An Efficient and Scalable Algorithm for Clustering XML Documents by Structure

Wang Lian, David Wai-lok Cheung, *Member*, *IEEE Computer Society*, Nikos Mamoulis, and Siu-Ming Yiu

Abstract—With the standardization of XML as an information exchange language over the net, a huge amount of information is formatted in XML documents. In order to analyze this information efficiently, decomposing the XML documents and storing them in relational tables is a popular practice. However, query processing becomes expensive since, in many cases, an excessive number of joins is required to recover information from the fragmented data. If a collection consists of documents with different structures (for example, they come from different DTDs), mining clusters in the documents could alleviate the fragmentation problem. We propose a hierarchical algorithm (S-GRACE) for clustering XML documents based on structural information in the data. The notion of structure graph (s-graph) is proposed, supporting a computationally efficient distance metric defined between documents and sets of documents. This simple metric yields our new clustering algorithm which is efficient and effective, compared to other approaches based on tree-edit distance. Experiments on real data show that our algorithm can discover clusters not easily identified by manual inspection.

Index Terms—Data mining, clustering, XML, semistructured data, query processing.

1 Introduction

Extensible Markup Language (XML) has been recognized as a standard data representation for interoperability over the Internet. Web pages formatted in XML have started to appear. Besides flat file storage, object-oriented databases, and native XML databases, developers have been using the more mature relational database technology to store semistructured data, following two alternative approaches: *schema mapping* and *structure mapping*. In the first approach, a relational schema is derived from the Document Type Definition (DTD) of the documents [19]. The second approach creates a set of generic tables that store the structural information such as the elements, paths, and attributes of the documents [20].[1] Both methods decompose the documents and insert their components to a set of tables. This, however, brings excessive *fragmentation*, which creates a serious negative impact in query evaluation: The number of joins required to process a path expression is almost equal to the length of the path [19].

If the collection consists of XML documents with different structures, we observe that the fragmentation problem can be alleviated by *clustering* the documents according to their structural characteristics and storing each cluster in a different set of tables. For example, the documents in the DBLP database [5] can be classified to *journal articles* and *conference papers*. In terms of the elements (tags) and the parent-children relationships among them, the journal articles carry very different structural information than the conference papers.

In Fig. 1, the journal article and the conference paper have common elements such as $author$ and $title$, and some different elements such as $inproceedings$ and $article$. The main difference is not due to the small number of distinct elements, but due to the large number of distinct edges (i.e., parent-children relationships) between the elements. In fact, all edges are different in this example. Sometimes, a different element could introduce many edges that distinguish one group of documents from another. Clustering documents according to their structural information would improve query selectivity since queries are commonly constructed based on path expressions. For example, queries involving the edge "$article/volume$" need not access any data from the conference papers.

XML documents have diverse types of structural information (apart from edges) in different refinement levels, e.g., attribute/element labels, edges, paths, twigs, etc. When defining the distance between two documents, choosing a simple structural component (e.g., label, edge) as a basis would make clustering fast. On the other hand, a metric based on too refined components could make it less efficient and, hence, nonpractical. We have observed that using directed edges to define a distance between two XML documents is a good choice. More importantly, this metric can be applied not only on documents, but also on groups of documents. Finally, as shown in the paper, this approach makes clustering on XML documents scalable to large collections. Since clustering is performed on documents, no data from a document would be stored in tables associated to different clusters than the one where the document belongs. However, if a query needs to refer to more than one document, it may be necessary to join the tables from

1. An element is a metadata (tag) describing the semantic of the associated data. A path (or a path expression) specifies a navigation through the structure of the XML data based on a sequence of tags.

• *The authors are with the Department of Computer Science and Information Systems, University of Hong Kong, Pokfulam Road, Hong Kong. E-mail: {wlian, dcheung, nikos, smyiu}@csis.hku.hk.*

Manuscript received 1 Sept. 2002; revised 1 Apr. 2003; accepted 10 Apr. 2003. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118551.

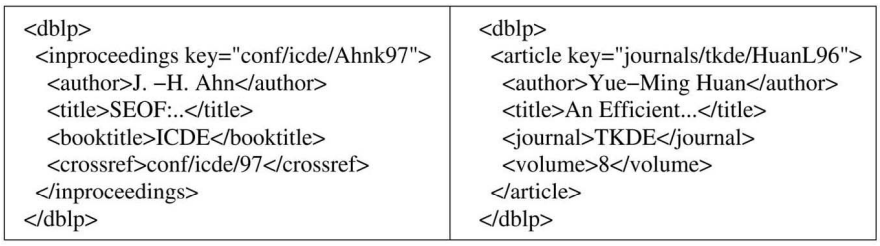


Fig. 1. Structural difference between article and conference papers.

two or more clusters. Some readers may think that this would create additional table joins. We will show in Section 2 that this is not the case.

Our contributions can be summarized as follows:

1. We show that, if a collection of XML documents have different structures, proper clustering alleviates the fragmentation problem.
2. We develop an algorithm S-GRACE which clusters XML documents by structure. The distance metric in S-GRACE is developed on the notion of structure graph which is a minimal summary of edge containment in documents.
3. We carry out performance studies on synthetic and real data. We show that S-GRACE is effective, efficient, and scalable. In the DBLP database [5], S-GRACE can identify clusters that cannot be spotted easily by manual inspection. Moreover, the queries on the partitioned schema derived from the clustering on the DBLP database exhibit large performance speed-up compared to the unpartitioned schema.

The rest of the paper is organized as follows: Section 1.1 discusses related work. Section 2 motivates the study and Section 3 describes the proposed S-GRACE clustering algorithm. Section 4 describes a query manager module, which transforms XQuery expressions [22] to queries on the database schema defined by the clustering process. In Section 5, we study the applicability of the proposed methodology on synthetic and real XML document collections. A discussion on how our work can be generalized using alternative graph summaries and clustering methods is made in Section 6. Finally, Section 7 concludes the paper with directions for future work.

1.1 Related Work

XML data can be stored in a file system [1], an object-oriented database [10], a relational database [19], or a native XML database system [15]. Using a file system is a straightforward option which, however, does not support query processing. Object-oriented database systems allow a flexible storage system of XML files. It can also support complicated query processing. Native XML database systems try to exploit features of semistructured data model in storing XML files. Nevertheless, both object-oriented and native XML database systems are neither mature nor efficient enough for industry adoption. On the other hand, even though relational database technology is not well-tuned for semistructured data, it is regarded as a practical approach because of its wide deployment in the commercial world.

In [19], the assumption of using a relational database to store XML files was established as a feasible approach. Based on that, different schema design methods were proposed. First, the notion of *DTD graph* was introduced, in which elements and attributes are nodes and the parent-children relationships become edges. Based on the graph, three approaches were proposed to design the database schema. Our approach proposed in this work also makes use of the structural information. However, it is based only on the data, without assuming the existence of DTDs. The algorithm STORED in [7] uses data mining to generate a relational schema from XML documents. The main contribution of STORED is the specification of a declarative language for mapping a semistructured data model to a relational model. Our approach is to discover the clusters among the XML documents so that each cluster can have a more refined schema.

