+13+6=20$, which is exactly equal to the given number of counter array m. Therefore, a conclusion can be drawn as follows:

Theorem 1: with the window-arranging method mentioned above, the number of windows arranged is exactly the same as the given number of counter array *m*.

Prove : if Tk≠0 starts at k[th] layer

Total number of windows =Tk+Tk-1+....+T1+T0

$$= T_{k-1} = 2(2^{l-k} - T_k) - 1 \quad +(k-1)$$

$$= 2^{l-k+1} - T_k + k - 2$$

$$= 2^{l-k+1} - \left(C_{k+1} - m' + 1\right) - 2 + k$$

$$= 2^{l-k+1} - 2^{l-k-1+2} + \left(l - k - 1 + 4\right) + m' - 1 - 2 + k$$

$$= l - k + 3 + m' - 3 + k$$

$$= l + m' = m$$

proven

Theorem 2: with the window-arranging method mentioned above, certain given counter array can cover the query of the most recent steaming data of length *L*-1.

Prove : if Tk≠0 starts at k[th] layer

Total coverage length

$$= T_k.2^k + T_{k-1}.2^{k-1} + 2^{k-2} + ... + 2 + 1$$

$$= T_k.2^k + \left[2\left(2^{l-k} - T_k\right) - 1\right]2^{k-1} + 2^{k-1} - 1$$

$$= T_k.2^k + 2^{l-k+1+k-1} - 2^k T_k - 2^{k-1} + 2^{k-1} - 1$$

$$= 2^l - 1 = L - 1$$

proven

If the buffer array C_{-1} is included, it can cover the query of the most recent streaming data with length *L* entirely. As it is within the query range, the maximum coverage range is 2^k. Therefore, it is not difficult to draw a conclusion as follows:

Theorem 3: The window-arranging method mentioned above is a scheme that can employ as many as *m* windows to query the most recent streaming data of coverage length *L*, and can guarantee the error less than 2^{k-1}. Among which:

$$k = \arg\max_i \left[\left(m - \log L\right) \le \left(2^{l-i+2} - \left(l - i + 4\right)\right)\right] - 1$$

Thus it can be seen that the maximum query error of the counter within query windows is 2^{k-1}, and the minimum error is 0. Errors are greater for older data spans, and smaller for newer ones.

According to the window-arranging method mentioned above, m counter array can be employed to counter streaming data of each individual window correspondingly. Each time when a data comes, counter array C-1 counters it, and then transfers it. Apart from transferring the values in C-1 to C0, it adds the values of former several counter arrays and transfers them to the following counter arrays. The time of transferring differs in different time t.

For example, if the length of the most recent streaming data query window *L*=32, then $l = \log_2 L = 5$, and the number of given counter array *m*=7, then $m' = 2$. According to the window-arranging method mentioned above, the number of each individual window should be $T_{-1}=1, T_0=1, T_1=1, T_2=3, T_3=2$.

To sum up, *FCW_MRFI* is given below to illustrate the refresh procedure of individual counter array C_i.

Algorithm 2: *FCW_MRFI*($m > \log_2 L = l$)

```
Begin
 1.  t=2;
 2.  receive and form hcount of the first two windows and save to C-1,, C0, C1=C-1+C0;
 3.  While not end condition do
 4.      t=t+1; C0=C-1;
 5.      form Count for newer group and store to C-1; temp1=C-1;
 6.      q=min(G(t),k)   /*k is the number of counter array, G(t) the location of "1", which is the lowest in binary representation of integer t */
 7.      for j=1 to q do
 8.          temp2=C_j^1 ;  C_j^1=temp1; temp1=temp2;
 9.          for i=2 to T_i do
 9.              temp2= C_j^i ;  C_j^i =temp1; temp1=temp2;
10.          endfor  i
11.          temp1=temp2+C_j^(T_i-1)
10.      end for j ;
11.  end While ;
     End
```

IV. EXPERIMENTAL INVESTIGATION

Experiments with real data sets and synthetic data sets are conducted to measure *FCW_MRFI* algorithm, and compare its performance with *TiTiCount* algorithm which adopts tilted time frame method. All experiments were operated on PCs with 512M memory, 1.7G CPU, WINDOWS XP operating system, and programmed using python2.6. In experiments, parameters are set as: $\phi = 0.05$,*b*=1000.

