Clustering XML Data Streams by Structure based on Sliding Windows and Exponential Histograms

Mingxia Gao
College of Computer Science
Beijing University of Technology
Beijing, China
gaomx@bjut.edu.cn

Furong Cheng
R&D Center TravelSky
Technology Limited
Beijing, China
bjcfr@163.com

Abstract—**To group online XML data streams by structure, this paper introduces an algorithm named the CXDSS-SWEH. It is a dynamic clustering algorithm based on sliding windows and exponential histograms. Firstly, the algorithm formalizes an XML document into a structure synopsis named Temporal Cluster Feature for XML Structure (TCFXS). Secondly, it allots the TCFXS to some cluster through measuring similarities between the TCFXS and each existing cluster. At last, updating clusters in sliding windows are real-time modified through criterions of false positive exponential histograms. We have conducted a series of experiments involving real and simulative XML data streams for validating empirical effects on clustering quality, memory and time consumption. Our experimental results have confirmed: (1) clustering quality of the CXDSS-SWEH is close to the methods XCLS and SW-XSCLS; (2) memory and time consumption of the CXDSS-SWEH are efficient and effective, compared to the SW-XSCLS.**

Keywords-**XML data stream; temporal cluster feature**

I. INTRODUCTION

The eXtensible Markup Language (XML) [1] is a self-description language used for data exchange and sharing. It has become more prevalent on the Web after recommended by W3C in February 1998 [8]. A lot of applications and services produce huge amounts of online XML data streams. Examples abound from online network monitoring to stock market updates. To analyze this category of data, many researchers focus on clustering XML data and propose a number of algorithms [3]–[12]. However, existing clustering methods mainly focus on static XML data and, generally, need to scan data many times. Though technologies of mining XML data have recently been extended to XML data streams [5], [7], [10], [11]. Most of methods query XML streams through different methods [5], [7], [10].

This paper proposes a dynamic algorithm named the CXDSS-SWEH to cluster online XML data streams by structure. It uses technologies of sliding windows and exponential histograms. Firstly, the algorithm formalizes an XML document into a structure synopsis named Temporal Cluster Feature for XML Structure (TCFXS). Secondly, it allots the TCFXS to some cluster through evaluating similarities between the TCFXS and each existing cluster. In fact, each existing cluster in sliding windows is a team of TCFXSs, which must satisfy criterions of false positive exponential histograms. At last, updating clusters in sliding windows are real-time modified through merging, deleting and so on. To validate empirical effects, we have conducted a series of experiments involving real and simulative XML data and compared with the static clustering method XCLS in the literature [8] and the stream clustering method SW-XSCLS in the literature [11] and accepted some promising results.

The remainder of this paper is organized as follows. Section 2 surveys the related work. Section 3 describes the problems and basic ideas. Section 4 presents a structure synopsis and a method of computing similarity between two synopses. Section 5 discusses the online clusters maintained in sliding windows. Section 6 provides experimental data, design, and results. Section 7 concludes the paper.

II. RELATED WORK

Existing technologies of clustering XML data mainly focus on static data. A great lots of methods for computing similarities between XML documents have been developed, ranging from various tree edit distance methods [3], [9], [12] to direct extracting document feature approaches [4], [6], [8].

The basic idea in all of these tree edit distance algorithms is to find the minimum cost of transforming a tree to another tree by using edit operators. A key differentiator of most of algorithms is the set of edit operations allowed or the structure of trees. For example, Theodore et al. [3] exploited the tree nature of XML documents and provided techniques for tree matching, merging and pruning. Nierman and Jagadish [9] proposed a method of edit distance including five kinds of operators. Zheng et al. [12] described an XED distance to evaluate difference between documents.

Different from edit distance methods, many approaches use other kinds of features to present XML documents, then directly measure similarities between features. Flesca [4] showed structure of XML documents into a time serial, appearance of every tag into an impulse and computed similarities between documents by the frequency domain of Fourier transform. Wang et al. [6] built an S-GRACE based on structure of documents and proposed a method of computing similarities between documents by graph matching. Nayak [8] modelled structure of XML documents into a Level Structure (LF), which can be seen as a simple ordered labelled tree and proposed a method of directly computing similarities between LSs.

The above methods generally need to scan and parse data many times. But, XML data streams only are permitted to scan or parse once. So, these static methods are not directly suitable for XML data streams.

