Chapter 2
Memory-Aware Mining of Indirect Associations Over Data Streams

Wen-Yang Lin, Shun-Fa Yang and Tzung-Pei Hong

Abstract In this study, we focus on over a data stream the mining of indirect associations, a type of infrequent patterns that reveal infrequent itempairs yet highly co-occurring with a frequent itemset called "mediator". We propose a generic framework MA-GIAMS, an extension of the GIAMS framework with memory-awareness capability that can cope with the variation of available memory space, making use of most available memory to accomplish the discovery of indirect association rules without incurring too much overhead and retaining as could as possible the accuracy of discovered rules. Empirical evaluations show that our algorithm can efficiently adjust the size of the data structure without sacrificing too much the accuracy of discovered indirect association rules.

Keywords Adaptation scheme · Indirect association · Memory constraint · Resource-awareness · Stream mining

2.1 Introduction

As the advent of emerging techniques in the information explosion age, e.g., internet, handheld devices, RFID, sensor network, e-commerce, etc., more and more systems or applications would generate continuous rapid flow of vast data in a timely and endless fashion. This heralds a new type of data source, called data streams, and brings new challenges to the data mining research community. Unlike

W.-Y. Lin (✉) · S.-F. Yang · T.-P. Hong
Department of Computer Science and Information Engineering,
National University of Kaohsiung, Kaohsiung, Taiwan
e-mail: wylin@nuk.edu.tw

T.-P. Hong
e-mail: tphong@nuk.edu.tw

traditional static data sources, streaming data is fast changing, continuously generated and unbounded in amount. It is nearly impossible to store a stream entirely in a persistent storage, making contemporary mining algorithms designed for static dataset awkward and inapplicable. Recent studies on stream data mining have achieved some general requirements for effective mining algorithms [4], including single-pass of scanning over data set, real-time execution, and low memory consumption.

Not until recently, however, researchers have begun to pay attention to another critical and more challenging issue, adaptive mining of streaming data with respect to variations in available resources, e.g., CPU power and memory space. Any mining algorithm running on these resources constrained devices has to be able to adjust and adapt its execution to utilize the very limited, changing in availability resources. Contemporary research work on resource-aware mining over data streams can be viewed from two aspects, the type of resources under concern, e.g., CPU power [3] or memory space [5], and the type of mining tasks employed, e.g., clustering [10], density estimation [8], frequent itemset mining [3, 13], etc.

From the viewpoint of data mining task, almost all works were focusing on frequent pattern discovering; no work, to our knowledge, has been devoted to the problem of mining infrequent patterns, e.g., indirect associations. Another minor restriction of the previous works is that they were conducted with respect to some specific stream window model, e.g., landmark, time-fading, or sliding window. Algorithms tailored for some particular window model usually are not applicable to and so need redesigning to fit to other models.

In this study, we aim at developing a generic mining framework with memory-aware capability that can adapt the computation in accordance with data arriving rate as well as the available memory space. Specifically, we focus on the mining of indirect associations [11], a type of infrequent patterns that reveal infrequent itempairs yet highly co-occurring with a frequent itemset called "mediator". For example, Coca-cola and Pepsi are competitive products and could be replaced by each other. So it is very likely that there is an indirect association rule revealing that consumers buy a kind of cookie tend to buy together with either Coca-cola or Pepsi but not both, denoted as ⟨Coca-cola, Pepsi | cookie⟩. Below is a formal definition of indirect association.

Definition 1 An itempair {a, b} is indirectly associated via a mediator M, denoted as ⟨a, b | M⟩ if the following conditions hold:

1. $\text{sup}(\{a, b\}) < \sigma_s$ (Itempair support condition);
2. $\text{sup}(\{a\} \cup M) \geq \sigma_f$ and $\text{sup}(\{b\} \cup M) \geq \sigma_f$ (Mediator support condition);
3. $\text{dep}(\{a\}, M) \geq \sigma_d$ and $\text{dep}(\{b\}, M) \geq \sigma_d$ (Mediator dependence condition); where sup(A) denotes the support of an itemset A, and dep(P, Q) is a measure of the dependence between itemsets P and Q.

