XStreamCluster: an Efficient Algorithm for Streaming XML data Clustering*

Odysseas Papapetrou[1], Ling Chen[2]

[1] L3S Research Center, University of Hannover, Germany,
papapetrou@L3S.de
[2] QCIS, University of Technology Sydney, Australia
ling.chen@uts.edu.au

Abstract. XML clustering finds many applications, ranging from storage to query processing. However, existing clustering algorithms focus on static XML collections, whereas modern information systems frequently deal with streaming XML data that needs to be processed online. Streaming XML clustering is a challenging task because of the high *computational* and *space* efficiency requirements implicated for online approaches. In this paper we propose *XStreamCluster*, which addresses the two challenges using a two-layered optimization. The bottom layer employs Bloom filters to encode the XML documents, providing a space-efficient solution to memory usage. The top layer is based on Locality Sensitive Hashing and contributes to the computational efficiency. The theoretical analysis shows that the approximate solution of XStreamCluster generates similarly good clusters as the exact solution, with high probability. The experimental results demonstrate that XStreamCluster improves both memory efficiency and computational time by at least an order of magnitude without affecting clustering quality, compared to its variants and a baseline approach.

1 Introduction

In the past few years we have seen a growing interest in processing streaming XML data, motivated by emerging applications such as management of complex event streams, monitoring the messages exchanged by web-services, and publish/subscribe services for RSS feeds [2]. Various research activities have been triggered accordingly, including query evaluation over streaming XML data [3], summarization of XML streams [2] as well as classification of XML tree streams [4]. However, to the best of our knowledge, there exists no work on clustering *streaming* XML data, albeit extensive research has been carried out toward clustering *static* XML collections [5–8].

Streaming XML clustering is important and useful in many applications. For example, it enables the building of individual indices for each of the clusters, which in turn improves the efficiency of query execution over XML streams. The

* Extended version of [1]

problem is different than the one of clustering static XML collections, due to the typical high *computation* and *space* efficiency requirements of online approaches. As we explain later, existing approaches for clustering static XML collections do not meet these requirements, and therefore cannot be applied to streaming data. Therefore, this work designs an online approach for clustering of streaming XML.

More specifically, we focus on streaming structure-based clustering, i.e., clustering based on the structural similarity of the documents in terms of the common edges shared by their XML graphs (e.g., the pairs of parent-child elements). As discussed in [6], this kind of clustering is particularly important for XML databases, as it yields clusters supporting efficient XML query processing. First, clusters not containing the query can be efficiently filtered out, thereby eliminating a large portion of the candidate documents inexpensively. Second, each cluster of documents can be indexed more efficiently in secondary memory, due to the structural similarity of the documents.

In the context of structural XML clustering, an XML document can be represented as a set of edges $(e_1, e_2, e_3, \ldots)$. This representation makes the problem similar to clustering of streaming categorical data. However, existing approaches for clustering streaming categorical data are also not sufficient for streaming XML. Although most of them are designed with special concerns on computational efficiency, according to [9] they are not sufficiently efficient in terms of memory, especially when clustering *massive-domain data* where the possible domain values are so large that the intermediate cluster statistics cannot be maintained easily. Therefore, considering XML streams that encode massive-domain data, it is critical to design online XML clustering approaches which are both time and space efficient.

In the massive domains case, the edges are drawn from a universe of millions of possibilities. Therefore, maintaining the cluster statistics for all clusters in main memory becomes challenging. Recently, an approximate algorithm was proposed which uses compact sketches for maintaining cluster statistics [9]. The promising results delivered by this approximate solution, motivated us to apply an even more compact sketching technique based on Bloom filters to encode the intermediate cluster statistics. In addition, considering XML streams consisting of heterogeneous documents where a large number of clusters is created, we reduce the number of required comparisons between each newly incoming document and all existing clusters using another approximation technique based on Locality Sensitive Hashing (LSH).

Precisely, we propose XStreamCluster, an effective algorithm which employs two optimization strategies to improve time and space efficiency respectively. At the top level, LSH is used to quickly detect the few candidate clusters for the new document out of all clusters. This first step reduces drastically the required document-cluster comparisons, improving the time efficiency of the algorithm. At the bottom level, Bloom filters are employed to encode the intermediate cluster statistics, contributing to the space efficiency. Although the two levels introduce a small probability of errors, our theoretical analysis shows that XStreamCluster provides similar results to an exact solution with very high probability.

In Section 2 we discuss the state-of-the-art in XML clustering. Section 3 describes our approach in detail, whereas Section 4 presents the experimental evaluation of the method, with respect to time and space efficiency, as well as clustering quality. Section 5 concludes this paper.

2 Related work

We now review related work in the areas of streaming clustering and clustering of XML data.

As an important data mining technique, clustering has been widely studied by different communities. Detailed surveys can be found in [10, 11]. In the scenario of streaming data, the problem of clustering has also been addressed before, e.g., [12–14]. Considering the large volume of incoming data, computational efficiency is one of the most critical issues addressed by these works. For example, [13, 14, 9] proposed approximate clustering for increasing the clustering efficiency. A micro-clustering approach with a pyramidal time-frame concept was proposed in [12] to support greater flexibility for querying stream clusters over different time horizons. Our work differs from these works substantially, since it focuses on XML data. Furthermore, our application requirements impose clustering with a similarity threshold; therefore, unlike previous works, the number of clusters k is not decided a priori, but it evolves based on the stream contents.

Recently, the issue of space efficiency in clustering massive-domain streaming data was stressed [9], and an approximate solution based on the *count-min sketch* technique was proposed. Our algorithm also provides an approximate solution by using compact sketches to maintain intermediate cluster information in main memory. However, since we measure the similarity between XML documents in terms of number of shared edges (rather than the frequency of shared edges), our algorithm utilizes the more compact structure of Bloom filters, further reducing the memory requirements.

Efforts on clustering steaming data have also been extended to domains containing categorical [15] and textual data [16]. The research on clustering semi-structured stream data is still limited. Asai et al. [17] investigated data mining from semi-structured data streams. However, they focused on discovering frequent tree patterns from XML streams instead of clustering XML documents.

Clustering of *static* XML documents also attracted a lot of attention. Based on the adopted similarity/distance measure, existing static XML clustering approaches can be broadly divided into the following categories: *structure-based approaches*, *content-based approaches*, and *hybrid approaches*.

Early structure-based approaches usually represent XML documents as tree structures. The *edit distance* is then used to measure the distance between two XML trees, based on a set of edit operations such as inserting, deleting, and relabeling a node or a subtree [5, 7]. However, computing tree edit distances requires quadratic time complexity, making it impractical for clustering of XML streams. Differently, Lian et al. [6] proposed to represent XML documents as a set of parent-child edges from the corresponding structure graphs, which enables the efficient calculation of structural similarity of XML documents. Therefore, in our work, we employ the distance measure defined in [6] to design our online clustering algorithm for streaming XML data.