Recently, technologies of mining XML documents has been successfully extended to XML data streams. But, these methods mainly focus on query fields. Koch and Scherzinger [5] introduced the notion of XML Stream Attribute Grammars (XSAGs), which is the first scalable query language for XML data streams. Yang et al. [10] built a SAX-based XML data streams query evaluation system and designed an algorithm that consumes buffers in line with the concurrency lower bound. Mayorga et al. [7] presented a method for building a stream synopsis to approximately query XML streams.

Mao et al. [11] proposed a clustering stream method named the SW-XSCLS. It extended LS features in the literature [8] to present stream synopses, and used sliding windows to maintain features as the CXDSS-SWEH in this paper. However, two methods have important differences as follows. (1) Methods of computing similarities are different. The SW-XSCLS uses the original method in the literature [8]. The CXDSS-SWEH proposes a new feature named Node List and a method of computing similarities between features. (2) Steps of combining two clusters are different. The SW-XSCLS uses same steps as those in the literature [2] and the CXDSS-SWEH presents a new method to save usable memory.

III. Problem Statement

A. XML Data Stream

An XML data stream in query fields often is defined as a massive, continuous sequence of tokens in XML documents. Among, tokens, respectively, denote beginning tokens, ending tokens of elements, and actual values of elements or attributes. The paper focuses on clustering structure between XML documents. So, an XML data stream is defined as a massive, continuous sequence of documents, as shown in definition 1.

Definition 1: Let $S = \{X_1, \ldots, X_i, \ldots\}$ be an XML data stream, and $\{< T_1, E_1 >, \ldots, < T_i, E_i >, \ldots\}$ be time stamps of these documents. T_i and E_i, respectively, denote the timestamps of document beginning and ending, where $i < m$, such that $T_i < T_m$, $E_i < E_m$. X_t is a sequence of tokens produced by parsing an XML document based on SAX.

B. Clustering XML Data Streams by Structure

Important differences between data streams and traditional data sets are that arriving data units are massive, continuous and infinite. To adapt to these characteristics, features to represent and analyze these data general use approximate formats. For semi-structured XML documents, approximate formats can be structure, content, or structure+content. The clustering method in this paper only focuses on structure synopses and omits all content information. It defines a Temporal Cluster Feature for XML Structure (TCFXS) as a structure synopsis. The detailed information about TCFXSs will be addressed in Section 4.

A sliding window mode is a better method of solving massive or infinite units in data streams. It only considers and deals with nearly arriving units at any time. It emphasizes that importance of units in a stream will wear off with time. To real-time maintain units in sliding windows, new added units and overdue units must be managed timely. The technology of exponential histograms is one of methods to manage data synopses in sliding windows. This paper uses the online method in the literature [2] for reference and defines an Exponential Histogram of Temporal Cluster Feature for XML Structure (EHTCFXS) to manage structure features in sliding windows. An EHTCFXS is a team of TCFXSs based on criterions of false positive exponential histograms. The detailed definition and maintaining algorithm will be addressed in Section 5.

Based on the above idea, clustering XML data streams by structure based on sliding windows and exponential histograms can be shown in definition 2. The definition tells us that there are three key problems must be solved to cluster online XML data streams, including of building TCFXSs, measuring similarities between TCFXSs, and maintaining EHTCFXSs in sliding windows timely.

Definition 2: Given an XML data stream in a sliding window at time t, the clustering solution, denoted by $C = \{C_1, C_2, \ldots, C_q\}$, is a partition of n XML documents, where C_i can be represented by an EHTCFXS. Among, $EHTCFXS = \{TCFXS_0, \ldots, TCFXS_i\}$; n is the number of XML documents in the stream; q is the number of clusters. The partition must satisfy two following rules: (1) $TCFXS(C_1 \cup C_2 \cup \ldots \cup C_q) = TCFXS(X_i \cup X_{i+1} \cup \ldots \cup X_{i+n-1})$ and (2) $TCFXS(C_i) \cap TCFXS(C_j) \neq TCFXS(X_i)$.

IV. Temporal Cluster Features

A. Temporal Cluster Features

Heterogeneous XML documents are basic units of XML documents streams. These XML documents with different structure and contents imply complex hierarchy and semantic information. To extract cluster features by structure, many secondary information can be overlooked. This paper defines a structure feature, called Node List, for a set of XML documents. A Node List is a list of pairs (node code and level value). They, respectively, present distinct elements in documents and the hierarchy of each element in own document.