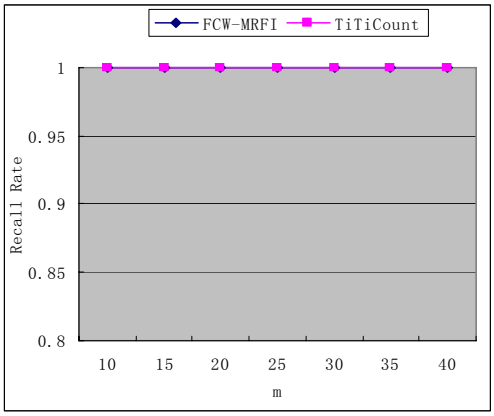
4.1 Synthetic Data

To measure this algorithm, 6 groups of data set satisfying Zipf distribution are created randomly, with parameters of each Zipf Distribution group being 0.5, 0.75, 1, 1.25, 1.5 and 1.75. The size of the data set is 1000k. When different number of counter array is given, recall rate and accuracy of *TiTiCount* and algorithm introduced in this paper are measured by query windows created randomly. Meanwhile, recall rate and accuracy of the two algorithms of different Zipf distributions are compared when *b*=1000, $\phi = 0.05$

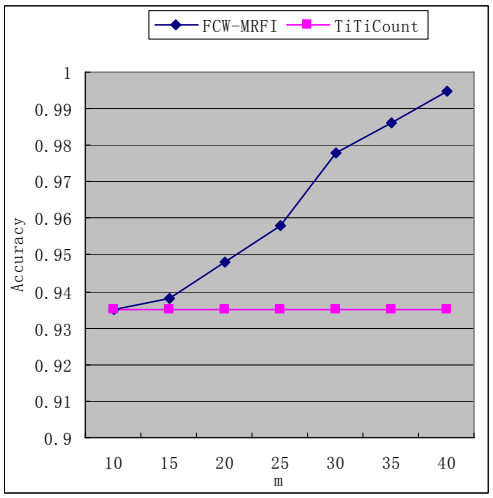
Figure 5(a) illustrates the comparison of recall rate of the two algorithms when ϕ=0.005 and Zipf distribution parameter is 1.5. It can be seen from Figure 5 that recall rate of both algorithms can reach 100% for data sets of relatively stable distribution.

Figure 3(b) illustrates the comparison of accuracy of

the two algorithms when values of *m* vary. For , as the value of *m* increases, the bigger counting window keep subdividing into smaller ones, thus guarantee more accurate frequency counting. However, the number of counting array of *TiTiCount* is stable. Therefore, as the value of *m* increases, *FCW_MRFI* is more accurate than *FCW_MRFI*.



(a) Comparison of Recall Rate

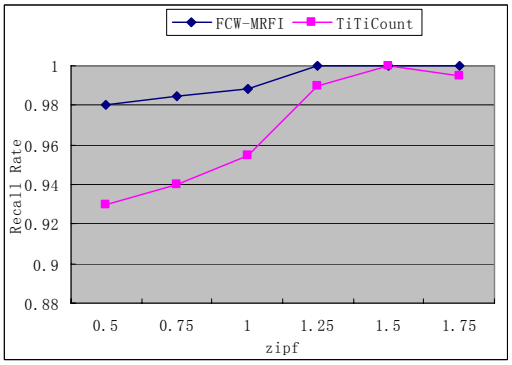


(b) Comparison of Accuracy

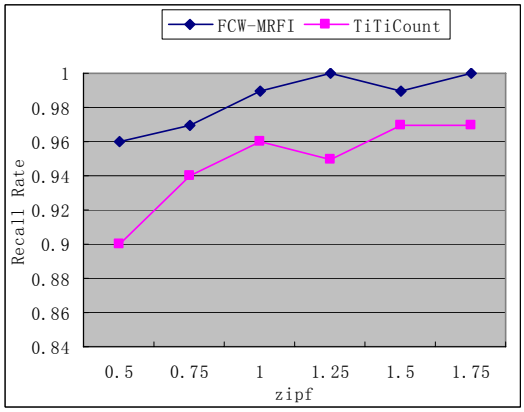
Figure 3 Comparison of Two Algorithm on Recall Rate and Accuracy when the value of *m* varies

When *m*=35, recall rate comparison for data sets of various Zip distribution parameters is illustrated in Figure 4(a). Recall rate of *FCW_MRFI* almost reach 100%, while *TiTiCount* can't reach 100% as its errors are greater in querying and counting windows.