We propose a generic framework MA-GIAMS with memory-awareness capability, an extension of our previously proposed GIAMS framework [7] that can represent all classical streaming models and retain user flexibility in defining new models. Our MA-GIAMS framework can cope with the variation of available memory space, making use of most available space to accomplish the discovery of indirect association rules without too much overhead and retaining as could as possible the accuracy of discovered rules. To realize the memory awareness scheme, we propose a victim searching and node releasing algorithm to adjust the structure for maintaining potential frequent itemsets in accordance with the available memory space. Empirical evaluations on a real dataset show our algorithm can efficiently adjust the size of the structure without sacrificing too much the accuracy of discovered indirect association rules.

The remainder of this paper is organized as follows. In Sect. 2.2, we provide some background knowledge and related work about resource-aware stream mining. Since our work in this study is based on the GIAMS framework, we give an overview of GIAMS in Sect. 2.3. Section 2.4 describes our proposed MA-GIAMS framework, including the algorithmic design for adaptive functionalities with respect to memory space variation, and the detailed data structure and procedure for realizing the algorithmic framework. Section 2.5 describes the experiments we conducted on evaluating the proposed framework. Finally, the conclusions and future works are presented in Sect. 2.6.

2.2 Related Work

The research of resource-aware stream mining focuses mainly on how to use the limited resources to efficiently accomplish the mining task while guarantee as could as possible the accuracy of mining results. Contemporary research work on resource-aware mining over data streams can be viewed from three aspects, the type of resources under concern, the type of mining tasks employed, and the type of adaptation techniques used.

Studies conducted from the viewpoint of resource type consider CPU power and memory space, or additional issues for mobile devices, battery and network bandwidth. The types of mining tasks studied include clustering, frequent pattern mining, kernel density estimation. Finally, the adaptation techniques can be input adaptation or algorithm adaptation. The input adaptation technique refers to schemes used in adjusting the amount of input data to keep up with the pace of the stream and meet the computing capacity. Three commonly used approaches including sampling, i.e., statistically choosing some input data, load shedding, i.e., discarding part of the input data, and data synopsis creation, referring to data summarization techniques that retain the data characteristics, statistics or profile.

Teng et al. [13] proposed a wavelet based data synopsis technique, called Resource-Aware Mining for Data Streams (RAM-DS), to transform the input data stream into different granularity of data from temporal view or frequency view in order to reduce the data amount. Their work focused on frequent pattern mining.

The group led by M.M. Gaber is one of the pioneers in resource-aware stream data mining. They have conducted a series of research studies on resourceawarestream clustering [5, 6], with ultimate goal at developing a general resource-aware framework that can adapt to variability of different resource availability over time. Their recent work [11] proposed a generic framework called Algorithm Granularity Settings, which uses three levels of adaptation strategies in the data input, algorithm processing, and data output, and cope with different issues of resource-awareness.

The work conducted by Dang et al. [2, 3] considers stream mining under CPU resource constraint, focusing on frequent pattern discovery, and using load shedding technique. Their approach relies on a way to estimate the system workload by approximately computing the number of maximal itemsets that can be generated from a transaction, and then employs the load shedding technique to sample transactions if the system workload is over the current CPU computing power.

Heinz and Seeger [8] proposed an algorithm adaptation method that is tailored to the problem of kernel density estimation. The resource type considered in their work is memory space.

In distributed computing environment, the network bandwidth available for data transmission is particularly important. Parthasarathy and Subramonian [9] developed a resource-aware scheduling scheme to cope with the bandwidth limitation.

2.3 Overview of GIAMS

Generic Indirect Association Mining over Streams (GIAMS) [7] is an algorithmic framework that can accommodates the three stream window model called Landmark model, Time-fading model and Sliding window, while retains user flexibility for defining new models. Users only have to set four variables (timestamp, window size, stride and decay rate) in accordance with the generic window model, then GIAMS can discover indirect association rules.