A distinct integer in a Node List replaces with the name of each distinct element in XML data streams. It denotes the first appearance order for the start event of a element in a start events stream (only recording start events) in a sliding window. Figure 1-(a) presents an example of a sequence of tokens parsed through SAX and the order of these start events, where the sequence of tokens is the first document arriving at sliding windows. The coding method can ensure that elements with same name in different documents use a same integer. A global name-index list in sliding windows will be defined and used to code names of distinct elements in XML data streams. It consists of pairs (element name and coding integer), as shown in Figure 1-(b). The coding integer of an element name will be acquired by searching the global name-index list. The corresponding integer is its coding integer when finding the element name from the global name-index list; otherwise existing max value plus one is its coding integer. At same time, a pair (the element name and the coding integer) is inserted in the name-index list as a new node. The level values of Node Lists can be acquired by a runtime-stack technology in the literature [13]. Figure 1-(c) gives an example to illustrate the process of acquiring level values by a runtime-stack. The start event of each document activates a null stack. Start events and end events of elements, respectively, inspire pushing and popping. The level value of an element just is the corresponding pointer value of the element in the stack. Figure 1-(d) presents the Node List formalized by the XML document in Figure 1-(a).

Definition 3: Given an XML data stream $S = \{X_1, \ldots, X_n\}$, let $(NodeList_{1\rightarrow n}, n, t)$ be a Temporal Cluster Feature for XML Structure (for short TCFXS(S)), where $NodeList_{1\rightarrow n} = \sum_{i=1}^{n} NodeList_i = NodeList(X_1 \cup X_2 \cup \ldots \cup X_n)$; n is the number of XML documents included in S; t is the time stamp T_n of the newest arriving XML document in S.

Two Node Lists can be combined by property 1 on basis of the above definition of Node Lists. The actual operating steps include comparing ordered integers, and inserting a new node. Figure 2 presents a schematic illustration of combining two Node Lists. The combining result only contains one copy of integer 1 and 2 because of they being a same level, and contains two copies of integer 3 and 5 with different level values. These integers in the combining Node List still satisfy partial order. The definition 3 is a Temporal Cluster Feature for XML Structure on basis of Node Lists. Two TCFXSs without time overlap in an XML data stream also are combined by property 2.

Property 1: Given two Node Lists, the result of combining them can be acquired by uniting all elements of every Node List, such that repeating elements of same level value and same code only keeps one copy.

Property 2: Given two TCFXSs without time overlap TS_1 and TS_2, the result of jointing two TCFXSs TS_3 can be acquired by combining two Node Lists, adding two number of documents, and maximizing two time stamps as following:
$TS_3.NodeList = TS_1.NodeList \cup TS_2.NodeList$,
$TS_3.n = TS_1.n + TS_2.n$,
$TS_3.t = Max\{TS_1.t, TS_2.t\}$.

B. Similarities between Two TCFXSs

The equation (1) is defined to compute similarities between two Node Lists. Its range is [0,1], among 1 denotes that two Node Lists are same, otherwise 0 denotes that two Node Lists have not any common element. The comparing progress of two Node Lists accords with the comparing progress of ordered integers. The progress can be described as the following: (1) node values are basic moving units; (2) if a node value in Node List 1 is less or equal to that in Node List 2, the Node List 1 is moved to its next node value and the progress continues, otherwise, the Node List 2 is moved to its next node value and the progress continues. Two Node Lists cannot back up in the comparing progress. So, the time complexity is $O(max\{N_1, N_2\})$ in the worst state, where N_1 and N_2, respectively, represent the total number of all elements in Node List 1 and Node List 2. As definition 3, at time t, structure synopses in a sliding window are TCFXSs including Node Lists. So, the similarity between two TCFXSs is equal to the similarity between corresponding Node Lists in the two TCFXSs.

$$\begin{aligned} NodeSim_{1\leftrightarrow 2} &= \frac{ComWeight_1 + ComWeight_2}{ObjWeight_1 + ObjWeight_2} \\ &= \frac{\sum_{i=1}^{M_1}(1/r)^{L_1^i} + \sum_{j=1}^{M_2}(1/r)^{L_2^j}}{\sum_{k=1}^{N_1}(1/r)^{L_1^k} + \sum_{k=1}^{N_2}(1/r)^{L_2^k}}, \end{aligned} \tag{1}$$

Where $ComWeight_1$ and $ComWeight_2$, respectively, denote the total weight of common elements in Node List 1 and Node List 2. $ObjWeight_1$ and $ObjWeight_2$, respectively, denote the total weight of all elements in Node List 1 and Node List 2. M_1 and M_2, respectively, represent the sum of occurrences of common elements in Node List 1 and Node List 2. N_1 and N_2, respectively, represent the sum of all elements in Node List 1 and Node List 2. L_1^i is level value of the i^{th} common element in Node List 1. L_2^j is the level value of the j^{th} common element in Node List 2. L_1^k and L_2^k, respectively, represent the level value of the k^{th} element in Node List 1 and Node List 2. r is the increasing factor of weight proposed by users.