When *m*=35, mining accuracy comparison for data sets of various Zip distribution parameters is illustrated in Figure 4(b). As can be seen from the figure, for data sets of various Zip distribution parameters, *FCW_MRFI* is more accurate than *TiTiCount*.



(a) Recall Rate Comparison



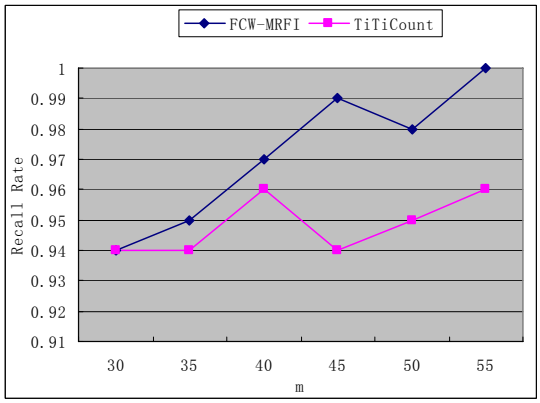
(b) Accuracy Comparison

Figure 4 recall rate and accuracy comparison for data sets of various Zip distribution parameters(ϕ =0.05,*m*=35)

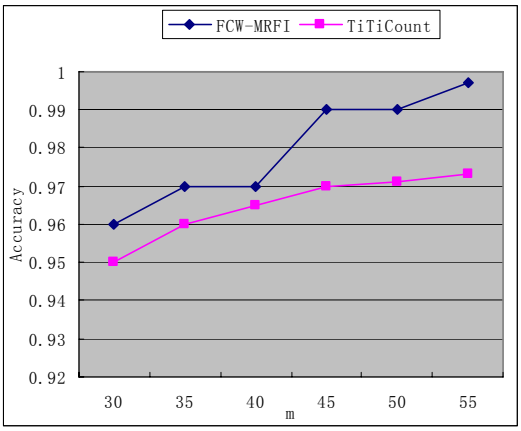
4.2 Real Data

In real data experiments, data set, kosarak[18] ,is adopted (http://fimi.cs.helsinki.fi/data/). The data set is composed of anonymous click stream of a Hungary online news gateway website, which contains about 800 million separate data items. 90 groups of query windows created randomly are adopted to compare their recall rate and accuracy.

Figure 5(a) illustrates the comparison of recall rate when the value of *m* varies. As seen from Figure 5(a), recall rate of *FCW_MRFI* is higher than that of *TiTiCount*. Figure 5(b) illustrates the comparison of accuracy when the value of *m* varies. Obviously, as the value of m increases, *FCW_MRFI* is more accurate than that of *TiTiCount*.