An important efficiency consideration for the existing structure-based algorithms is the large number of incurred document-cluster comparisons, as shown in Section 4. Particularly for the case of massive domain data, a large number of clusters is expected, aggravating the costs substantially. These costs become prohibitively expensive for clustering of *streaming* XML, rendering the existing algorithms unsuitable. Therefore, in this work we also include a probabilistic method which reduces drastically the comparisons between the incoming document and existing clusters, allowing XStreamCluster to handle streams.

Content-based XML clustering approaches are mainly used for clustering text-centric XML documents. Vector space models have been widely used to represent XML documents [18, 19]. Recently, there were also a few hybrid approaches which cluster XML documents by considering both structures and contents. For example, Doucet and Lehtonen [20] extract bags-of-words and bags-of-tags from the content and structure of XML documents as feature vectors. The type of clustering performed is thus substantially a textual clustering, while the results were shown to be better than those of other competing methods in the INEX 2006 contest. Although our work focuses on structural similarity, the proposed algorithm can be extended to cluster XML documents based on content, as well as the the combination of structure and content, once XML documents are represented as vector space models. For example, we can use *counting Bloom filters* to encode the feature vectors of XML documents and clusters.

3 Streaming XML clustering with XStreamCluster

We start by introducing the framework of our algorithm, and the preliminaries concerning the XML representation and distance measure. We then elaborate on the two optimization strategies. XStreamCluster (Fig. 1) clusters streaming XML documents at a single pass. When a new XML document arrives, instead of comparing it against all existing clusters, the top-level strategy - the LSH-based Candidate Cluster Detection - efficiently selects a few candidate clusters which are most similar to the new document. The algorithm then proceeds to compute the distance between the new document and each of the candidate clusters. To reduce memory requirements, the bottom-level strategy - the Bloom filter based Distance Calculation - computes the distance between the XML document and each of the candidate clusters based on their Bloom filter representations. Finally, a decision is made to either assign the new document to one of the existing clusters, if their distance is sufficiently low, or to initialize a new cluster for the current document.

3.1 Preliminaries

As discussed in Section 2, existing work on clustering static XML documents adopted various similarity/distance measures, ranging from structure-based mea-

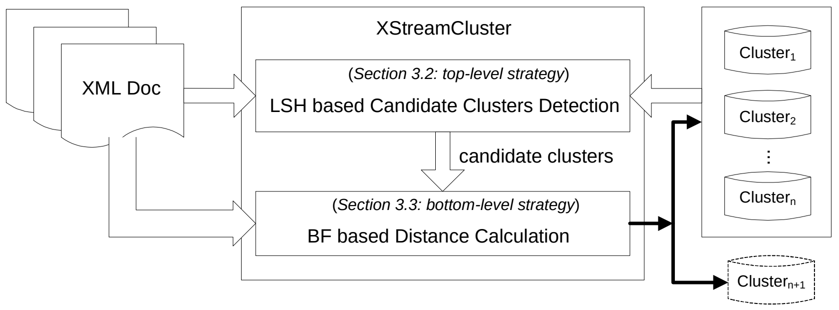
XML Doc
XStreamCluster
(Section 3.2: top-level strategy)
LSH based Candidate Clusters Detection
candidate clusters
(Section 3.3: bottom-level strategy)
BF based Distance Calculation
Cluster1
Cluster2
Clustern
Clustern+1

Fig. 1. The framework of XStreamCluster.

sures to content-based measures. In our work, we focus on clustering XML documents based on their structure. In the same context, Lian et al. in [6] proposed using a revised Jaccard similarity measure, and demonstrated its effectiveness for structure-based XML clustering. They have also demonstrated experimentally the benefits of XML clustering based on this measure, for optimizing the database storage layer with respect to XML query processing. We therefore decided to use the same measure for XStreamCluster, which we briefly describe below. Note however that the algorithm can also be adapted to other similarity measures. In the following sections, we will outline the required changes for adapting the algorithm to different distance measures.

In order to define the distance between two XML documents, the documents are first represented as structure graphs, or *s-graphs*.

Definition 1. (***Structure Graph***) *Given a set of XML documents C, the structure graph of C, $sg(C) = (N, E)$, is a directed graph such that N is the set of all the elements and attributes in the documents in C and $(a, b) \in E$ if and only if a is a parent element of element b or b is an attribute of element a in some document in C.*

For example, Fig. 2(b) shows the s-graphs of the two XML documents of Fig. 2(a). Given the s-graphs of two XML documents, a revised Jaccard coefficient metric is used to measure their distance.

Definition 2. (***XML Distance***) *For two XML documents d_1 and d_2, the distance between them is defined as $dist(d_1, d_2) = 1 - \frac{|sg(d_1) \cap sg(d_2)|}{\max\{|sg(d_1)|, |sg(d_2)|\}}$ where $|sg(d_i)|$ is the number of edges in $sg(d_i)$ and $sg(d_1) \cap sg(d_2)$ is the set of common edges of $sg(d_1)$ and $sg(d_2)$.*

As an example, consider the two XML documents and their s-graphs in Fig. 2 (a) and (b). Since $|sg(d_1) \cap sg(d_2)| = 3$ and $\max\{|sg(d_1)|, |sg(d_2)|\} = 5$, the distance between the two documents is $1 - 3/5 = 0.4$.

As stated in [6], this distance metric provides a nice feature to generate clusters which support efficient query answering. For example, if an XPath query q has an answer in some document d contained in a cluster C, then the s-graph

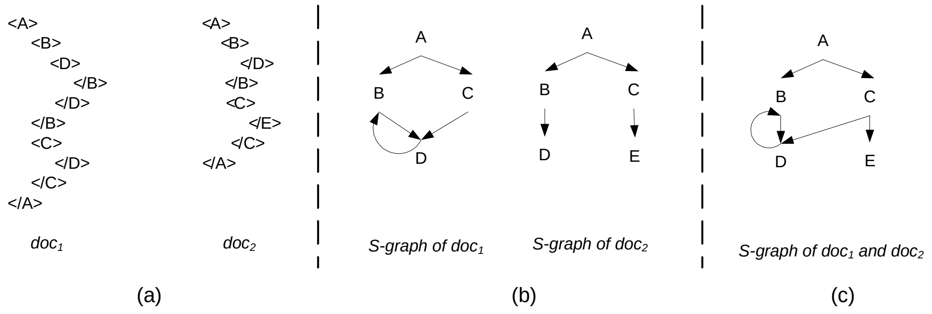

Fig. 2. The S-Graph Representation of XML documents.

of q, $sg(q)$, is a subgraph of $sg(C)$. That is, using the s-graphs of the clusters $C_1, C_2, \ldots, C_n$, we can safely filter out all clusters C_i for which $sg(q) \notin sg(C_i)$.