Clustering is a well-studied subject [12], [16]. There have been considerable works on Web clustering. Previous work includes text-based [23] and link-based methods [11]. Their goal is to group Web documents of similar topics together, whereas our goal is to group XML documents of similar structures together. In the future, many Web pages could be in XML. Therefore, clustering XML files is a relevant problem in Web mining or categorical data [12]. Recently, Nierman and Jagadish [17] proposed a method to cluster XML documents according to structural similarity. The algorithm measures structural similarity between documents using the "edit distance" between tree structures. The motivation is to induce a "better" DTD for each cluster. Arguably, this approach can allow us to cluster XML documents and then refine the database schema using the DTD of each cluster. However, computing the edit distance between two documents has a complexity of $O(|A| \cdot |B|)$, where $|A|$ and $|B|$ are their respective sizes [17]. Computation of the edit distances for each documents pair is required by the clustering algorithm. The cost of this approach is too high for practical applications. On the other hand, we cluster graph summaries which are much smaller than the original documents and we define a similarity metric which is very cheap to compute. Furthermore, an XML document can be an arbitrary graph rather than a tree because of the explicit element references. For example, both id/idref attribute and XLink construct can create a cross-elements reference [6]. Our methodology can be applied to arbitrary XML graphs, not only trees.

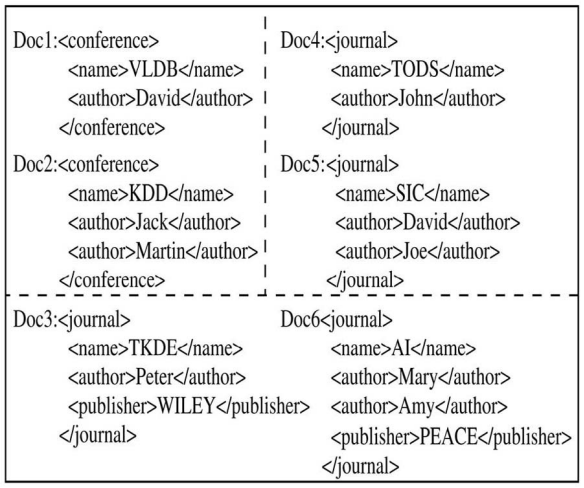


Fig. 2. Documents.

conference

doc_id	parent_id	self_id	value
1	null	1	null
2	null	1	null

journal

doc_id	parent_id	self_id	value
3	null	1	null
...	...	...	...
6	null	1	null

publisher

doc_id	parent_id	self_id	value
3	1	4	WILEY
6	1	5	PEACE

name

doc_id	parent_id	self_id	value
1	1	2	VLDB
...	...	...	...
6	1	2	AI

author

doc_id	parent_id	self_id	value
1	1	3	David
...	...	...	...
6	1	4	Amy

Fig. 3. Schema A.

2 Motivation

2.1 Background

Many query languages proposed for semistructured data can be used on XML documents, e.g., Lorel [15], XQL, and XQuery [22]. A semistructured query can be decomposed into a set of *path expressions* using XPath [21]. The query results are derived by joining the intermediate results of the path expressions. To simplify our discussion, without loss of generality, we assume the set of path expressions are either *absolute paths*, (in the form of $/a/b/\cdots/c/d$), or *relative paths*, (in the form of $//a/b/\cdots/c/d$). Absolute paths start at the root of the document while relative paths can start anywhere in the tree structure. Also, we assume the path expressions do not include wildcards ("*"), "//" (ancestor/descendent relationship), and *function* operators. We call such path expressions *simple path expressions*.[2] The following is an example of a semistructured query (XQuery) which returns all the authors who have written at least one conference paper and one journal article. The two XPath expressions in the first two "for" statements return the conference authors and the journal authors separately. A join (the fourth statement) on the authors returned gives the final results.

```
for $e1 in document("all.xml")/conference/author
  for $e2 in document("all.xml")/journal/author
    return $e1/text()
    where $e1/text()=$e2/text()
```

2.2 Motivating Example

In order to store XML documents with relational databases, XML documents need to be flattened and fragmented before they are stored in tables. Hence, possibly multiple tables must be joined in order to answer path queries. In Fig. 2, there are six XML documents forming three partitions (clusters) separated by the dashed lines, all of which conform to the following DTD:

$$< !ELEMENT\ conference\ (name, author)^* >$$
$$< !ELEMENT\ journal\ (name, author, publisher)^* > .$$

There are several methods for mapping XML documents to relational tables. Each one has a different technique for rewriting semistructural queries to SQL. To simplify our discussion, we use the mapping and rewriting method in [19].[3] Fig. 3 presents Schema A for storing all the six documents together, which is generated according to [19].[4] The mapping method tries to include as many descendants of an element as possible into a single relation. It also creates a relation for each element because an XML document can be rooted at any element in a DTD. The value of $self\_id$ is the linear order of the elements in a document. An element of a document can be identified by its doc_id and $self\_id$. Fig. 4 shows Schema B, in which each partition has its own set of tables. Schema B is, in fact, a *projection* of Schema A on the partitions generated in a simple way: For each partition, we create the same set of tables as that in Schema A and rename them by appending the partition id. The documents in each partition are inserted into these tables as if the tables in Schema A were projected into the partition. Empty tables are removed. Suppose two queries q_1 and q_2 (in XQuery format) are submitted to both Schemas A and B:

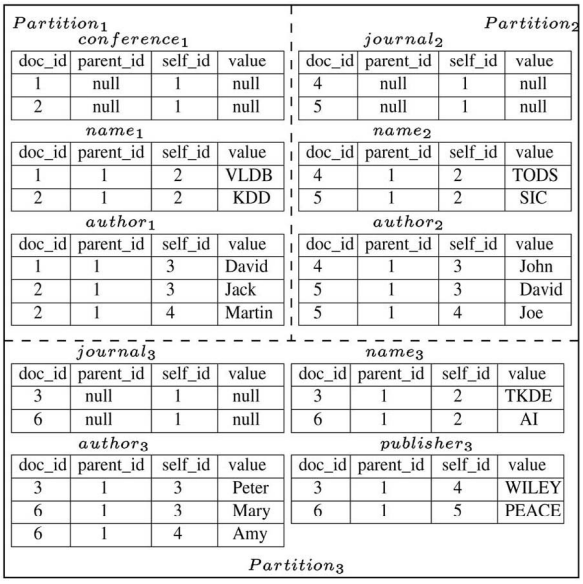
- q_1: find authors and publishers for all journal papers and
- q_2: find authors who have written at least one journal article and one conference paper.

Fig. 5 shows these four queries in SQL. Notice that the structure of q_1 is the same on both Schemas A and B . In Schema A, we need to join the tables $journal$, $author$, and $publisher$. In Schema B, we only need to join the smaller

2. If we modify the definition of s-graph in Section 3, we can extend path expressions to include general relative paths.