Suppose that we have a data stream $S = (t_0, t_1, t_{2,\ldots}t_i,\ldots)$, where t_i denotes the transaction arrived at time i. Since data stream is a continuous and unlimited incoming data along with time, a window W usually is specified, representing the sequence of data arrived from t_i to t_j, denoted as $W[i, j] = (t_i, t_{i+1},\ldots, t_j)$. GIAMS adopts a generic window model Ψ for data stream mining, which is dictated as a four-tuple specification, $\Psi(l, w, s, d)$, where l denotes the timestamp at which the window start, w as the window size, s is the stride the window moves forward, and d is the decay rate. The stride notation s is introduced to allow the window moving forward in a batch (block) of transactions (of size s). In this regard, the stride notation s also specifies the size of block.

The GIAMS framework is developed according to the paradigm proposed by Tan et al. [12]: First, discovers the set of frequent itemsets with support higher than σ_f, and then generates the set of qualified indirect associations from the frequent itemsets. Based on this paradigm, GIAMS works in the following scenario: (1) The user first sets the streaming window model by specifying the parameters described

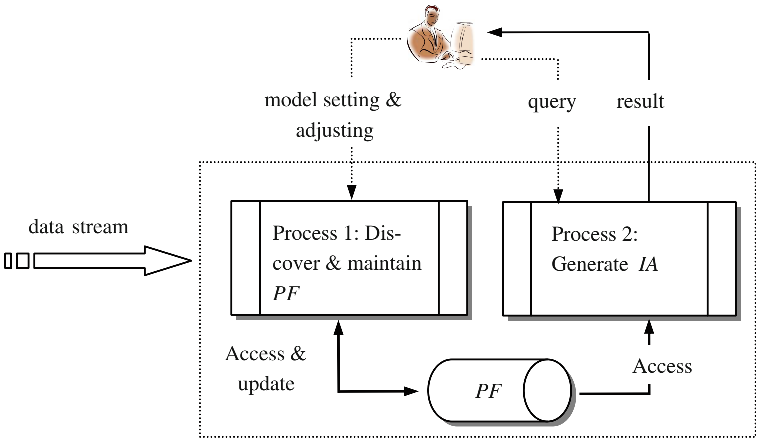

Fig. 2.1 The GIAMS generic framework for indirect association mining

previously; (2) The framework then executes the process for discovering and maintaining the set of potential frequent itemsets *PF* as the data continuously stream in; and (3) At any moment once the user issues a query about the current indirect associations the second process for generating the qualified indirect associations is executed to generate from *PF* the set of indirect associations *IA*. Figure 2.1 depicts the generic streaming framework for indirect associations mining.

2.4 Adaptation Scheme for Available Memory Awareness

2.4.1 Data Structure

The structure used in our MA-GIAMS for realizing *PF* is a modification of the tree structure used in GIAMS, called *Card-Stree*, which is a forest of search trees keeping itemsets of different cardinalities, appearing in the current window, say ST_1, ST_2, …, ST_k, for ST_k maintaining the set of frequent *k*-itemsets. We name the modified structure *Card-Stree**. Each node in *Card-Stree** except the root keep the information of the maintained itemset. More specifically, for each itemset *X*, the node records *X.id*, the identifier of *X*; *X.bidv*, the vector of identifiers of the blocks that *X* appears; *X.countv*, the vector that stores the number of occurrences of *X* within each block; and *X.tlcount*, the total occurrences that *X* appears in the current window. An example of *Card-Stree** after processing the example data stream shown in Fig. 2.2 is illustrated in Fig. 2.3. Because each node consumes approximately the same amount of memory space, in what follows we use a node as the memory unit.