Note that the distance metric can be adopted straightforwardly in the context of streaming data clustering, because the s-graph of a document can be created efficiently in a single pass. Furthermore, the merging of the s-graph of a new document with that of an existing cluster can be performed efficiently, as discussed later.

3.2 LSH-based Candidate Clusters Detection

Traditional single-pass clustering algorithms need to compare each incoming document against all existing clusters, to find out the cluster with the minimum distance. However, considering XML datasets with heterogeneous structures, there may exist a large number of clusters, requiring a huge amount of time for comparing each document with all existing clusters. XStreamCluster addresses this issue by reducing the number of required document-cluster comparisons drastically. This reduction is based on an inverted index of clusters, built using Locality Sensitive Hashing (LSH).

The main idea behind LSH is to hash points from a high-dimensional space such that nearby points have the same hash values, and dissimilar points have different hash values. LSH is probabilistic, that is, two similar points will end up with the same hash value with a high probability p_1, whereas two dissimilar points will have the same hash value with a very low probability p_2. Central to LSH is the notion of *locality sensitive hash families*, i.e., an ordered collection of hash functions, formally defined as (r_1, r_2, p_1, p_2)-sensitive hash families [21].

Definition 3. *Let $\mathcal{S}$ denote a set of points, and $Dist(\cdot,\cdot)$ denote a distance function between points from $\mathcal{S}$. A family of hash functions $\mathcal{H}$ is called (r_1, r_2, p_1, p_2)-sensitive, where $r_1 \leq r_2$ and $p_1 \geq p_2$, if for any two points $p, q \in \mathcal{S}$ and for any $h_i \in \mathcal{H}$:*

– *if $Dist(p,q) \leq r_1$ then $Pr[h_i(p) = h_i(q)] \geq p_1$*

– *if* $Dist(p, q) \geq r_2$ *then* $Pr[h_i(p) = h_i(q)] \leq p_2$

For the case where the points of $\mathcal{S}$ are sets of elements (e.g., s-graphs are sets of edges), and $Dist(\cdot, \cdot)$ denotes the Jaccard coefficient, a suitable locality sensitive hash family implementation is *minwise independent permutations* [22]. Particularly, when hashing is conducted using minwise independent permutations, the probability that two points have the same hash value is $Pr[h_i(p) = h_i(q)] = 1 - Dist(p, q)$. For the case that $Dist(p, q) \leq r_1$, $Pr[h_i(p) = h_i(q)] \geq 1 - r_1$.

XStreamCluster employs LSH for efficiently detecting the candidate clusters for each document. Let $\mathcal{H}$ denote a locality sensitive hash family, based on minwise independent permutations. L hash tables are constructed, each corresponding to a composite hash function $g_i(\cdot)$, for $i = 1 \ldots L$. These hash functions $g_1(\cdot), g_2(\cdot), \ldots, g_L(\cdot)$ are obtained by merging k hash functions chosen randomly from $\mathcal{H}$, i.e., for a point $p : g_i(p) = [h_{i1}(p) \oplus h_{i2}(p) \ldots \oplus h_{ik}(p)]$. Each cluster s-graph is hashed to all L hash tables, using the corresponding hash functions. When XStreamCluster reads a new document d, it computes $g_1(sg(d)), \ldots, g_L(sg(d))$ and finds from the corresponding hash tables all clusters that collide with d in at least one hash table. These clusters, denoted as $\mathcal{C}(d)$, are returned as the candidate clusters for the document.

There is a latent difference between our approach for constructing the LSH inverted index of clusters and previous LSH algorithms, e.g., [21]. Previous approaches construct $g_i(p)$ by mapping each of $h_{ij}(p)$ to a single bit, for all $j : [1 \ldots k]$, and concatenating the results to a binary string of k bits. Due to this mapping to bits, the probability that two points $h_{ij}(p)$ and $h_{ij}(q)$ will map into the same bit value is at least 0.5, independent of their distance $Dist(p, q)$. Therefore, the probability for false positives is high. Previous works compensate for this issue by increasing the number of hash functions k, and thereby increasing the number of bits in each hash key $g(\cdot)$. But increasing the number of hash functions has a negative effect on computational complexity, which we want to avoid for the streaming data scenario. To this end, instead of mapping each of $h_{ij}(p)$ to a single bit, we represent the value of $h_{ij}(p)$ in the binary numeral system. We then generate $g_i(p)$ using the logic operation of *exclusive or* (denoted with XOR) on the set of $h_{ij}(p)$ values. Our theoretical analysis shows that the LSH based candidate cluster detection strategy retrieves the optimal cluster for each document with high probability.

Theorem 1. *The optimal cluster* C_{opt} *for document* d *will be included in* $\mathcal{C}(d)$ *with a probability* $Pr \geq 1 - (1 - (1 - \delta)^k)^L$, *where* δ *denotes the maximum acceptable distance between a document* d *and a cluster* C *for assigning* d *to* C.

Proof. The optimal cluster C_{opt} will be included in $\mathcal{C}(d)$ if C_{opt} collides with the document d in at least one of the L hash tables. We first compute the probability that d and C_{opt} collide in a given table ht_i, and from there we derive the probability that they collide in at least one table.

d and C_{opt} will collide in ht_i when $g_i(sg(C_{opt})) = g_i(sg(d))$. The probability $Pr[g_i(sg(C_{opt})) = g_i(sg(d))]$ is equal to the probability that all corresponding hash values in $g_i(\cdot)$ are equal, i.e., $h_{ij}(sg(C_{opt})) = h_{ij}(sg(d))$ for $j = [1 \dots k]$. According to [22], when hashing is implemented using minwise independent permutations, $Pr[g_i(sg(C_{opt})) = g_i(sg(d))] = \prod_{j=1}^{k} Pr[h_{ij}(sg(C_{opt})) = h_{ij}(sg(d))]$ $= \prod_{j=1}^{k}(1 - Dist(sg(C_{opt}), sg(d)))$. Note that, $Dist(sg(C_{opt}), sg(d))$ is by definition less than δ, otherwise C_{opt} would not be an acceptable cluster for d. Therefore, $Pr[g_i(sg(C_{opt})) = g_i(sg(d))] = (1-Dist(sg(C_{opt}), sg(d)))^k \geq (1-\delta)^k$.