3. Since the problem we are studying is on clustering XML documents, the choice of mapping and rewriting method does not affect the generality of our result. As will be seen later on, other mapping methods can also be used for mapping and rewriting. (We have also tested the mapping method in [20] in Section 5.)

4. Some attributes were not listed in Fig. 3 for simplicity.



$Partition_1$

$conference_1$

doc_id	parent_id	self_id	value
1	null	1	null
2	null	1	null

$name_1$

doc_id	parent_id	self_id	value
1	1	2	VLDB
2	1	2	KDD

$author_1$

doc_id	parent_id	self_id	value
1	1	3	David
2	1	3	Jack
2	1	4	Martin

$Partition_2$

$journal_2$

doc_id	parent_id	self_id	value
4	null	1	null
5	null	1	null

$name_2$

doc_id	parent_id	self_id	value
4	1	2	TODS
5	1	2	SIC

$author_2$

doc_id	parent_id	self_id	value
4	1	3	John
5	1	3	David
5	1	4	Joe

$Partition_3$

$journal_3$

doc_id	parent_id	self_id	value
3	null	1	null
6	null	1	null

$name_3$

doc_id	parent_id	self_id	value
3	1	2	TKDE
6	1	2	AI

$author_3$

doc_id	parent_id	self_id	value
3	1	3	Peter
6	1	3	Mary
6	1	4	Amy

$publisher_3$

doc_id	parent_id	self_id	value
3	1	4	WILEY
6	1	5	PEACE

Fig. 4. Schema B.

tables $journal_3$, $author_3$, and $publisher_3$. Thus, the cost of running q_1 in Schema B is much smaller than in Schema A.

Let us analyze the cost of q_2 which joins documents in different clusters. The journal articles are separated into $Partition_2$ and $Partition_3$, while conference papers are all in $Partition_1$. The SQL code for q_2 in Schema B consists of two sections of SQL codes connected by a *union all* clause, and each section of SQL code is exactly the same as that in Schema A. The join between *journal* and *author* in Schema A has been transformed into two joins in Schema B: the join between $journal_2$ and $author_2$ and the join between $journal_3$ and $author_3$. The joins between

1. $journal_2$ and $author_1$,
2. $journal_2$ and $author_3$,
3. $journal_3$ and $author_1$, and
4. $journal_3$ and $author_2$ are all eliminated.

This is due to two reasons: 1) we need not join journals with authors of conference papers and 2) we need not join a journal with authors of another journal. This join cost reduction accelerates query processing (the improvement depends on the implementation of the RDBMS). We call this an improvement related to the *intradocument joins* because the journal-author join is to recover an element-subelement relationship within a document.

Note that no additional join cost is introduced due to the clustering. For example, in Schema B, we need to join the *author* tables in different partitions. However, this join already exists in Schema A. In fact, the self-join of the *author* table in Schema A is transformed into two joins in Schema B: the join between $author_1$ and $author_2$ and the join between $author_1$ and $author_3$. The sizes of the tables involved have decreased and the processing does not incur extra cost in Schema B.

Summarizing, we have illustrated how a query on Schema A can be mapped into Schema B, on which the query requires less join cost in its processing than on Schema A. In the rest of this paper, given a relational schema and a partitioning (clustering) of a set of XML documents, we use the term *partitioned schema* to represent the schemas in the partitions which are projections of the tables in the original schema (*unpartitioned schema*) into the partitions as described in Fig. 4.

q_1 on Schema A	q_2 on Schema A
select a.value, d.value	select a1.value
from *author* a,	from *conference* c, *journal* j,
journal j, *publisher* d	*author* a1, *author* a2
where	where a1.doc_id=c.doc_id
a.doc_id=j.doc_id	and a1.parent_id=c.self_id
and a.parent_id=j.self_id	and a2.doc_id=j.doc_id
and d.doc_id=j.doc_id	and a2.parent_id=j.self_id
and d.parent_id=j.self_id	and a1.value=a2.value
q_1 on Schema B	q_2 on Schema B
select a.value, d.value	select a1.value
from $author_3$ a,	from $conference_1$ c, $journal_2$ j,
$journal_3$ j, $publisher_3$ d	$author_1$ a1, $author_2$ a2
where	where a1.doc_id=c.doc_id
a.doc_id=j.doc_id	and a1.parent_id=c.self_id
and a.parent_id=j.self_id	and a2.doc_id=j.doc_id
and d.doc_id=j.doc_id	and a2.parent_id=j.self_id
and d.parent_id=j.self_id	and a1.value=a2.value
	union all
	select a1.value
	from $conference_1$ c, $journal_3$ j,
	$author_1$ a1, $author_3$ a3
	where a1.doc_id=c.doc_id
	and a1.parent_id=c.self_id
	and a3.doc_id=j.doc_id
	and a3.parent_id=j.self_id
	and a1.value=a3.value

Fig. 5. SQL codes of q_1 and q_2.

Clustering documents by structural information does not eliminate the fragmentation problem; it alleviates it by reducing the join cost, in particular, the cost on intradocument joins. The schema design in our example follows the technique in [19]. If we use the structure mapping techniques in [20], the effect would be even better. The experimental results in Section 5 show the performance gain using different mapping techniques.

3 Clustering of XML Documents

After establishing a motivation to cluster XML documents, we turn our attention to the development of an effective clustering algorithm. In this section, we define a method to summarize XML documents such that a simple and efficient similarity metric can be applied. Then, we show how this metric can be used in combination with a clustering algorithm to divide a large collection of XML documents into groups according to their structural characteristics. Although our definitions and methodology assume a database of XML documents, they can be seamlessly applied for any collection of semistructured data.

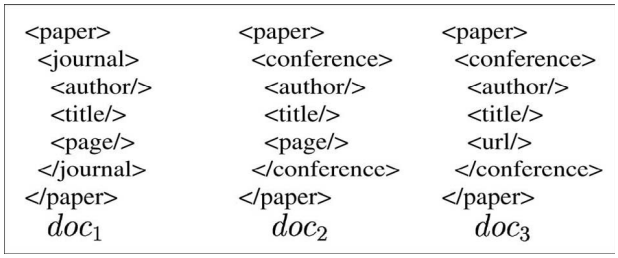


Fig. 6. Differences in elements.

3.1 Similarity between XML Documents

Because semistructured data has not been a popular data format until the appearance of XML files, conventional clustering techniques do not have special emphasis on this data type. What would be a proper approach for clustering semistructured data? Let us consider some options for defining the *similarity* between XML documents. We can treat the elements of a document as attributes and convert the document into a transaction of binary attributes. Jaccard Coefficient or Cosine function [18], among various other similarity measures, can be used to measure the similarity between documents. However, many structurally different documents have almost the same set of elements. In Fig. 6, doc_1 and doc_2 have only one different element, but they should be in two different clusters according to the semantics, assuming that many applications would be interested in posting queries to journal and conference papers separately. In other words, doc_2 and doc_3 should be separated from doc_1 to form a cluster.