V. Maintaining TCFXSs in Sliding Windows

A. EHTCFXSs

Definition 4: Let an EHTCFXS be a team of TCFXSs in sliding windows, that is $EHTCFXS =$

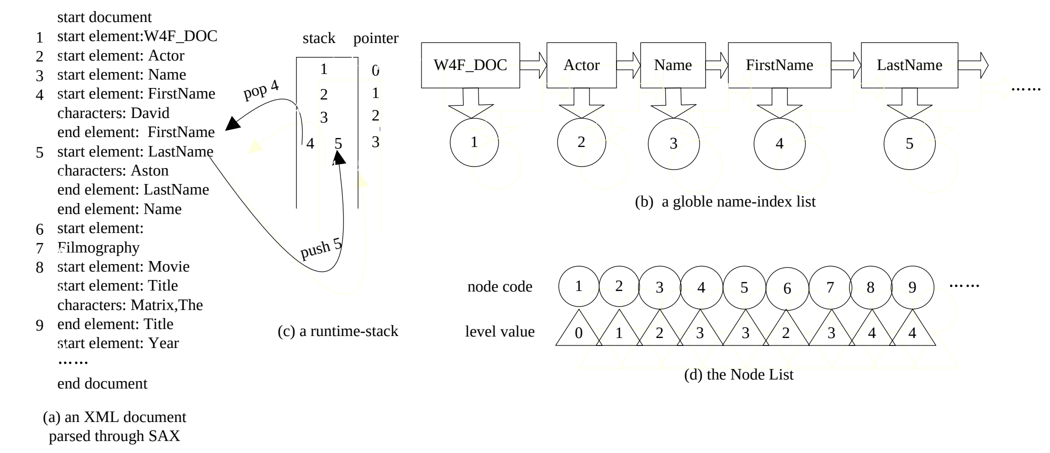

Figure 1. An example of acquiring coding values and level values for an XML document

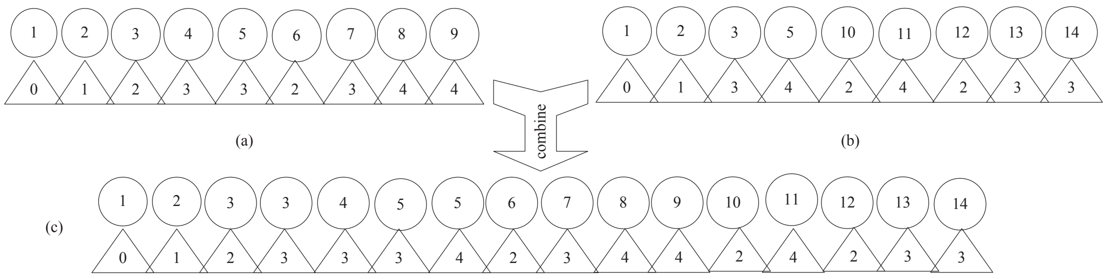

Figure 2. Combining Node Lists

$\{TCFXS(S_0), \ldots, TCFXS(S_i), \ldots\}$. $TCFXS(S_i)$ is the structure synopsis of the i^{th} sub-stream $S_i = \{X_{i_1}, \ldots, X_{i_n}\}$ with time stamp $T_{i_1}, \ldots, T_{i_n}$, such as $T_{i_j} < T_{i_m}$, where $j < m$.

To real-time maintain TCFXSs in sliding windows, an Exponential Histogram of Temporal Cluster Feature for XML Structure (for short EHTCFXS) in definition 4 can be seen as a cluster. The corresponding sub-stream of every TCFXS in an EHTCFXS is appointed to a grade value, which are denoted by superscripts such as $S_1^m, S_2^m, \ldots$. The grade value is related to the total number of documents in every sub-stream. The total number of documents in every sub-stream and the total number of sub-streams in each grade must adhere to four criterions of false positive exponential histograms in the literature [2] as following: (1) the timestamps of all XML documents in S_i are less than those in S_j, where $i < j$; (2) the number of documents in any sub-stream S just is $2^0 = 1, 2^1 = 2, 2^2 = 4\ldots$, and the new arriving sub-stream S_n only includes one document; (3) the grade value of S^j is j , where the number of documents in S^j is 2^j. The total number of sub-streams in each grade must be $[\frac{1}{\varepsilon}]$ or $[\frac{1}{\varepsilon}+1]$, as ε is error parameter proposed by users; (4) all documents in every sub-stream after S_1 are not overdue, where $TCFXS(S_1).t$ is in effect. The first criterion is used to ensure that TCFXSs in an EHTCFXS have not time overlap. The second criterion is an inherent requirement of an exponential histogram. The third criterion is used to limit the number of documents in every sub-stream. Besides, it uses an optional parameter (defined by users) to limit the total number of sub-streams in each grade. The forth criterion denotes how to identify overdue documents. When the first criterion is in effect, we only need to detect the oldest TCFXS in an EHTCFXS for identifying overdue documents.