Recall that LSH constructs L different hash tables, and C_{opt} will be included in $\mathcal{C}(d)$ if it collides with d in at least one of these tables. The probability that this happens is

$$\begin{aligned} Pr[\exists i : g_i(sg(C_{opt})) = g_i(sg(d))] = 1 - Pr[\nexists i : g_i(sg(C_{opt})) = g_i(sg(d))] \\ =1 - \prod_{i=1}^{L}(1 - Pr[g_i(sg(C_{opt})) = g_i(sg(d))]) \\ \geq 1 - \prod_{i=1}^{L}(1 - (1-\delta)^k) = 1 - (1 - (1-\delta)^k)^L \end{aligned} \qquad \square$$

For initializing the LSH inverted index, XStreamCluster needs to set the values of δ, k and L. The value of δ corresponds to the maximum acceptable distance between a document and the cluster for assigning the document to that cluster. Therefore, it depends on the requirements of the particular application, as well as the characteristics of the data. Nevertheless, as we show in the experimental evaluation, XStreamCluster offers substantial performance benefits for a wide range of δ. Note that δ is expressed using the standard Jaccard coefficient. Since the interesting measure for our work is the revised Jaccard coefficient, proposed in [6], we compute δ as follows $\delta \leq (1-\delta')/(1+\delta')$, where δ' is the same threshold expressed using the revised Jaccard coefficient. In order to set the values of L and k, the user first decides on the probability pr that a lookup in the LSH inverted index will return the optimal cluster for a document. Then, the values for L and k can be selected appropriately by considering Theorem 1. For example, let $\delta = 0.1$, and $pr \geq 0.95$. Then, according to Theorem 1: $1-(1-(1-0.9^k))^L \geq 0.95$. If we create $L = 10$ hashtables, then setting k as any value no greater than 12.8 should satisfy the required probability. However, the lower the value of k, the more candidate clusters will be returned, which incurs more time to filter false positive candidates. Consequently, we can set $k = 12$ hash functions for each hash table, which satisfies the probability requirements and minimizes the false positives.

After assigning the new document d to a cluster C, we need to update the L hash keys of C in the LSH hash tables. Normally, we would need to recompute these keys from scratch, which requires additional computation. Minwise hashing allows us to compute the updated hash values for the cluster C, denoted with $h'_{ij}(C)$, by using the values of $h_{ij}(d)$ and the current values of $h_{ij}(C)$ as follows: $h'_{ij}(C) = \min(h_{ij}(d), h_{ij}(C))$. The updated values of the $g_1(sg(C)), \dots, g_L(sg(C))$ can then be computed accordingly.

As already noted, the LSH-based index can be adapted to different distance definitions. For example, Broder et al. [22] present a locality sensitive hashing scheme for the standard Jaccard index, whereas Gionis et al. show how LSH can be adapted to use Euclidean distance. Similarly, Charikar [23] shows how LSH can be adapted to use the standard Cosine distance. Their results are directly applicable to our approach.

3.3 Bottom-level strategy: Bloom filter based Distance Calculation

After the top-level strategy detects a set of candidate clusters, we need to compute the distance between the new document and each candidate cluster, for finding the nearest one. As mentioned, for space efficiency XStreamCluster encodes s-graphs with Bloom filters. We now describe this encoding, and show how the distance between two s-graphs can be computed from their Bloom filters.

A Bloom filter is a space-efficient encoding of a set $S = \{e_1, e_2, \ldots, e_n\}$ of n elements from a universe U. It consists of an array of m bits and a family of λ pairwise independent hash functions $F = \{f_1, f_2, \ldots, f_\lambda\}$, where each function hashes elements of U to one of the m array positions. The m bits are initially set to 0. An element e_i is inserted into the Bloom filter by setting the positions of the bit array returned by $f_j(e_i)$ to 1, for $j = 1, 2, \ldots, \lambda$. To encode an s-graph with a Bloom filter, we hash all s-graph edges in an empty Bloom filter with a predefined length m and λ hash functions.

Recall from Section 3.1 that the XML distance between a document d and a cluster C is $dist(d, C) = 1 - \frac{|sg(d) \cap sg(C)|}{max\{|sg(d)|, |sg(C)|\}}$. Therefore, we need to estimate the values of $|sg(d)|$, $|sg(C)|$, and $|sg(d) \cap sg(C)|$ from the Bloom filter representations of $sg(d)$ and $sg(C)$. With BF_x we denote the Bloom filter encoding of $sg(x)$, where x denotes a document or a cluster. Let m and λ denote the length and number of hash functions of BF_x, and t_x be the number of true bits in BF_x. We estimate $|sg(x)|$ and $|sg(x) \cap sg(y)|$ as follows (the proofs are directly derived from [24]):

$$E(|sg(x)|) = \frac{\ln(1 - t_x/m)}{\lambda \ln(1 - 1/m)} \tag{1}$$

$$E(|sg(x) \cap sg(y)|) = 1 - \frac{\ln\left(m - \frac{mt_\wedge - t_x t_y}{m - t_x - t_y + t_\wedge}\right) - \ln(m)}{\lambda \ln(1 - 1/m)} \tag{2}$$

where $t_\wedge$ denotes the number of true bits in the Bloom filter produced by merging BF_x and BF_y with bitwise-AND.

Notice that the distances calculated using the estimated values of $|sg(d)|$, $|sg(C)|$, and $|sg(d) \cap sg(C)|$, may deviate slightly from the actual distance values. These deviations do not necessarily lead to a wrong assignment, as long as the nearest cluster (the one with the smallest distance) is correctly identified using the estimated values. A wrong assignment occurs only when the nearest cluster is not identified. Some of the wrong assignments have negligible effects, e.g., when the difference of the distances between the document and the two clusters is negligibly small; others may have a significant negative effect, e.g., when the assigned cluster is significantly worse than the optimal one. We are interested in the latter case, which we refer to as *significantly wrong assignments*, and analyze the probability of such errors.

Given a document d, the optimal cluster C_{opt} for d, and a suboptimal cluster C_{sub}, we define the assignment of d to cluster C_{sub} as a significantly wrong assignment if $dist(sg(d), sg(C_{sub})) - dist(sg(d), sg(C_{opt})) > \Delta$, where Δ is a user-chosen threshold. Since d was assigned to C_{sub} instead of C_{opt}, the estimated distance of d with C_{sub}, denoted as $\overline{dist}(sg(d), sg(C_{sub}))$, was smaller than the corresponding distance for C_{opt}. Therefore, we aim to find the probability $Pr[dist(sg(d), sg(C_{sub})) - dist(sg(d), sg(C_{opt})) > \Delta]$, given that $\overline{dist}(sg(d), sg(C_{sub})) < \overline{dist}(sg(d), sg(C_{opt}))$.

We use the following notations. $|ovl(d, C)|$ and $|\overline{ovl}(d, C)|$ denote the actual overlap cardinality and expected overlap cardinality (computed with Eqn. 2) of the sets $sg(d)$ and $sg(C)$. With t_d and t_C we denote the number of true bits in the Bloom filter of $sg(d)$ and $sg(C)$, whereas $t_\wedge$ denotes the number of true bits in the Bloom filter produced by merging the two Bloom filters with bitwise-AND. With $S(t_d, t_C, x)$ we denote the expected value of $t_\wedge$, given that $|ovl(d, C)| = x$. As shown in [24], $S(t_d, t_C, x)$ can be computed as follows: $S(t_d, t_C, x) = \frac{t_d t_C + m(1-(1-1/m)^{\lambda x})(m - t_d - t_C)}{m(1-1/m)^{\lambda x}}$.