Since XML documents can often be modeled as node-labeled trees, another option would be to use *tree distance* [24]to measure their similarity. In [17], besides node relabeling, node insertion, and node deletion, the tree distance method is refined to allow insertion and deletion of subtrees, which makes it more feasible to calculate the similarity of document trees. However, the cost of computing the tree distance between two documents is high (quadratic to their sizes), rendering it unsuitable for a collection of large documents.

Nierman and Jagadish [17] suggest assigning different costs to the tree editing operators. Practically, there is no simple way to do this assignment such that the resulting clustering would perform well. For example, in Fig. 7, if subtree deletion costs less than subtree renaming, then $dist(doc_1, doc_2) < dist(doc_1, doc_3)$. In the opposite case, we would have $dist(doc_1, doc_2) > dist(doc_1, doc_3)$. The situation may be even worse, if we cannot find a proper cost

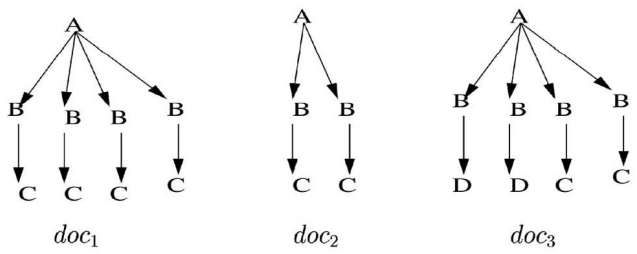


Fig. 7. Tree distances between documents.

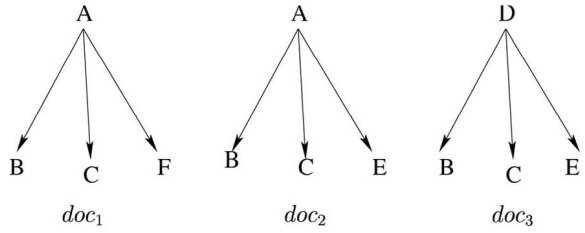


Fig. 8. Tree distances between documents.

assignment for all the documents; there may exist different assignments for different subtrees.

Besides that, in some cases, it may not be possible to distinguish documents that are structurally different using the edit distance. In Fig. 8, the tree distance between doc_1 and doc_2 will be the same as that between doc_2 and doc_3, because only one relabeling operation is required in both cases to transform the "source" tree into the "destination" tree. If we cluster doc_1 and doc_2 together, the DTD covering them would be $< !ELEMENTA(B, C, E, F)^* >$ which has only four edges. On the other hand, the DTD covering doc_2 and doc_3 would be $< !ELEMENTA(B, C, E)^* >$ and $< !ELEMENTD(B, C, E)^* >$, which has a total of six edges. Notice that the documents in the latter case should be better clustered separately because A and D probably are two different object types such as journal and conference paper in the DBLP database. This simple example shows that the tree distance based method may not be able to distinguish structural differences in some cases. In the following, we propose a new notion to measure the similarity between XML documents.

Definition 1. *Given a set of XML documents C, the **structure graph** (or **s-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.*

Notice that the structure graph defined here is different from the DTD graph in [19]. The structure graphs are derived from XML documents, not from their DTD. For example, the s-graph $sg(doc_1, doc_2)$ of two documents doc_1 and doc_2 is the set of nodes and edges appearing in either document, as illustrated in Fig. 9. In the same manner, a path expression q can be viewed as a graph (N, E), where N is the set of elements or attributes in q and E is the set of

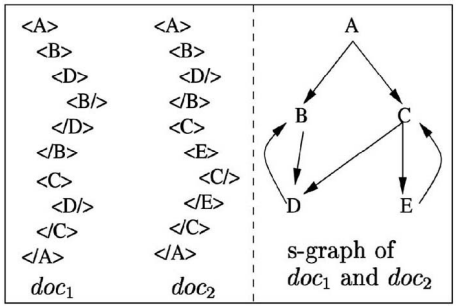


Fig. 9. An example s-graph.

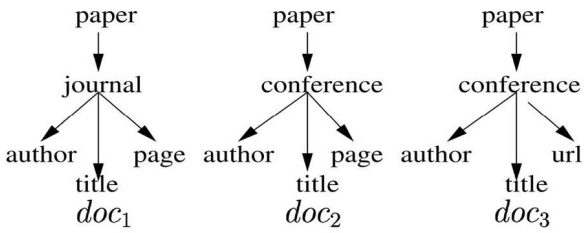


Fig. 10. S-graph-based similarity.

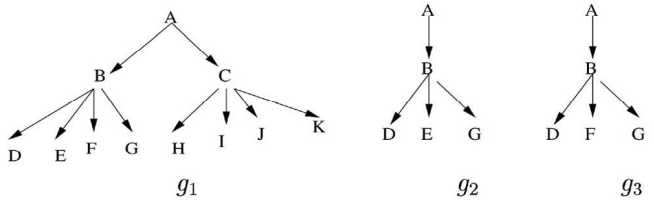


Fig. 11. Subcluster inside a cluster.

element-subelement or element-attribute relationships in q. Given a path expression q which has an answer in an XML document X, the directed graph representing q is a subgraph in the s-graph of X. For simplicity, we will denote the graph of a path expression q also by q.

Theorem 1. *Given a set of XML documents C, if a path expression q has answer in some document in C, then q is a subgraph of $sg(C)$. Also, $sg(C)$ is the minimal graph that has this property.*

The minimality property of $sg(C)$ is derived from the observation that any proper subgraph of $sg(C)$ will not contain all path expressions that can be answered by any document in C. Thus, the s-graph of C is a "compact" representation of the documents in C with respect to the path expressions. Note that the construction of $sg(C)$ can be done efficiently by a single scan of the documents in C, provided that each document fits into memory.

Corollary 1. *Given two sets of XML documents C_1 and C_2, if a path expression q has an answer in a document of C_1 and a document of C_2, then q is a subgraph of both $sg(C_1)$ and $sg(C_2)$.*

It follows from Corollary 1 that, if the structure graphs of two sets of documents have few overlapping edges, then there are very few path expressions that can be answered by both of them. Hence, it is reasonable to store them in separate sets of tables. The following distance metric is derived from this observation.

Definition 2. *For two XML documents C_1 and C_2, the distance between them is defined by*

$$dist(C_1, C_2) = 1 - \frac{|sg(C_1) \cap sg(C_2)|}{max\{|sg(C_1)|, |sg(C_2)|\}},$$

where $|sg(C_i)|$ is the number of edges in $sg(C_i), i = 1, 2$ and $sg(C_1) \cap sg(C_2)$ is the set of common edges of $sg(C_1)$ and $sg(C_2)$.