B. Grouping XML Data Stream by Structure

At any time t, multi-EHTCFXSs are maintained in a sliding window. When an XML document X_p arrives to the sliding window, three interrelated processes are used to decide which cluster X_p will belong to. Firstly, X_p is parsed and rebuild into a $TCFXS(X_p)$. Secondly, similarities between each existing cluster and $TCFXS(X_p)$ are computed and the largest similarity value sim_{max} and the corresponding $EHTCFXS_{max}$ are selected. At last, one of two managing methods is implemented according to the result of comparing sim_{max} with the least similarity threshold ω. $TCFXS(X_p)$ is merged into the existing cluster $EHTCFXS_{max}$, where $sim_{max} \geq \omega$. Otherwise, a new EHTCFXS only including $TCFXS(X_p)$ is built.

The steps of merging $TCFXS(X_p)$ into

$EHTCFXS_{max}$ are introduced as follows: (1) build a new 0-grade $TCFXS(S^0)$ based on definition 3, where $S^0 = \{X_p\}$; (2) add $TCFXS(S^0)$ into the set of 0-grade sub-streams in $EHTCFXS_{max}$. If the total number of 0-grade sub-streams is $[\frac{1}{\varepsilon}+2]$, two oldest TCFXSs in the set are merged into a new 1-grade by property 2; (3) repeat the step 2 for sets of the different grade sub-streams until the total number of each grade sub-streams meet with the 3th criterion of false positive exponential histograms described in Section 5.1; (4) compare the total number of documents in the sliding window with the size of the sliding window N, and then delete overdue TCFXSs and corresponding EHTCFXS where the total number of documents exceeds N.

If similarities between $TCFXS(X_p)$ and each existing cluster are less than the least similarity threshold, a new EHTCFXS only including X_p will be built. Online maintaining these EHTCFXSs needs memory, so the total number of clusters in a sliding window is limited to usable memory. When the total number of EHTCFXSs in a sliding window exceeds the allowed largest number NC, some rules are used to decrease the total number of EHTCFXSs in the sliding window. This paper proposes a simply and direct method to delete some EHTCFXSs according to the following rules. One rule is that the selected EHTCFXS should include the least number of documents; another is that the selected EHTCFXS must include a special TCFXS which is in 0-grade sub-streams and has the oldest time stamp. The first rule indicates the EHTCFXS is an isolated point in streams. The second rule denotes that the EHTCFXS doesn't be updated lately and be close to overdue clusters. The experiments in this paper use the second rule, reasons are as the following: (1) deleting isolated points only increases few memory; (2) mining isolated points is one of goals of clustering research; (3) the technology of sliding windows focuses on recent or current clusters, which just tallies with the goal of the second rule.

C. Algorithm and Time Complexity

Further to the above discussions, Algorithm 1 (called Clustering XML Data Streams by Structure based on Sliding Windows and Exponential Histograms, for short CXDSS-SWEH) outlines incremental updating progress of clustering results. The Step 1 directly calls sub-algorithm $CreateTCFXS(X_t, NameIndex)$ to build $TCFXS(X_t)$ for a newest arriving XML document X_t. Given the length of the existing global name-index is L_{max} and the total number of distinct elements in X_t is N_t, the time complexity of the sub-algorithm can be approximated as $O(L_{max} \times N_t)$ according to the definition and building method introduced in Section 4.1. The initialization of clustering (Steps 2 to 4) generates an EHTCFXS only including $TCFXS(X_t)$. Therefore, the overall time of Steps 2 to 4 is constant. The first loop of Steps 6 to 8 computes similarities between each