We use the results of [24] to probabilistically bound the maximum deviation of the estimated overlap cardinality from the actual overlap cardinality.

Lemma 1 (Probabilistic Bounds). *For any $n_l \leq |\overline{ovl}(d, C)|$, the probability $Pr[ovl(d, C) > n_l]$ is at least equal to $1 - e^{t_\wedge - 1 - S(t_d, t_C, n_l)} \left(\frac{S(t_d, t_C, n_l)}{t_\wedge - 1}\right)^{t_\wedge - 1}$. Furthermore, for any $n_r \geq |\overline{ovl}(d, C)|$, the probability $Pr[ovl(d, C) < n_r]$ is at least equal to $1 - e^{-\frac{(t_\wedge + 1 - S(t_d, t_C, n_r))^2}{2S(t_d, t_C, n_r)}}$.*

Lemma 1 is used to compute the probability of a significantly wrong assignment. In particular we study the worst-case scenario, where the expected cardinalities of the overlap of the two clusters ($|\overline{ovl}(d, C_{sub})|$ and $|\overline{ovl}(d, C_{opt})|$) get the minimum possible value, given that the two clusters are candidates for the document. This value, denoted with $\overline{minOvl}$, is determined from the parameter δ as follows: $\delta = 1 - \overline{minOvl}/card \Rightarrow \overline{minOvl} = card(1 - \delta)$, with $card = \min(|sg(C_{opt})|, |sg(C_{sub})|)$. This is without loss of generality, because the accuracy of the estimations further increases when the overlap increases [24]. Furthermore, for simplification, we assume that we know $|sg(C_{opt})|$ and $|sg(C_{sub})|$, and that $|sg(d)| < |sg(C_{opt})|$ and $|sg(d)| < |sg(C_{sub})|$. We discuss about relaxing these assumptions later. For the theorem we use as shortcuts $t_{\text{mopt}} = S(t_d, t_{C_{opt}}, \overline{minOvl})$ and $t_{\text{msub}} = S(t_d, t_{C_{sub}}, \overline{minOvl})$.

Theorem 2. *The probability of a significantly wrong assignment $Pr[dist(sg(d), sg(C_{sub})) - dist(sg(d), sg(C_{opt})) > \Delta]$ is at most $1 - (1 - (\frac{t_l}{t_{msub} - 1})^{t_{msub} - 1} \times e^{t_{msub} - 1 - t_l}) \times (1 - e^{-\frac{(t_{mopt} + 1 - t_r)^2}{2t_r}})$, where $t_l = S(t_d, t_{C_{sub}}, \overline{minOvl} - \frac{\Delta'}{2|sg(C_{opt})|})$, $t_r = S(t_d, t_{C_{opt}}, \overline{minOvl} + \frac{\Delta'}{2|sg(C_{sub})|})$, and $\Delta' = \Delta \times |sg(C_{opt})| \times |sg(C_{sub})| - \overline{minOvl} \times (|sg(C_{sub})| - |sg(C_{opt})|)$.*

Proof. We first rewrite the probability of a significantly wrong assignment to a more convenient form.

$$\begin{aligned} Pr_{err} =& Pr[dist(sg(d), sg(C_{sub}) - dist(sg(d), sg(C_{opt}) > \Delta] \\ =& Pr[\frac{|sg(d) \cap sg(C_{opt})|}{\max(|sg(d)|, |sg(C_{opt})|)} - \frac{|sg(d) \cap sg(C_{sub})|}{\max(|sg(d)|, |sg(C_{sub})|)} > \Delta] \\ =& 1 - Pr[|sg(C_{sub})| \times |sg(d) \cap sg(C_{opt})| - \\ & |sg(C_{opt})| \times |sg(d) \cap sg(C_{sub})| \leq \Delta \times |sg(C_{opt})| \times |sg(C_{sub})|] \end{aligned} \quad (3)$$

We use Δ' as a shortcut to $\Delta \times |sg(C_{opt})| \times |sg(C_{sub})| - \overline{minOvl} \times (|sg(C_{sub})| - |sg(C_{opt})|)$. A pair of values satisfying the inequality of Eqn. 3 is: $|sg(C_{opt})| \times |sg(d) \cap sg(C_{sub})| > |sg(C_{opt})| \times \overline{minOvl} - \Delta'/2$ (left bound) and $|sg(C_{sub})| \times |sg(d) \cap sg(C_{opt})| < |sg(C_{sub})| \times \overline{minOvl} + \Delta'/2$ (right bound). The corresponding probabilities can be computed using Lemma 1. For the left bound we have: $Pr[|sg(C_{opt})| \times |sg(d) \cap sg(C_{sub})| > |sg(C_{opt})| \times \overline{minOvl} - \Delta'/2] = Pr[|sg(d) \cap sg(C_{sub})| > \overline{minOvl} - \Delta'/(2|sg(C_{opt})|)] \geq 1 - \left(\frac{t_l}{t_{\text{msub}}-1}\right)^{t_{\text{msub}}-1} e^{t_{\text{msub}}-1-t_l}$, where t_l denotes the expected number of true bits in the AND-merged Bloom filter when the number of elements in the intersection is $\overline{minOvl} - \frac{\Delta'}{2|sg(C_{opt})|}$. We compute t_l using $S(t_d, t_{C_{sub}}, \overline{minOvl} - \frac{\Delta'}{2|sg(C_{opt})|})$.

For the right bound we have: $Pr[|sg(d) \cap sg(C_{opt})| < \overline{minOvl} + \frac{\Delta'}{2|sg(C_{sub})|}] \geq 1 - e^{-\frac{(t_{\text{mopt}}+1-t_r)^2}{2t_r}}$, with $t_r = S(t_d, t_{C_{opt}}, \overline{minOvl} + \frac{\Delta'}{2|sg(C_{sub})|})$.

The resulting probability of a significantly wrong assignment is $Pr_{err} \leq 1 - \left(1 - \left(\frac{t_l}{t_{\text{msub}}-1}\right)^{t_{\text{msub}}-1} e^{t_{\text{msub}}-1-t_l}\right) \times \left(1 - e^{-\frac{\left(t_{\text{mopt}}+1-t_r\right)^2}{2t_r}}\right)$ □

As an example, consider the case when $\delta = \Delta = 0.2$, $m = 4096$, $\lambda = 2$, and $|sg(C_{sub})| = |sg(C_{opt})| = 1000$. Then, according to Theorem 2, the probability of a significantly wrong assignment is less than 0.025. We can further reduce this error probability by increasing the Bloom filter length. For example, for $m = 8192$ the probability is reduced to less than 0.002, and for $m = 10000$, the probability becomes less than 7×10^{-4}.