It is straightforward to show that $dist(C_1, C_2)$ is a metric [3]. If the number of common element-subelement relationships between C_1 and C_2 is large, the distance between the s-graphs will be small, and vice versa. In Fig. 10, we have the s-graphs of three documents. Using the metric in Definition 2, we would have $dist(\{doc_2\}, \{doc_3\}) = 0.25$ and $dist(\{doc_1\}, \{doc_2\}) = dist(\{doc_1\}, \{doc_3\}) = 1$. A clustering algorithm would merge doc_2 and doc_3, and leave doc_1 outside. This shows that the metric is effective in separating documents that are structurally different. It is important to point out here that using s-graphs allows the application of the same metric on documents as well as sets of documents, a property that simplifies the clustering process.

The metric has another nice characteristic. It prevents an s-graph which is a subgraph of another s-graph from being "swallowed," if they should form two clusters. In Fig. 11, we have three s-graphs such that $dist(\{g_2\}, \{g_3\}) = 0.25$ and $dist(\{g_1\}, \{g_2\}) = dist(\{g_1\}, \{g_3\}) = 0.6$. A clustering algorithm with this metric can separate the documents associated with g_2 and g_3 from those with g_1, even though both g_2 and g_3 are subgraphs of g_1. Following the same reason, outliers with large s-graphs would be prevented from wrongfully swallowed nonoutliers whose s-graphs are subgraphs of the outliers' s-graphs.

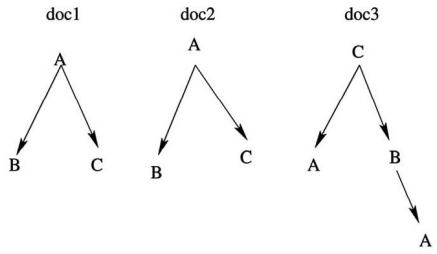
3.2 A Framework for Clustering XML Documents

Our purpose is to cluster XML files based on their structure. We achieve this by summarizing their structure in s-graphs and using the metric in Definition 2 to compute the clusters. Our approach is implemented in two steps:

- Step 1. Extract and encode structural information: This step scans the documents, computes their s-graphs, and encodes them in a data structure.
- Step 2. Perform clustering on the structural information: This step applies a suitable clustering algorithm on the encoded information to generate the clusters.

Initially, the s-graphs of all the documents are computed and stored in a structure called SG. An s-graph can be represented by a bit string which encodes the edges in the graph. Each entry in SG has two information fields: 1) a bit string representing the edges of an s-graph and 2) a set containing the ids of all the documents whose s-graphs are represented by this bit string. Obviously, s-graphs with no documents corresponding to them are not contained in SG. Fig. 12 shows an example with three documents. Since many documents may have the same s-graph, the size of SG is much smaller than the total number of documents. In general, SG should be small enough to fit into the memory. In the extreme case, a general approach such as sampling can be used. Once SG is computed, clustering is performed on the bit strings. Therefore, we transform the problem of clustering XML documents into clustering a smaller set of bit strings, which is fast and scalable.

In our framework, we have separated the encoding and extraction of the structural information from the clustering part. Many appropriate algorithms could be used to cluster the s-graphs. However, it is not natural to treat the s-graph information as numerical data because it is encoded as binary attributes with only two domain values. Therefore, an appropriate clustering algorithm on categorical data



SG

	entry 1	entry 2
A->A	0	0
A->B	1	0
A->C	1	0
B->A	0	1
B->B	0	0
B->C	0	0
C->A	0	1
C->B	0	1
C->C	0	0
	{doc1, doc2}	{doc3}

Fig. 12. An example of s-graph encoding.

would be a better choice. In the following, we will explain how we have applied a representative categorical clustering algorithm (ROCK [12]) on the s-graphs. In Section 6, we also discuss our experience in using a density-based clustering algorithm to cluster the s-graphs for comparision purpose (DBSCAN [9]).

3.3 The S-GRACE Algorithm

S-GRACE is a hierarchical clustering algorithm on XML documents, which applies ROCK [12] on the s-graphs extracted from the documents. As pointed out in [12], pure distance-based clustering algorithm may not be effective on categorical or binary data. ROCK tries to handle the case that, even though some data points may not be close enough in distance but they share a large number of common neighbors, it would be beneficial to consider them belonging to the same cluster. This observation would help to cluster s-graphs which a share large number of common neighbors.[5] The pseudocode of S-GRACE is shown in Fig. 13. The input D is a set of XML documents. In the beginning, as discussed in Section 3.2, the s-graphs of the documents are computed and stored in the array SG. The procedure $pre\_clustering$ (line 1) creates SG from D using hashing. Two s-graphs in SG are *neighbors* if their distance is smaller than an input threshold θ. $Compute\_distance$ (line 2) computes the distance between all pairs of s-graphs in SG and stores them in the array $DIST$.

ROCK exploits the $link$ property in selecting the best pair of clusters to be merged in the hierarchical merging process. Given two s-graphs x and y in SG, $link(x, y)$ is the number of common neighbors of x and y, where an s-graph z is a neighbor of x, if $dist(x, z) \leq \theta$, (θ is a given distance threshold). In S-GRACE, the number of neighbors of an s-graph is weighted by the number of documents it represents. For a pair of clusters C_i, C_j, $link[C_i, C_j]$ is the number of cross links between elements in C_i and C_j, (i.e., $link[C_i, C_j] = \sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r)$). Also, a $goodness\ measure$ $g(C_i, C_j)$ between a pair of clusters C_i, C_j is defined by

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_i)^{1+2f(\gamma)} - n_i^{1+2f(\gamma)} - n_j^{1+2f(\gamma)}},$$

```
/*Input  D: a set of XML documents*/
/*Input  θ: a similarity threshold */
/*Input  α: an integer */
/*Input  β: a control parameter for labeling outliers */
/*Input  k: a control parameter for the number of clusters */
/*Output  Q: a set of clusters; O: outliers set*/
1)  SG=pre_clustering(D);
2)  DIST=compute_distance(SG);
3)  LINK=compute_link(DIST, SG, θ);
4)  O=remove_outlier(LINK, SG, β);
5)  for each s ∈ SG do
6)    q[s] = build_local_heap(LINK, s);
7)  Q=build_global_heap(SG, q);
8)  while size(Q) > α × k do {
9)    u = extract_max(Q);
10)   v = max(q[u]);
11)   delete(Q, v);
12)   w = merge(u, v);
13)   for each x ∈ q[u] ⋃ q[v] do {
14)     LINK[x,w] = LINK[x,u] + LINK[x,v] ;
15)     delete(q[x], u); delete(q[x], v);
16)     insert(q[x], w, g(x,w)); insert(q[w], x, g(x,w));
17)     update(Q, x, q[w]);
18)   }
19)   insert(Q, w, q[w]);
20)   deallocate(q[u]); deallocate(q[v]);
21) }
22) O = O ⋃ remove_outlier(LINK, Q, q, β);
23) second_cluster(LINK, Q, q, k);
```

Fig. 13. S-GRACE.