Algorithm 1 CXDSS-SWEH.

```
Input: $X_t$ is a XML document arriving on window at time t;
        $\omega$ is the least similarity threshold;
        $NC$ is the largest number of EHTCFXSs included in window;
        $N$ is the largest number of XML documents contained in window;
        NameIndex is a list of saving pairs of element name and number;
      H[k] is a group of EHTCFXSs of representing clustering results before time t.
Output: H[l] is a group of EHTCFXSs which represent clustering results at time t.

1: get TCFXS($X_t$) using sub-Algorithm CreateTCFXS($X_t$,NameIndex);
2: if (k ==0)
3:     { generate an EHTCFXS H[1] only containing TCFXS($X_t$) ;
4:       return H[1]; }
5: else{ $sim_{max}$=0; count=k;
6:        for(i=1; i<=k; i++)
7:            { computing similarity sim between $H_i$ and TCFXS($X_t$) ;
8:              if (sim> $sim_{max}$) { $sim_{max}$=sim; $H_{max}$=$H_i$;}}
9:          if (NodeSim(TCFXS($X_t$), $H_{max}$) >$\omega$)
10:             {  add        ($X_t$) into the set of 0-grade sub-streams in $H_{max}$;
11:                num=the cardinal of the set of 0-grade sub-streams in $H_{max}$;
12:                i=1;
13:                 while (num =1 /ε+2)
14:                     {merge two oldest TCFXSs in the set of (i-1)-grade into $T_i$ ;
15:                      add $T_i$ into the set of i-grade sub-streams in $H_{max}$;
16:                      num=the cardinal of the set of i-grade sub-streams in $H_{max}$;
17:                      i=i++;  }}
18:         else {generate an EHTCFXS $H_{k+1}$ only containing TCFXS($X_t$) ;
19:             count=k+1;
20:                if (count> NC)
21:                   {delete the overdue EHTCFXS ;
22:                  count--; }}
23:       sum number of XML documents contained in H[count] as $D_{sum}$;
24:       if ($D_{sum}$> N)
25:          {get the EHTCFXS $H_{old}$ containing the oldest TCFXS;
26:           delete the oldest TCFXS;
27:           if ($H_{old}$ == null)
28:              {delete the $H_{old}$ from H[count] ; count--; } }
29:         return H[count] ;}
```

existing cluster and $TCFXS(X_t)$ and chooses the largest value Sim_{max} and the corresponding EHTCFXS H_{max}. Its time complexity can be involved in comparing progress of computing similarities in Section 4.2. Given the total number of elements in two Node Lists, respectively, is N_1 and N_2, the time of Steps 6 to 8 is the same as larger value in two values, that is $O(Max(N_1, N_2))$, in the worst state. Conditional statements of Steps 9 to 17 firstly compare Sim_{max} with the least similarity threshold, then decide whether building a new cluster (Steps 18 to 22) or inserting $TCFXS(X_t)$ into H_{max} (Step 10 to 17) according to the comparing result. Among, the work of Steps 20 to 22 adjusts the total number of clusters through comparing the total number of existing clusters with the largest number of clusters allowed in a sliding window. Step 21 takes $O(k)$ time to scan existing clusters and delete the cluster which is not updated newly, where k is the number of existing clusters. Step 25 takes constant time to acquire the total number of XML documents in the sliding window. Overdue TCFXSs and corresponding EHTCFXS are deleted (Steps 27 to 29), where the total number of documents exceeds threshold value N. Give a cluster including M TCFXSs, deleting overdue TCFXSs need to scan all TCFXSs. So the total time complexity of Steps 25 to 29 is $O(M)$. In summary, the total time complexity of Algorithm 1 normally can be approximated as $O(L_{max} \times N_t + max(N_1, N_2) + k + M) \approx O(L_{max} \times N_t)$, where $O(k), O(M), O(max(N_1, N_2)) \ll O(L_{max} \times N_t)$.

VI. EXPERIMENTAL VALIDATIONS

A. Experimental Data Sets and Design

Our experiments involve two datasets, the XMLFiles real dataset used to evaluate cluster quality, and the XMLSimples simulated dataset used to measure memory and time consumption. The XMLFiles dataset contains 437 XML documents. The documents are from 23 various domains such as Movie (74), University (22), Automobile (189), Bibliography (16), Company (35), Hospitality message (25), Travel (10), Order (10), Auction data (4), Appointment (2), Document page (14), Bookstore (2), Play (20), Club (12), Medical (2), and Nutrition (1). The number of tags varies from 10 to 100 in these sources. The nesting level varies from 2 to 15. The XMLSimples simulated dataset randomly is created by an XML tool named oxygen based on some schemas of mature industries such as civil aviation and web application. There are 10419 documents in the dataset. Their size varies from 1k to hundreds of k.