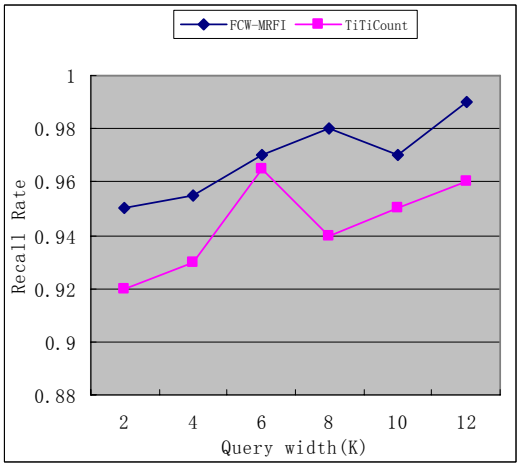
(a) Recall Rate Comparison



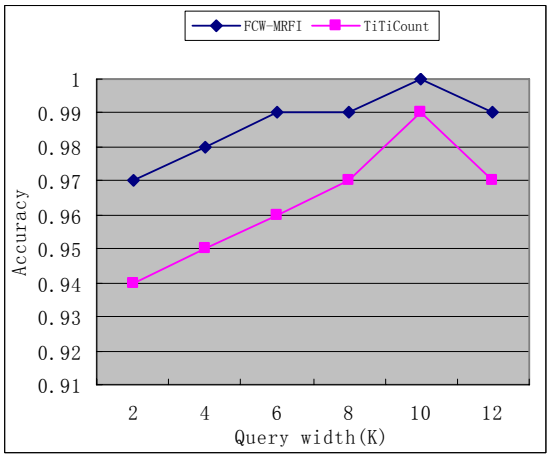
(b) Accuracy Comparison

Figure 5 Recall Rate and Accuracy Comparison when the value of *m* varies (kosarak data set, ϕ=0.05)

Figure 6(a) illustrates the recall rate comparison of two algorithms with different query width. As seen from the figure, recall rate of *FCW_MRFI* is higher than that of *t*. Figure 6(b) illustrate the accuracy comparison of the two algorithms. Results in the figure shows that *FCW_MRFI* is more accurate than *t*. Accuracy rate of *FCW_MRFI* are all over 95%, even 100%, which proves better performance of *FCW_MRFI*.



(a) Recall Rate Comparison



(b) Accuracy comparison

Figure 6 recall rate and accuracy comparison of two algorithms with different query width

V. Conclusion

FCW_MRFI, which is a streaming data frequent item mining algorithm based on variable window, is introduced in this paper. The *FCW_MRFI* algorithm can mine frequent item in any window of recent streaming data, whose given length is *L*. Meanwhile, it divides recent streaming data into several windows of variable length according to *m*, which is the number of the counter array. This algorithm can achieve smaller query error in recent windows, and can minimize the maximum query error in the whole recent streaming data. In order to compare its accuracy and recall rate with other existing methods, experiments with real data sets and synthetic data sets are conducted, which proves that *FCW_MRFI* algorithm offers much improved accuracy in recent frequent item mining in data stream.

Acknowledgements

This research was supported in part by modern education technology research project of Jiangsu province (2013-R-24925).

References

[1] J. Misra, D. Gries. Finding repeated elements[J]. Science of Computer Programming, 1982, pp.143~152.

[2] L. Golab, D. DeHaan, A. L.Ortiz, et al. Finding frequent items in sliding windows with multinomially-distributed item frequencies[C]. In Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004, pp.425~426.

[3] G. S. Manku, R. Motwani. Approximate Frequency Counts over Data Streams[C]. In Proc. of VLDB, 2002, pp.346~357.

[4] T. Calders, N. Dexters, B. Goethals. Mining Frequent Itemsets in a Stream[C]. In Proceedings of 7th IEEE International Conference on Data Mining, 2007, pp. 83~92.

[5] Frequent itemset mining dataset repository, university of helsinki (2008), http://fimi.cs.helsinki.fi/data/

[6] L. f. Zhang, Y. Guan, Frequency estimation over sliding windows[C], Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, 2008, pp.1385~1387.

[7] H. F. Li, S. Y. Lee. Mining frequent itemsets over data streams using efficient window sliding techniques[J]. Expert Systems with Applications.2009, pp.1466-1477.

[8] H. T.Lam ,T. Calders, Mining Top-K Frequent Items in a Data Stream with Flexible Sliding Windows,Copyright 2010 ACM 978-1-4503-0055-110/07

[9] C.N. Yang, and Y.Y Yang, and C.Y. Chiu, "Image Library Systems: A Novel Installment Payment for Buying Images on the Web," Journal of Computers, Vol. 20, No.1, 2009,pp. 43-49, Apr.

[10] X. Xu, J. Lin . A novel time advancing mechanism for agent-oriented supply chain simulation. Journal of Computers, Vol.4, No.12, 2009, pp.1301-1308.

[11] Ma cuixia, Meng xiangxu. Research on Object Constraints Model and Inverted Constraints in Parametric Design, Journal of Computers, Vol.23, No.9, 2000, pp.991-995.