In Theorem 2, for simplification we assume that $|sg(C_{opt})|$ and $|sg(C_{sub})|$ are given. In practice, we can closely approximate both cardinalities using Eqn. 1. In addition, we can obtain probabilistic lower and upper bounds for $|sg(C_{opt})|$ and $|sg(C_{sub})|$, as described in [24], and use these to derive the worst-case values (i.e., the ones that minimize $\overline{minOvl}$, and maximize the probability of a significantly wrong assignment). Integrating these probabilistic guarantees in the analysis of Theorem 2 is part of our future work.

The Bloom filter encoding also allows an efficient updating of the s-graph representations of a cluster when a new document is assigned to it. As explained in [24], the bitwise-OR operation of two Bloom filters equals to the creation of a new Bloom filter of the union of two sets. We can therefore simply merge the corresponding Bloom filters of the document and the cluster with bitwise-OR, rather than generating the new Bloom filter for the updated cluster from scratch.

Algorithm 1 XStreamCluster

```
    INPUT: XML Stream $\mathcal{D}$, dist. threshold $\delta$
    OUTPUT: Set of clusters $\mathcal{C}$
 1: Initialize $L$ hash tables $ht_1, \ldots, ht_L$, corresponding to $g_1, \cdots, g_L$
 2: $\mathcal{C} \leftarrow \{\}$
 3: for each document $d$ from $\mathcal{D}$ do
 4:   for each $ht_i$, $i : [1 \ldots L]$ do
 5:     $\mathcal{C}(d) = \mathcal{C}(d) \cup ht_i.get(g_i(sg(d)))$
 6:   end for
 7:   Hash $sg(d)$ in $BF_d$
 8:   for each cluster $C \in \mathcal{C}(d)$ do
 9:     if $\overline{dist}(d, C) \geq \delta$ then
10:       $\mathcal{C}(d) = \mathcal{C}(d)/C$
11:     end if
12:   end for
13:   if $|\mathcal{C}(d)| \neq 0$ then
14:     Assign $d$ to cluster C = argmin dist(d,C)
15:   else
16:     Initialize a new cluster $C$ with $d$, $\mathcal{C} = \mathcal{C} \cup \{C\}$
17:   end if
18:   Update Bloom filter of $C$ and $L$ hashtables
19: end for
```

Similar to the case of the LSH-based index, the bottom-level optimization can also be adapted to different distance definitions. The constituting components of the Cosine distance, the standard Jaccard index, and the Euclidean distance among others, are $|sg(d_{1|2})|$, $|sg(d_1) \cap sg(d_2)|$, and $|sg(d_1) \cup sg(d_2)|$, which can be efficiently estimated using the Bloom filter representations of the documents and clusters [24]. Therefore, XStreamCluster can also be adapted to other distance definitions. This would be interesting for different application domains, such as standard high-dimensionality clustering, and text clustering. This is part of our future work.

The full algorithm of XStreamCluster is illustrated in Algorithm 1.

4 Experimental Evaluation

XStreamCluster was evaluated in terms of efficiency, scalability, and clustering quality, using streams of up to 1 million XML documents. Our simulations were carried out in a single dedicated Intel Xeon 3.6Ghz core.

Datasets. We conducted experiments with two streams. The first (STREAM1) was generated using a set of 250 synthetic DTDs. To verify the applicability of the experimental results for real DTDs, the second stream (STREAM2) was created following a set of 22 real, publicly available DTDs. In particular, for STREAM1 we first generated x DTDs, out of which we created y different XML documents with XML Generator [25]. For generating each document, we randomly selected one of the available DTDs as an input for the XML Generator. The values of x and y varied for each experiment, with a maximum of 250 and 1 million respectively. The resulting documents were fed to the stream in a random order. For generating STREAM2 we followed the same procedure, but using a set of real, frequently used DTDs. The full list of the used DTDs and the DTD generator are available online, in http://www.l3s.de/~papapetrou/dtdgen.html.

Algorithms. To evaluate in depth the contribution of each of the strategies to the algorithm's efficiency and effectiveness, we have compared three different variants of XStreamCluster. Furthermore, XStreamCluster was compared with the existing static algorithm which employs s-graphs for representing and comparing clusters and documents [6], called S-GRACE. In particular, we implemented and evaluated the following algorithms:

S-GRACE: We adapted S-GRACE [6] to streaming data. This required the following extensions: (a) we changed the clustering algorithm from ROCK to K-Means, and (b) we represented the s-graphs as extensible bit arrays, instead of bit arrays of fixed sizes.

XStreamBF: XStreamCluster with only the bottom-level strategy in place (i.e., encoding of s-graphs as Bloom filters; documents were compared to all clusters).

XStreamLSH: XStreamCluster with only the top-level strategy (i.e., indexing clusters using LSH; s-graphs were represented as extensible bit arrays).

XStreamCluster: The algorithm as presented in this paper.

Methodology. To evaluate efficiency and scalability, we measured the average time and memory required for clustering streams of up to 1 million documents. Quality was evaluated using the standard measure of normalized mutual information. In the following, we report average measures after 4 executions of each experiment. We present results with Bloom filters of 1024 bits, with 2 hash functions. The LSH index was configured for satisfying a correctness probability of 0.9. We present detailed results for STREAM1, and summarize the STREAM2 results, noting the differences.

4.1 Efficiency

With respect to efficiency, we compared the memory and execution time of each algorithm for clustering the two streams. To ensure that time measures were not affected by other activities unrelated to the clustering algorithm, e.g., network latency, we excluded the time spent in reading the stream.

For the first experiment, we studied how the efficiency of the algorithms changes with respect to the diversity of the stream. We controlled the diversity of the stream by choosing the number of DTDs out of which STREAM1 was generated. Figures 3 (a) and (b) plot the time and memory requirements of the four algorithms for clustering different instances of STREAM1, each generated by a different number of DTDs. The distance threshold for this experiment was set to 0.1, and the number of documents in the stream was set to $100k$. We see that XStreamCluster clearly outperforms S-GRACE in terms of speed; it requires up to two orders of magnitude less time for clustering the same stream. XStreamLSH presents the same speed improvement. The efficiency of both algorithms is due to the top-level strategy for candidate clusters detection, which drastically reduces the cluster-document comparisons. XStreamBF does not present this speed improvement as it does not employ an LSH inverted index.

We also observe that the speed improvement of XStreamLSH and XStreamCluster is more apparent for higher number of DTDs. This is because more DTDs lead to more clusters. For the S-GRACE and XStreamBF algorithms, more clusters lead to longer bit arrays, thereby requiring more time to cluster a document.