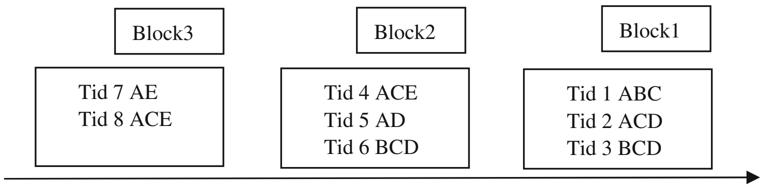

Fig. 2.2 An example data stream

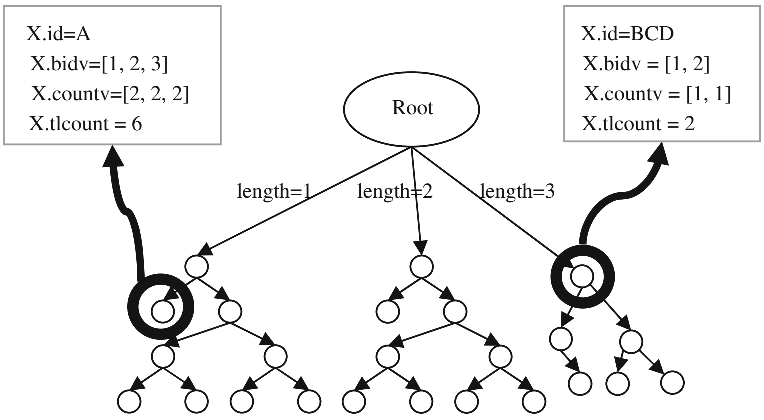

Fig. 2.3 An illustration of *Card-Stree**

2.4.2 Adaptive Node Releasing

A simple and intuitive approach is blindly dropping some itemsets while the memory space is not enough. This approach, however, would loss too much information, making the mining results incorrect and leading to wrong analysis. Rather, we employ a strategy similar to the concept of cache replacement. When the memory space is insufficient, we decide which itemsets in the current *Card-Stree** are less important and can be deleted to release enough space for accommodating the incoming, more important itemsets.

In this regard, we propose a node releasing mechanism to cope with the situation when memory space is not enough to maintain all of potential frequent itemsets in the *Card-tree**. Note that the processing of stream mining needs to be computed in real time. As such, the main design concern is the efficiency, i.e., how to efficiently search and determine the victim nodes for deletion, without sacrificing too much the accuracy of the discovered rules.

Firstly, we note that for mining indirect associations, the set of 2-itemsets is the most important set, because from which all length-2 mediators and the infrequent itempairs are generated. Our node replacement thus is executed only when the memory space is not enough and the new generated itemsets from the incoming transaction are of lengths 1 and 2. In other words, those new generated k-itemsets with $k > 2$ are discarded immediately when the memory is not enough.

Secondly, since the frequency of long itemsets is usually less than that of short itemsets, and lengthy rules constructed from long itemsets are less understandable to the users, our approach replaces nodes according to their cardinalities, first choosing the longest itemsets. More precisely, suppose that we need to release n nodes and let k denote the largest length of itemsets in *Card-Stree**. Our approach will search for the top n nodes with the smallest counts in the ST_k subtree. If there are less than n nodes found in ST_k, then the search continues in subtrees ST_{k-1}, ST_{k-2}, and so on. However, during the search process, we will delete any node whose count is equal to 1 and decrement the number of nodes to be released. This is because these nodes represent the least occurring itemsets.

In addition, to speedup the search of n victim nodes, we introduce a link structure called *victim-list* to maintain the nodes in *Card-Stree** chosen for deletion. Each node in *victim-list* contains three fields, the count of the itemsets, the number of itemsets having this count, and pointers to all the corresponding nodes in the *Card-Stree**. All nodes in *victim-list* are sorted in decreasing order of counts.

There are some remarks noticeable in the aforementioned procedure. First, the victims are maintained and deleted in groups, distinguishing by count. That is why we may end up with more than n victims in *victim-list*. Second, for efficiency concern, we only compare the count of the node X under concern with that of the first group in *victim-list*. This avoids the overhead for linear searching along the entire *victim-list*.