The experiments are run in a PC with Pentium IV 2.4GHz and Windows XP. The static algorithm XCLS in the literature [8] and the stream-oriented clustering algorithm SW-XSCLS in the literature [11], as comparing methods, are implemented in same conditions and criterions. To eliminate influence of orders of XML documents and acquire more precise clustering results, we re-adjust orders of documents in two datasets through a hash algorithm, and simulate smooth XML data streams.

B. Experimental Results and Analysis

Figure 3 shows intra-cluster and inter-cluster similarities on XMLFiles dataset for three algorithms. Intra-cluster similarities of three methods across different number of documents are over 0.975, and inter-cluster similarities are less 0.07, as shown in Figure 3. In essence, the XCLS and the SW-XSCLS use same methods to compute similarities, so the change of clusters quality is similar. However, existing clusters in the SW-XSCLS can include fewer documents because of removing overdue documents in sliding windows. So the influence of Level Structures in the SW-XSCLS is less than that in the XCLS. Results of the SW-XSCLS are better than that of the XCLS. The intra-cluster similarity of the CXDSS-SWEH is less than others and the inter-cluster similarity of the CXDSS-SWEH is larger than them, as shown in Figure 3. The reason is that the equation of computing similarities in the XCLS and the SW-XSCLS is not symmetrical for some peculiar documents in real dataset XMLFiles. For example, two documents have common number of levels, but elements in their level 3 satisfy inclusion relation instead of equivalence relation. For these documents, similarities computed by the XCLS and the SW-XSCLS are 1, otherwise the similarity computed by the CXDSS-SWEH is less 1. In fact, their structure is not exactly same. So, the computing equation in the CXDSS-SWEH is more logical than that in the XCLS and the SW-XSCLS.

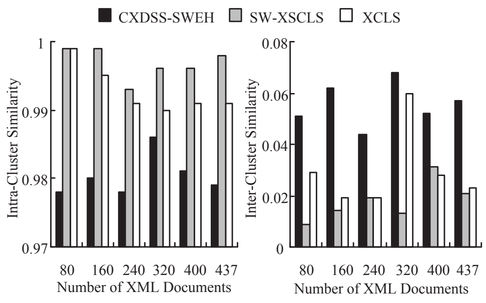

Figure 3. Quality comparison ($r = 2, \varepsilon = 2, \omega = 0.8, N = 100, NC = 50$)

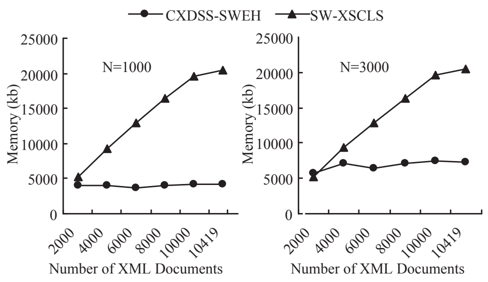

Figure 4. Memory consumed vs. number of XML documents ($r = 2, \varepsilon = 2, \omega = 0.8, NC = 130$)

Though the size of sliding windows is different, as shown in Figure 4-light and right, the memory consumption of the same algorithm has not increased evidently. But, the memory consumption of the CXDSS-SWEH is obviously less than that of the SW-XSCLS. The memory consumption often consists of two part. One part is used to maintaining clusters, and another is used to building structure synopses. For a smooth stream, the number of clusters in a sliding window generally fixed on at any time. A structure cluster with high intra-cluster similarities consumes fixed memory to save the DTD or Schema of all XML documents in the cluster. So, the memory consumption of maintaining clusters is steady. The SW-XSCLS consumed more memory than the CXDSS-SWEH in building structure synopses. As described in Section 4.1, the progress of parsing and building Node Lists uses runtime stacks to save memory. The progress of parsing and building Level Structures in SW-XSCLS is based on pruning DOM trees. When the structure of documents is complex, lots of memory is used to save trees.

Figure 5 shows a comparison of time cost on XML-Simples data with the SW-XSCLS and the CXDSS-SWEH. Though sizes of sliding windows are different, the trends of time change for two methods are similar and increase with the number of XML documents. As described in Section 5.3, the time complexity of the CXDSS-SWEH is proportion to the total number of distinct elements in sliding windows. When the number of documents increases, the total number of distinct elements generally can increase. So time cost also