5. We need to point out that the novelty here is the extraction of proper information in the form of s-graphs as a base for clustering. ROCK is by no means the only available method for clustering s-graphs, but it is the more preferrable one as shown by our experimental result.

where n_i and n_j are the number of documents in C_i and C_j, respectively, and $f(\gamma)$ is an index on the estimation of number of neighbors for C_i and C_j [12]. In fact, the denominator is the expected number of cross links between the two clusters. $Compute\_link$ (line 3) computes the $link$ value between all pairs of s-graphs in SG and stores them in the array $LINK$. $Remove\_outlier$ then removes the clusters that have no neighbors. Initially, each entry in SG is a separate cluster. For each cluster i, we build a local heap $q[i]$ and maintain the heap during the execution of the algorithm. $q[i]$ contains all clusters j such that $link[i, j]$ is nonzero. The clusters in $q[i]$ are sorted in decreasing order by the goodness measures with respect to i. In addition, the algorithm maintains a global heap Q that contains all the clusters. The clusters i in Q are sorted in the decreasing order by their best goodness measures, $g(i, max(q[i]))$, where $max(q[i])$ is the element in $q[i]$ which has the maximum goodness measure.

The while loop (lines 8-21) iterates until only $\alpha \times k$ clusters remain in the global heap Q, where α is a small integer controlling the merging process. During each iteration, the algorithm merges the pair of clusters that have the highest goodness measure in Q and updates the heaps and $LINK$. The s-graph of a cluster obtained by merging two clusters contains the nodes and edges of the two source clusters (refers to Definition 1). Outside the loop, $remove\_outlier$ removes some more outliers from the remaining clusters which are small groups loosely connected to other nonoutlier groups. $Second\_cluster$ (line 23) further combines clusters until k clusters remain. It also merges a pair of clusters at a time. The purpose is to allow different control strategies to choose the pair of clusters to be merged in the last stage of S-GRACE.

In S-GRACE-1 (i.e., version 1 of the algorithm), we use the baseline strategy: The loop in $second\_cluster$ is the same as the while loop in lines 8-21. In S-GRACE-2, among the pairs of clusters with the top t normalized link values, we select and merge the pair that leads to a cluster with the minimum number of documents. This effectively will distribute the documents evenly among the clusters. In S-GRACE-3, among the pairs of clusters having the top t normalized link values, we select and merge the pair that has the minimum number of edges in the s-graph in the resulting cluster. This strategy makes the s-graph of the clusters as small as possible, and, consequently, reduces the number of clusters (partitions) that a path query would have to visit.

3.4 Complexity

Let N be the number of different elements and attributes in D. Since there are N^2 distinct edges, in the worst case, the size of the bit array representing a s-graph is bounded by N^2 bits. However, in typical cases, the number of distinct edges is much smaller than N^2. In all real data sets, we have checked this number and it is a small multiple of N, which means that the time required to scan $|D|$ documents and compute their bit-strings is $O(|D|\kappa N)$, where κ is a small constant. For example, for DBLP and NITF [13], κ is between three and four. In Section 5, Table 3 shows that the time to construct SG is usually less than 6 percent of the time of scanning all the documents.

TABLE 1
Input Parameters for Data Generation

Name	Meaning	Specification
CL	number of cluster DTDs	2-10
OL	DTD overlap upper bound	0.3-0.7
OR	outlier doc ratio	0.01-0.2
N	total number of docs	10,000-200,000
D	number of docs in a cluster	2,000-40,000
W	distribution of '*'	Poisson
P	distribution of '+'	Poisson
Q	probability of '?' to be 1	a number between 0 & 1
Max	distribution of doc depth	Poisson

Computing the distances between all pairs of initial s-graphs requires $O(m^2)$ time, where m is the number of distinct s-graphs in SG. Building the table $LINK$ generally requires $O(m^3)$. However, it can be reduced to $O(m^{2.37})$ [4]. Furthermore, we can expect that, on average, the number of neighbors λ for each s-graph will be small compared to m. Under this condition, an algorithm was designed in [12] that can further reduce the time complexity to $O(m^2\lambda)$.

Since updating local heaps for each merging requires $O(m \log m)$ time, the while loop of the algorithm requires $O(m^2 \log m)$ time. The last step ($second\_cluster$) is similar to the while loop, hence it also requires $O(m^2 \log m)$ time. Thus, the overall time complexity of S-GRACE is $O(|D|N^2 + m^{2.37})$ in the worst case and $O(|D|\kappa N + m^2\lambda)$ on the average.

SG stores the bit strings of s-graphs and document ids, so it requires $O(mN^2 + |D|)$ space. Both $DIST$ and $LINK$ require $O(m^2)$ space. The number of local heaps is $O(m)$ and each local heap contains $O(m)$ entries (the size of each entry is $O(N^2)$). Thus, all local heaps consume $O(m^2N^2)$ space. The global heap stores $O(m)$ clusters and $|D|$ document ids, so it requires $O(mN^2 + |D|)$ space. Thus, the overall space complexity of S-GRACE is $O(m^2N^2 + |D|)$ in the worst case and $O(m^2\kappa N + |D|)$ on the average.

4 Query Rewriting

Most methods for storing XML data in relational tables provide some query rewriting mechanism to transform a semistructured query like XQuery to SQL. Following our discussion in Section 2.2, we can assume a relational schema (Schema A: Fig. 3) for storing the XML documents before the documents are partitioned. After partitioning, there is a new schema (Schema B: Fig. 4), which is the projection of Schema A on each partition. If a query has results in the documents within a partition, its processing on the tables of that partition is a straightforward query rewriting as illustrated by the example on query q_1 in Table 1.

If the query needs to integrate the results from multiple partitions, some issues in rewriting would need to be dealt with. Given a path expression of a query, we need to first identify all the partitions that contain it, i.e, those that may have answers. For this task, we have designed a Query Manager.

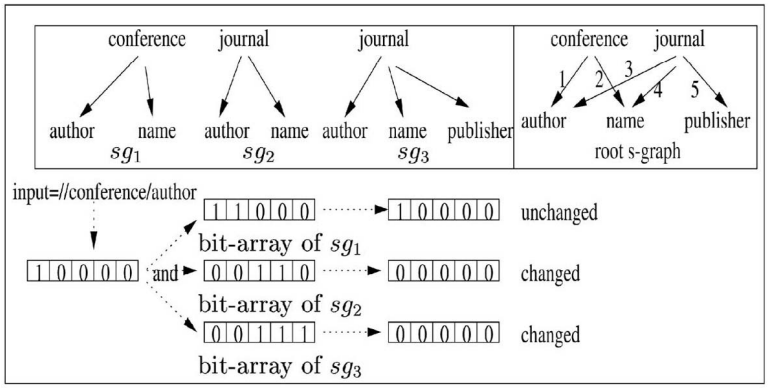


Fig. 14. Usage of Query Manager.