2.4.3 Algorithm Description

We only brief the modification to the Process 1 in GIAMS as shown in Fig. 2.1; there is no change to Process 2. In summary, for each k-itemset X generated from the input block of transactions, if X is already in *Card-Stree**, we update its information. Otherwise, if the memory space is enough, then insert X into *Card-Stree**. On the other hand, if the memory is not enough and $k \geq 3$, then X is discarded. But for $k < 3$, we temporarily store X into a buffer. After all itemsets generated from the current transaction have been inspected, Process 1 will call algorithm Victim_Searching&Releasing described in Sect 2.4.2 to release memory, and insert at most #*victim* of the new generated k-itemsets in buffer into *Card-Stree**.

2.5 Experimental Results

To evaluate the effectiveness and efficiency of MA-GIAMS, we conducted a preliminary experiment on a real dataset msnbc [1], which was constructed from the web log of news pages in msn.com for the entire day of September, 28, 1999,

Table 2.1 Parameter settings for generic window model used in this experiment

s	w	d	σ_s	σ_f	σ_d
10,000	80,000	1	0.01	0.01	0.1

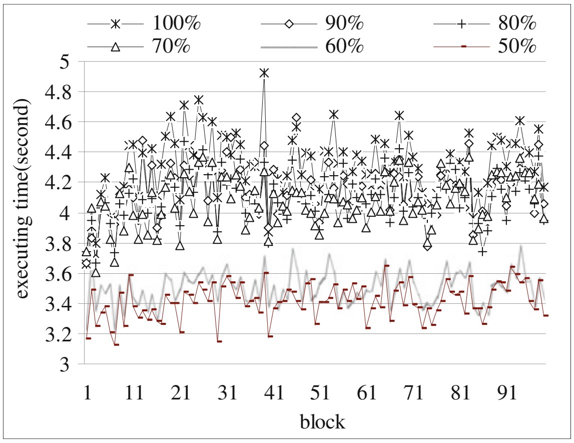

Fig. 2.4 Execution times of MA-GIAMS running over msnbc with available memory variation

which consists of 989818 transactions. More detailed description of this dataset can be found in [1]. The evaluation was inspected from two aspects, execution time and pattern accuracy. In this evaluation, we consider the sliding window model, with detailed settings of the generic model shown in Table 2.1. Since $s = 10{,}000$, the test set was divided into 99 blocks. All experiments were done on Intel(R) Core(TM) i5-2400(3.1G) PC with 4 GB of main memory, running the Windows 7 32-bit operation system. All programs were implemented in Visual C++ 2008.

To inspect the performance and effectiveness of our memory awareness adaptation scheme, we run MA-GIAMS under five settings of available memory, i.e., ranging from 90 to 50 % of the original memory space, with 10 % decrement. And compare the results with those running with sufficient memory space.

First, we evaluate the execution times of MA-GIAMS. The results are depicted in Fig. 2.4, where x-axis denotes the block number. As the results demonstrate, most of the time MA-GIAMS is faster than GIAMS even MA-GIAMS incurs overhead for running victim searching and node releasing to cope with insufficient memory. And the execution times increase as the available memory increase. This is because when the memory is not sufficient there are certain amount of itemsets that originally have to be maintained in the *Card-Stree** are pruned, which has the effect in decreasing the number of create and update operations, without doing too much of node replacement operations.

Next, we inspect the accuracy of the results generated by MA-GIAMS. Because our algorithm introduces storage shedder to prune itemsets to be maintained in the *Card-Stree** structure, so error may occur to the discovered frequent itemsets, including the missing rate and support missing rate of frequent itemsets, and the precision and recall with respect to indirect association rules.