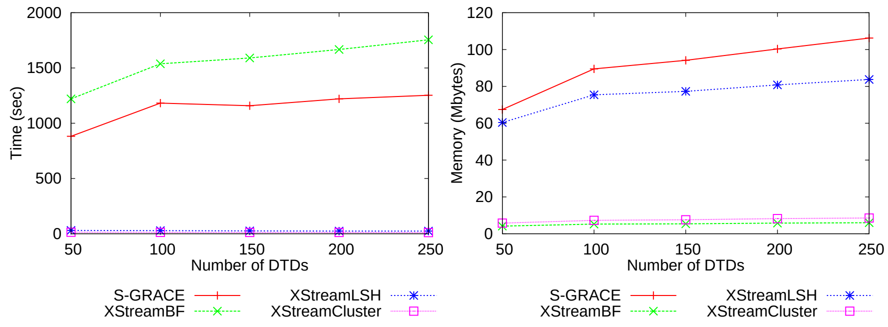

Fig. 3. (a) Time requirements, and, (b) Memory requirements, for clustering STREAM1 with respect to the number of DTDs.

Furthermore, more clusters lead to an increase in the cluster-document comparisons since each document needs to be compared to all clusters. This is not the case for XStreamLSH and XStreamCluster though, which pre-filter the candidate clusters for each document by using the top-level strategy. Therefore, the execution time of XStreamLSH and XStreamCluster is almost constant.

With respect to memory requirements (Figure 3(b)) we see that XStreamCluster and XStreamBF require at least one order of magnitude less memory compared to S-GRACE. The difference is again particularly visible for a higher number of DTDs, which results to a higher number of clusters. The huge memory savings are due to the Bloom filter encodings employed by the two algorithms.

The experiment was also conducted using STREAM2. However, since STREAM2 was generated from a limited number of DTDs, instead of varying the number of DTDs we varied the value of the distance threshold δ, which also had an influence on the number of clusters: reducing the δ value resulted to more clusters. Table 1 presents example results, for $\delta = 0.1$ and 0.2. As expected, reducing the δ value leads to an increase in memory and computational cost for S-GRACE and XStreamBF, due to the increase in the number of clusters. On the other hand, the speed of XStreamCluster and XStreamLSH actually increases by reducing the distance threshold, because the LSH index is initialized with less hash tables and hash functions.

Algorithm	Time (sec)		Memory (Mbytes)		NMI	
Distance thres.	0.1	0.2	0.1	0.2	0.1	0.2
S-GRACE	61	35	23.1	14	0.72	0.76
XStreamBF	363	253	1.7	1.1	0.72	0.76
XStreamLSH	8	11	19.1	13.8	0.716	0.755
XStreamCluster	4	7	2.5	3.1	0.715	0.755

Table 1. Example results for STREAM2

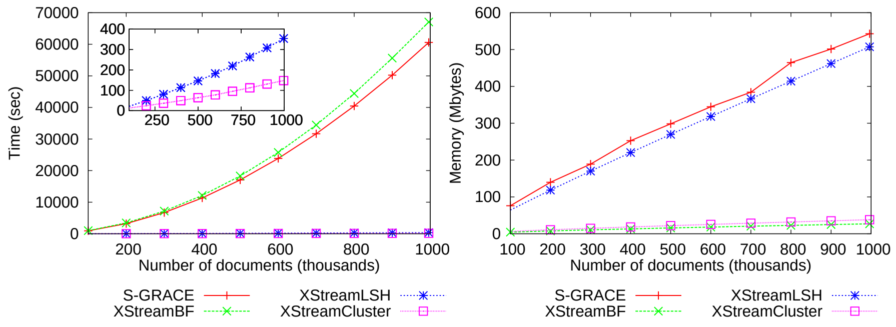

Fig. 4. (a) Execution time, and (b) Memory requirements, for clustering STREAM1 with respect to number of documents.

4.2 Scalability

To verify the scalability of XStreamCluster, we compared it against S-GRACE and its variants for clustering streams of different sizes, reaching up to 1 million documents. In particular, we have generated instances of STREAM1 and STREAM2 with 1 million documents, and used all four algorithms to cluster them. During clustering, we monitored memory and execution time every $100k$ documents. The experiment was repeated for various configurations. In the following we report the results for STREAM1 corresponding to 100 DTDs where δ is set to 0.1. The results for other settings lead to the same conclusions.

Figures 4(a) and 4(b) present the execution time and memory usage with respect to number of documents. With respect to execution time, we see that S-GRACE and XStreamBF fail to scale. Their execution time increases exponentially with the number of documents, because of the increase in the number of clusters. On the other hand, XStreamCluster and XStreamLSH have a linear scale-up with respect to the number of documents, i.e., the cost for clustering each document remains constant with the number of clusters. This is achieved due to the efficient filtering of clusters with the LSH-based candidate cluster detection strategy.

With respect to memory requirements (Fig. 4(b)), all algorithms scale linearly with the number of documents, but the algorithms that use Bloom filters require an order of magnitude less memory. Interestingly, for clustering 1 million documents, XStreamCluster requires only 39 Mbytes memory, which is an affordable amount for any off-the-shelf PC. Therefore, XStreamCluster can keep all its memory structures in fast main memory, instead of resorting to the slower, secondary storage. Keeping as many data structures as possible in main memory is very important for algorithms working with streams, because of their high efficiency requirements.

4.3 Clustering quality

We evaluated the clustering quality of XStreamCluster by using the standard measure of Normalized Mutual Information (NMI). NMI reflects how close the clustering result approaches an optimal classification – a ground truth – which is usually constructed by humans. It is formally defined as follows:

$$NMI(\Omega, \mathcal{C}) = \frac{I(\Omega, \mathcal{C})}{[H(\Omega) + H(\mathcal{C})]/2}, \quad \text{where} \quad I(\Omega, \mathcal{C}) = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N|\omega_k \cap c_j|}{|\omega_k||c_j|}$$

N is the number of documents, $\mathcal{C} = \{c_1, c_2, \ldots, c_j\}$ represents the set of clusters generated by the clustering algorithm, and $\Omega = \{\omega_1, \omega_2 \ldots \omega_k\}$ the set of classes in the optimal classification. $H(\mathcal{C})$ and $H(\Omega)$ define the entropy of the cluster and class *sizes*, i.e., $H(\mathcal{C}) = -\sum_j \frac{|c_j|}{N} \log \frac{|c_j|}{N}$ and $H(\Omega) = -\sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N}$. An NMI of 0 indicates a completely random assignment of documents to clusters, whereas an NMI of 1 denotes a clustering which perfectly corresponds to the optimal classification. For our datasets, the optimal classification Ω was defined by the DTD of each document: two XML files were considered to belong to the same class if they were generated from the same DTD file. We choose NMI over *purity* and *entropy*, which are also used in the literature, because the latter measures are sensitive to the number of clusters, and they cannot be used for comparing two clustering solutions if the solutions generate a different number of clusters. Since the number of clusters is not predetermined for any of the compared algorithms, it may happen that the clustering solutions have a different number of clusters, thereby invalidating the entropy and purity measures.