4.1 Query Manager

The task of the Query Manager (QM) is to determine the partitions that contain a given path expression. The QM maintains a *root* s-graph sg_r, and a set of bit arrays, one for each partition's s-graph. The root s-graph is the s-graph of the entire document set and is equal to the union of all the partitions' s-graphs. Each edge in sg_r is labeled by a predefined traversal order from 1 to n, where n is the number of edges in sg_r. For every partition, the size of the bit array for its s-graph is also n and the bits are also indexed by the traversal order in sg_r. In addition, all nodes in sg_r can be accessed from a hash-table.

Any path expression beginning with $/A$ (absolute path) or $//A$ (relative path) which does not contain a "$*$" or "//" can be transformed into a bit array of size n. The bitwise **AND** is applied to this bit array and those of the partitions. If the bit array of the path does not change after ANDing with a partition P_i, then P_i contains the path expression. Fig. 14 illustrates the functionality of the Query Manager. Observe that only the first partition (summarized by s-graph sg_1) contains results for the input query because it is the only graph that does not alter the query s-graph after the AND operation.

Now let us consider path expressions which begin with $/A$ or $//A$ and contain "$*$" or "//" followed by a descendant label B. We can evaluate them by first locating the node representing A in the root s-graph (using the hash table) and traversing sg_r starting from node A. While traversing the graph, relative path "//" and the wild card "$*$" are binded until B is reached. All the paths from A to B in sg_r can be identified to derive the query results. Notice that since the size of the sg_r is usually small, this process is not expensive. In addition, this method can avoid generating path queries with intermediate labels which do not appear in the document collection between A and B. Consider the root s-graph in Fig. 15. The path expression $A//B$ would generate two path queries A/B and $A/D/C/B$ because we can traverse from A to B via the two paths.

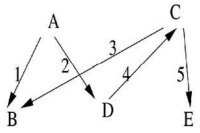


Fig. 15. An example of root s-graph.

4.2 Integrating Results from Different Partitions

With the help of QM, we can identify all partitions containing a path expression. If a semistructured query contains only a path expression, the rewriting is straightforward: union the results from all the partitions containing the path. If the query contains several related path expressions, some joins are inevitable. In Schema A, a query relating multiple path expressions will be rewritten into joins among tables in the schema. In Schema B, joins may be performed across partitions. As we have explained in Section 2, the tables in Schema B are projections of those in Schema A on the partitions. Therefore, each join in Schema A will correspond to several joins in Schema B. The SQL code of each join in Schema B is the same as that in Schema A except the tables are the projection of the corresponding tables on the partitions. For example, in the query q_2 in Fig. 5, $//journal/author$ is contained in two partitions while $//conference/author$ is in one partition. In order to join them, there should be $2 \times 1 = 2$ joins, and the table names in each join are changed accordingly as shown in Fig. 5.

4.3 Generalization of S-Graph

In Theorem 1, we have not considered general path expressions that include general ancestor/descendant relationship between two neighboring tags. In Section 4.1, we also discussed how to use the current s-graph definition to process such queries. Our approach essentially replaces a general path expression with a set of simple path expressions such that we can union the answers of the set of simple path expressions to give the answers of the general path expression. Also, each simple path expression is contained in the s-graph of the whole document collection. An alternative approach is to extend the s-graph to include not only parent/child edges, but also ancestor/descendant relationships that occur in documents. For example, we could encode $b//c$ relationships in the s-graphs as a special ancestor/descendant edges between b and c so that general path expressions such as $/a/b//c$ can be answered.

We have two choices on how to apply this s-graph generalization: either before the clustering or afterward. We recommend following the second choice because the

redundant information added to the s-graph in the first choice may make the size of the s-graph unnecessarily large. Extending the s-graph in each partition after the clustering would be enough to answer the relative path expression queries.

5 Performance Studies

In this section, we investigate the effectiveness, efficiency, and scalability of S-GRACE via experiments on both synthetic and real data. We generated the synthetic data using a real DTD. The real data are XML files from the DBLP database [5] containing computer science bibliography entries. Experiments are carried out in a computer with four Intel Pentium 3 Xeon 700MHZ processors and 4G memory running Solaris 8 Intel Edition.

5.1 Synthetic Data Generation

The XML GENERATOR in [8] is a tool, which generates XML documents based on a given DTD. It gives us very little control on the cluster distribution and similarity. Another method [2] generates complex XML documents, but also cannot control the similarity. Therefore, we had to build our own generator, which is a three-step process:

1. Given a DTD D, we randomly generate a set of sub-DTDs (smaller DTDs in D) in which the overlap between every pair of sub-DTDs is smaller than a threshold. A DTD can be represented by a graph G in which every element is a node and every element-subelement relationship is an edge. Assume that G_1 and G_2 are the graphs of sub-DTDs D_1 and D_2, respectively, the overlap between D_1 and D_2,

$$overlap(D_1, D_2) = (\text{number of common edges in } G_1 \text{ and } G_2)/(\text{minimum number of edges in } G_1 \text{ and } G_2).$$

   These sub-DTDs are used to generate clusters of documents. We call these sub-DTDs *cluster DTDs*.
2. We also create a set of sub-DTDs for the generation of outlier documents. We combine some pairs of cluster DTDs to form a set of *outlier DTDs*.
3. We generate documents based on the sub-DTDs generated in the first two steps.

Our synthetic data was produced using the NITF (News Industry Text Format) DTD [13] as seed DTD. The parameters used in the generation process are listed in Table 1. The first three parameters are defined to control the first and second steps of the generation process. The last six parameters are used to generate documents on a specific DTD.

A cluster DTD C is defined from the input DTD D in the following way. Starting from the root node r of the DTD graph of D, for each subelement s, if it is accompanied by "$*$" or "?," we randomly decide whether to include it in C or not. If it is accompanied by "+," then it is always included in C. If there are choices among several subelements of r, then they are included in C according to a random distribution. The same procedure is repeated on the new nodes until the number of elements and edges reach a threshold. To generate the set of cluster DTDs, the above procedure is repeated. A new DTD must satisfy the overlap constraint. The process terminates when there are enough DTDs.

The procedure that generates documents from a cluster DTD D is very similar. Starting from the root element r of D, for each subelement, if it is accompanied by "$*$" or "+," we decide how many times it should appear according to a distribution (such as Poisson). If it is accompanied by "?," the element appears or not by tossing a biased coin. If there are choices among several subelements of r, then their appearance in the document follows a random distribution. The process on the newly generated elements is repeated until some termination conditions have been reached.