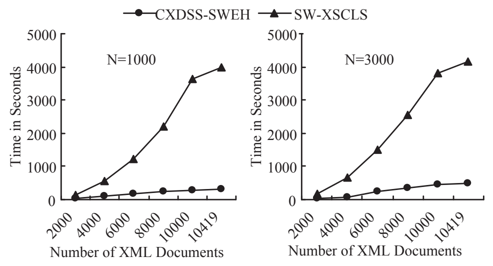

Figure 5. Time vs. number of XML documents ($r = 2, \varepsilon = 2, \omega = 0.8, NC = 130$)

increase with the number of documents. Structure synopses in the SW-XSCLS also are Level Structures. Time cost of maintaining these synopses in sliding windows increases with the number of documents, as described in the literature [8]. The time cost of the CXDSS-SWEH is less than that of the SW-XSCLS with the same number of documents and parameters user-defined, as presented in Figure 5. The equation of computing similarities in the XCLS is not transitive. The asymmetry would lead to twice matching progress for acquiring common elements. In the worst state, the time complexity of each matching progress is $O(m \times n)$: m and n, respectively, are the total number of elements in two Level Structures. And the time complexity of matching progress in the CXDSS-SWEH is $O(max\{N_1, N_2\})$: N_1 and N_2, respectively, are the number of elements in two Node Lists. From angle of time complexity, these steps just are the key steps in whole algorithm, and will take the most time. Hence, the time cost of the SW-XSCLS increases extremely large as the number of XML documents increases, whereas there is no significant difference in time cost of the CXDSS-SWEH.

VII. Conclusion And Future Work

To group an online XML data stream by structure, this paper introduces an algorithm, named CXDSS-SWEH, based on sliding windows and exponential histograms. The method parses XML documents through SAX, and then formalizes their structure into synopses named TCFXS. Each cluster in sliding windows consists of a team of TCFXSs, which satisfy criterions of false positive exponential histograms. To validate empirical effects, we have conducted a series of experiments involving real and simulative XML data. Our experimental results have confirmed: (1) clustering quality of the CXDSS-SWEH is close to the static clustering method XCLS and the stream clustering method SW-XSCLS; (2) memory and time consumption of the CXDSS-SWEH are efficient and effective, compared to the SW-XSCLS.

But, the existing cluster feature only considers structure, and omits all semantic information. In many fields, such as identifying online buyers, context in XML is more important. In future work, we will improve some context features.

Acknowledgment

The work is the National Natural Science Foundation of China (60873145), and the Doctor Start-up Program of Beijing University of Technology (X0007011200901).

References

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, S. Microsystems, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C Recommendation 16 August 2006, 2006.

[2] J. Chang, F. Cao, and A. Zhou. *Clustering evolving data streams over sliding windows.* Journal of Software, 18(4):905-918, 2007.

[3] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis. *A methodology for clustering xml documents by structure.* Information Systems Journal, 31(3):187-228, 2006.

[4] S. Flesca1, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. *Detecting structural similarities between xml documents.* In Proceedings of the 5th International Workshop on the Web and Databases, WebDB, pages 55-60, June 2002.

[5] C. Koch and S. Scherzinger. *Attribute grammars for scalable query processing on xml streams.* The VLDB Journal, 16:317-342, 2007.

[6] W. Lian, D. W. lok Cheung, N. Mamoulis, and S.-M. Yiu. *An efficient and scalable algorithm for clustering xml documents by structure.* IEEE Transactions on Knowledge and Data Engineering, 16(1):82-96, 2004.

[7] V. Mayorga and N. Polyzotis. *Sketch-based summarization of ordered xml streams.* In IEEE 25th International Conference on Data Engineering, 2009, pages 541-552, April 2009.

[8] R. Nayak. *Fast and effiective clustering of xml data using structural information.* Knowl Inf Syst, 14:197-215, 2008.

[9] A. Nierman and H. V. Jagadish. *Evaluating structural similarity in xml documents.*In Proceedings of the 5th International Workshop on the Web and Databases, WebDB 2002, pages 61-66, June 2002.

[10] C. Yang, C. Liu, J. Li, J. X. Yu, and J. Wang. *Semantics based buffier reduction for queries over xml data streams.* In Nineteenth Australasian Database Conference,ADC2008, pages 145-153, January 2008.

[11] G. Mao, M. Gao, W. Yao. *An algorithm for clustering XML data stream using sliding window.* The Third International Conference on Advances in Databases, Knowledge, and Data Applications, pages 96-101, January 2011.

[12] S. Zheng, A. Zhou, and L. Zhang. *Similarity measure and structural index of xml documents.* Chinese Journal of Computers, 26(9):1116-1122, 2003.

[13] N. Bruno, L. Gravano, N. Koudas, and D. Srivastava. *Navigation- vs. index-based XML multi-query processing.* In Proceedings of ICDE, pages 139-150, 2003.