Figure 5(a) presents NMI with respect to distance threshold δ for STREAM1, which consists of $100k$ documents generated from 100 different DTDs. We see that all XStreamCluster variants achieve a clustering quality nearly equal to S-GRACE. Quality of XStreamBF is practically equal to quality of S-GRACE, which means that introducing the Bloom filters as cluster representations does not result to quality reduction. XStreamCluster and XStreamLSH have a small difference compared to S-GRACE, which is due to the aggressive filtering of clusters that takes place during clustering at the top-level strategy. This difference is negligible, especially for small distance threshold values.

We further studied how the diversity of the stream influences the algorithms' quality, by repeating the experiment using streams generated from a different number of DTDs. Figure 5(b) shows the NMI with respect to the number of DTDs used for generating STREAM1. We see that XStreamCluster variants again achieve a quality almost equal to S-GRACE. The difference between XStreamCluster and S-GRACE reduces with an increase in the number of DTDs, and becomes negligible for the streams generated from more than 100 DTDs.

As shown in the last column of Table 1, the same outcome was observed on the experiments with STREAM2. XStreamBF produced an equivalent solution to S-GRACE, whereas XStreamLSH and XStreamCluster approximated closely the optimal quality. The difference between the approximate solution produced by XStreamCluster and the exact solution produced by S-GRACE was less than 0.01 in terms of NMI, in all experiments.

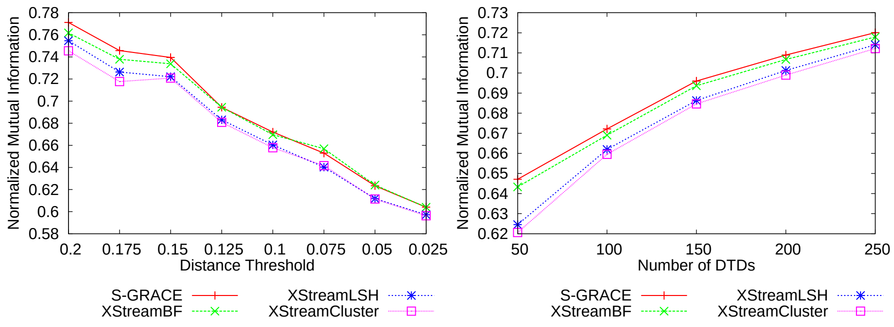

Fig. 5. Normalized Mutual Information for STREAM1. Varying (a) the distance threshold, and, (b) the number of DTDs.

Summarizing, XStreamCluster achieves good clustering of XML documents requiring at least an order of magnitude less cost compared to S-GRACE, with respect to both execution time and memory. The experimental results show that it is especially suited for clustering large and diverse streams, both with respect to quality and efficiency. Owing to the low memory and time requirements, it is easily deployable in standard off-the-shelf PCs and scales to huge XML streams.

5 Conclusions

We presented XStreamCluster, the first algorithm that addresses clustering of *streaming XML documents*. The algorithm combines two optimization strategies, Bloom filters for reducing the memory requirements, and Locality Sensitive Hashing to reduce significantly the cost of clustering. We provided theoretical analysis showing that XStreamCluster provides an approximately similar quality of clustering as exact solutions do. Our experimental results also confirmed the efficiency and effectiveness contributed by the two strategies of XStreamCluster.

For future work, we will consider additional distance measures, including the ones which combine both content similarity and structure similarity, and compare their effect on clustering efficiency and effectiveness. Furthermore, we will adapt the XStreamCluster algorithm for data other than XML, like user-generated streaming data on the web. Finally, we are working towards extending XStreamCluster to enable ‘forgetting old documents’, in a sliding window fashion.

References

1. Papapetrou, O., Chen, L.: XStreamCluster: an Efficient Algorithm for Streaming XML data Clustering. In: Proc. of DASFAA. (2011)
2. Mayorga, V., Polyzotis, N.: Sketch-based summarization of ordered XML streams. In: Proc. of ICDE. (2009)

3. Josifovski, V., Fontoura, M., Barta., A.: Querying XML streams. VLDB Journal **14**(2) (2005)
4. Bifet, A., Gavald, R.: Adaptive XML tree classification on evolving data streams. In: Proc. of ECML/PKDD. (2009)
5. Dalamagas, T., Cheng, T., Winkel, K.J., Sellis, T.: A methodology for clustering XML documents by structure. Inf. Syst. **31**(3) (2006) 187–228
6. Lian, W., Cheung, D.W.L., Mamoulis, N., Yiu, S.M.: An efficient and scalable algorithm for clustering XML documents by structure. IEEE TKDE **16**(1) (2004) 82–96
7. Nierman, A., Jagadish, H.V.: Evaluatating structural similarity in XML documents. In: Proc. of ACM SIGMOD WebDB Workshop. (2002) 61–66
8. Tagarelli, A., Greco, S.: Toward semantic XML clustering. In: Proc. SDM. (2006)
9. Aggarwal, C.C.: A framework for clustering massive-domain data streams. In: Proc. of IEEE ICDE. (2009)
10. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall (1988)
11. Kaufman, L., Rousseuw, P.: Finding groups in data - An introduction to cluster analysis. Wiley (1990)
12. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proc. of VLDB. (2003)
13. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: Proc. of IEEE FOCS. (2000)
14. O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., Motwani, R.: Streaming-data algorithms for high-quality clustering. In: Proc. of ICDE. (2002)
15. Ong, K.L., Li, W., Ng, W.K., Lim, E.P.: Sclope: an algorithm for clustering data streams of categorical attributes. In: LNCS, Vol 3181. (2004)
16. Zhong, S.: Efficient streaming text clustering. In: Neural Networks, Vol 5-6. (2005)
17. Asai, T., Arimura, H., Abe, K., Kawasoe, S., Arikawa, S.: Online algorithms for mining semi-structured data stream. In: Proc. of ICDM. (2002) 27–34
18. Candiller, L., Tellier, I., Torre, F.: Transforming xml trees for efficient classification and clustering. In: INEX. (2005) 469–480
19. Doucet, A., Ahonen Myka, H.: Naive clustering of a large XML document collection. In: INEX. (2002) 81–87
20. Doucet, A., Lehtonen, M.: Unsupervised classification of text-centric XML document collections. In: Proc. of INEX. (2006)
21. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proc. of VLDB. (1999)
22. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. In: Proc. of STOC '98, ACM (1998) 327–336
23. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: STOC, New York, NY, USA, ACM (2002) 380–388
24. Papapetrou, O., Siberski, W., Nejdl, W.: Cardinality estimation and dynamic length adaptation for bloom filters. Distributed and Parallel Databases **28**(1) (2010)
25. Diaz, A.L., Lovell, D.: XML generator (1999) Available at http://www.alphaworks.ibm.com/tech/xmlgenerator.