We first evaluate how much of frequent itemsets generated by GIAMS without memory limitation will be lost when memory shortage occurs, which is measured as error rate:

$$Miss = |F_{\text{true}} \cap F_{\text{est}}| \,/\, |F_{\text{true}}| \tag{2.1}$$

where F_{true} denotes the set of frequent itemsets discovered by GIAMS while F_{est} represents that by MA-GIAMS with insufficient memory. We also check the difference between the supports of the discovered frequent itemsets with and without memory limitation, which is measured by the following formula called Average Support Error (ASE):

$$ASE = \frac{\sum_{x \in F} (Tsup(x) - Esup(x))}{|F|} \tag{2.2}$$

where *Tsup* denotes the frequent itemsets that are discovered by GIAMS, *Esup* denotes the frequent itemsets discovered by MA-GIAMS under memory limitation. As the results displayed in Fig. 2.5, the missing rate of frequent itemsets generated by our MA-GIMAS are nearly zero for sufficient memory and are less than 0.2 even the available memory is very restricted, e.g., 60 and 50 %. All ASEs are nearly zero in all cases, even in the case that only 50 % of memory is available.

We then examine the accuracy of rules discovered by MA-GIAMS. We consider two measurements, *precision* and *recall*. Let IA_{true} denote the set of indirect associations discovered by GIAMS without memory limitation and IA_{est} denote the set discovered by MA-GIAMS with insufficient memory. The precision measures the ratio of how many indirect associations in IA_{est} are also in IA_{true}, while recall examines the percentage of how many indirect associations in IA_{true} are missed generated by MA-GIAMS. These two criteria are define as follows:

$$Precision = |IA_{\text{true}} \cap IA_{\text{est}}| \,/\, |IA_{\text{est}}| \tag{2.3}$$

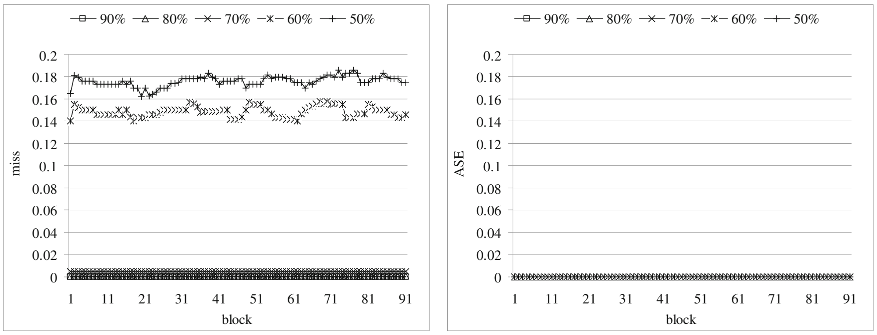

Fig. 2.5 Miss and ASE of MA-GIAMS over msnbc with available memory variation

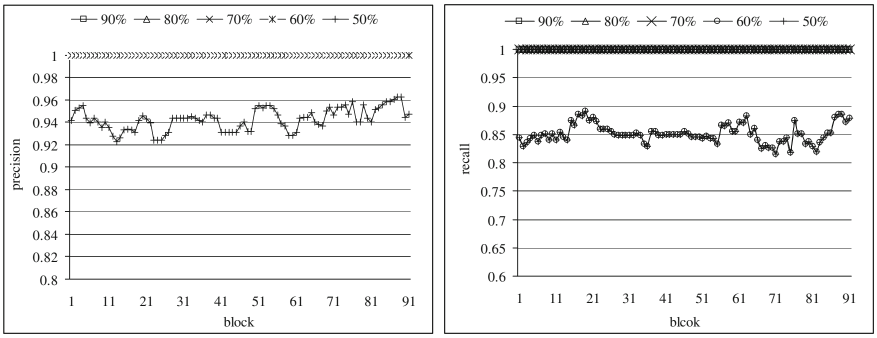

Fig. 2.6 Precisions and recalls of MA-GIAMS in terms of discovered indirect associations from msnbc with available memory variation

$$Recall = |IA_{\text{true}} \cap IA_{\text{est}}| \,/\, |IA_{\text{true}}| \tag{2.4}$$

As the results illustrated in Fig. 2.6, the memory adaptation scheme of our MA-GIAMS performs very well. All of the precisions are larger than 0.9 and recalls are above 0.8, meaning the percentages of false indirect associations discovered by our MA-GIAMS are less than 10 % and the percentages of true indirect associations not discovered by MA-GIAMS are less than 20 %, respectively, even in the case that only 50 % of memory is available.

2.6 Conclusions

In this paper, we have considered the problem of memory-aware mining of indirect association rules over data streams. We have proposed a generic framework MA-GIAMS to cope with this problem. Our proposed framework is based on GIAMS, a mining framework that can accommodate most of the contemporary stream window models and allow user-defined specific window models. Our framework add some mechanisms to GIAMS, including a resource monitor, a load shedder, and a storage shedder, as a whole can adapt the computation in accordance with data arriving rate as well as the available memory space. We have conducted a preliminary experiment to evaluate the effectiveness and performance of the proposed framework. The experimental results showed that our framework can effectively adjust the memory consumption during the course of frequent pattern mining with very little overhead. The results also showed that even with limited memory space, i.e., only 50 % of the original space, our framework not only can discover most of the frequent patterns serving as mediators for qualified indirect associations but also maintain the accuracy of discovered patterns. More experiments on over types of real datasets as well as synthetic datasets will be performed in the near future.

The study of resource-aware mining from streaming data is still in its infancy. Many research issues are worthy of further investigation. In this study, we confine the resource to memory space. Many applications developed in mobile environment, such as sensor networks, intelligent cell phones, however, have to consider additional resources constraint, mainly battery and network bandwidth. We will extend our framework to accommodate these new types of resources.

Acknowledgments This work is partially supported by National Science Council of Taiwan under grant No. NSC97-2221-E-390-016-MY2.

References

1. Cadez I, Heckerman D, Meek C, Smyth P, White S (2000) Visualization of navigation patterns on a web site using model-based clustering. In: 6th ACM SIGKDD international conference on knowledge discovery and data mining, pp 280–284
2. Dang XH, Ng WK, Ong KL (2006) Adaptive load shedding for mining frequent patterns from data streams. In: International conference on data warehousing and knowledge discovery, pp 342–351
3. Dang XH, Ng WK, Ong KL, Lee VCS (2007) Discovering frequent sets from data streams with CPU constraint. In: 6th Australasian data mining conference, pp 121–128
4. Domingos P, Hulten G (2003) A general framework for mining massive data streams. J Comp Graph Stat 12(4):945–949
5. Gaber MM, Krishnaswamy S, Zaslavsky A (2005) Resource-aware mining of data streams. J Univers Comp Sci 11(8):1440–1453
6. Gaber MM, Yu PS (2006) A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering. In: ACM symposium on applied computing, pp 649–656
7. Lin WY, Wei YE, Chen CH (2011) A generic approach for mining indirect association rules in data streams. In: 24th international conference on industrial, engineering and other applications of applied intelligent systems, pp 95–104
8. Heinz C, Seeger B (2008) Cluster kernels: resource-aware kernel density estimators over streaming data. IEEE Trans Knowl Data Eng 20(7):880–893
9. Parthasarathy S, Subramonian R (2001) An interactive resource-aware framework for distributed data mining. IEEE technical committee on distributed processing letters, pp 24–32
10. Shah R, Krishnaswamy S, Gaber MM (2005) Resource-aware very fast K-means for ubiquitous data stream mining. In: 2nd international workshop on knowledge discovery in data streams, pp 40–50
11. Tan PN, Kumar V, Srivastava J (2000) Indirect association: mining higher order dependencies in data. In: 4th European conference on principles of data mining and knowledge discovery, pp 632–637
12. Tan PN, Kumar V (2001) Mining indirect associations in web data. In: 3rd international workshop on mining web log data across all customers touch points, pp 145–166
13. Teng WG, Chen MS, Yu PS (2004) Resource-aware mining with variable granularities in data streams. In: 5th SIAM conference on data mining, pp 22–24