TABLE 2
Clustering Accuracy as a Function of Database Size

N	Alg	CS	IS	SD	R
10K	1	0.79	0.13	56	0.024
10K	2	0.97	0.094	6	0.025
10K	3	0.97	0.094	6	0.025
20K	1	0.716	0.143	60	0.2
20K	2	0.906	0.019	5	0.019
20K	3	0.906	0.019	5	0.019
40K	1	0.61	0.159	1562	0.025
40K	2	0.964	0.1113	445	0.025
40K	3	0.964	0.1113	445	0.025
100K	1	0.716	0.191	4373	0.014
100K	2	0.931	0.15	128	0.011
100K	3	0.931	0.15	128	0.011
200K	1	0.4	0.33	13011	0.077
200K	2	0.922	0.1237	3030	0.025
200K	3	0.86	0.122	5674	0.025

5.2 Experiments on Synthetic Data

In this group of experiments, we compare the performances of S-GRACE-1, S-GRACE-2, and S-GRACE-3 (described in Section 3.3) on different sets of synthetic data. We have five control parameters in our data generation:

1. total number of documents,
2. number of clusters,
3. number of outliers,
4. overlapping between clusters, and
5. sizes of the clusters.

Due to space limitations, we present only the effects of the first three parameters in Tables 2, 4, and 5, respectively.

The first column of each table shows the parameter varied in the experiment. The second column indicates which version of S-GRACE is used, i.e., if the value is i, $1 \leq i \leq 3$, then S-GRACE-i is used. The third to sixth columns are four indicators which measure the goodness of the clusters discovered by S-GRACE. *CS* is a measure on the closeness between the clusters found by S-GRACE and

TABLE 3
Processing Cost of S-GRACE-2

number of doc	10K	20K	40K	100K	200K
preprocessing time (sec)	67.5	138.2	271.3	667.9	1363.7
no. of s-graphs	1773	2679	3903	5470	7728
creation of SG (sec)	2.5	5.8	12.6	24	48.2
clustering time (sec)	230	541	1215	2210	4479

TABLE 4
Clustering Accuracy Varying the Number of Clusters

CL	Alg	CS	IS	SD	R
6	1	0.64	0.2	75	0.22
6	2	0.82	0.15	68	0.22
6	3	0.716	0.15	84	0.22
7	1	0.632	0.177	420	0.027
7	2	0.858	0.158	65	0.027
7	3	0.858	0.158	65	0.027
8	1	0.79	0.137	78	0.009
8	2	0.902	0.114	68	0.009
8	3	0.874	0.12	87	0.009
9	1	0.807	0.134	89	0.011
9	2	0.914	0.12	62	0.011
9	3	0.914	0.132	67	0.011
10	1	0.66	0.155	130	0.009
10	2	0.85	0.139	65	0.009
10	3	0.85	0.139	65	0.009

the clusters in the data. For each found cluster C, we measure the similarity between it and the cluster DTD in the data generation which has the highest similarity to C. (We use the term *similarity* between two clusters, C_1 and C_2, to denote the quantity $1 - dist(C_1, C_2)$ as defined in Definition 2.) The value of *CS* is the average similarity of the found clusters with the corresponding DTDs. *IS* is the average similarity over all pairs of clusters found by S-GRACE. *SD* is the standard deviation of the number of documents in the clusters found by S-GRACE. Finally, *R* is the ratio of outlier documents found by S-GRACE. A good clustering technique would result in a large *CS* (close to 1) and a small *IS* and *SD* (close to 0). The value of *R* should be close to the outlier ratio in the data generation.

5.2.1 Varying the Number of Documents

In this experiment, we test the scalability of our algorithms to the database size (N), which varies from 10K to 200K documents. The data are generated using the following parameters: $\mathrm{CL} = 5$, $\mathrm{OL} = 0.3$, $\mathrm{OR} = 0.02$, and all cluster DTDs generate the same number of documents with $\mathrm{D} = 2\mathrm{K}, 4\mathrm{K}, 8\mathrm{K}, 20\mathrm{K}, 40\mathrm{K}$. We input $k = 4, 5, 6, 7, 8$ to our algorithm and only show the result of $k = 5$ in Table 2 because $k = 5$ gives the best values of CS, IS, SD, and R.All four indicators reveal that S-GRACE-2 and S-GRACE-3 are more effective than S-GRACE-1. S-GRACE-2 has a slight edge on S-GRACE-3. The CS values are very high which shows that both S-GRACE-2 and S-GRACE-3 are very accurate in discovering clusters.

Table 3 shows the processing cost of S-GRACE-2 as a parameter of database size. The preprocessing cost is the time to read the documents and turn them into a hash table of bit arrays. The creation time of SG involves the scanning of the hash table to create SG. The document size in this experiment ranged from 0.5Kb to 20Kb with an average of 2.5Kb.

TABLE 5
Clustering Accuracy Varying the Outlier Ratio

OR	Alg	CS	IS	SD	R
0.01	1	0.82	0.112	78	0.012
0.01	2	0.92	0.106	7	0.012
0.01	3	0.92	0.106	7	0.012
0.05	1	0.8	0.11	78	0.05
0.05	2	0.914	0.115	9	0.05
0.05	3	0.914	0.115	9	0.05
0.10	1	0.86	0.147	40	0.115
0.10	2	0.89	0.127	21	0.115
0.10	3	0.89	0.127	21	0.115
0.15	1	0.427	0.328	141	0.135
0.15	2	0.83	0.135	51	0.135
0.15	3	0.58	0.25	69	0.135
0.20	1	0.58	0.23	192	0.2
0.20	2	0.785	0.212	71	0.2
0.20	3	0.66	0.19	101	0.21

5.2.2 Varying the Number of Clusters

In this experiment, we test the robustness of S-GRACE to the number of clusters. The number of clusters varies from six to 10. The data are generated with the following parameters: $\mathrm{CL} = 6, 7, 8, 9, 10$, $\mathrm{OL} = 0.4$, $\mathrm{OR} = 0.02$, and $\mathrm{D} = 5\mathrm{K}$. k takes values from $\{4, 5, 6, 7, 8, 9, 10\}$ for each data set. Table 4 shows the results when k is equal to CL. In this case, we get the best values of CS, IS, SD, and R. Again, S-GRACE-2 performs slightly better than S-GRACE-3. The baseline algorithm S-GRACE-1, as expected, has the worst accuracy.

5.2.3 Varying the Ratio of Outliers

In this experiment, we validate the performance of S-GRACE varying the ratio of outliers. The data are generated using the following parameters: $\mathrm{CL} = 5$, $\mathrm{OL} = 0.3$, $\mathrm{OR} = 0.01, 0.05, 0.10, 0.15, 0.20$, and $\mathrm{D} = 5\mathrm{K}$. Again, k takes values from $\{4, 5, 6, 7, 8, 9, 10\}$ and only the result of $k = 5$ is shown in Table 5. It is clear that the ratio of outliers, R, discovered by S-GRACE is very close to the ratio of outliers, OR, used in the data generation. This shows that S-GRACE is quite effective in the discovery of the outliers.

Besides the above experiments, we also tested the robustness of the algorithms to changes in the overlap between cluster DTDs and the size of the clusters. Again S-GRACE-2 usually gives us the best result in terms of accuracy. S-GRACE-3 performs well in a few cases, while the S-GRACE-1 is always worse than the other two.

5.3 S-GRACE-2 on Real Data and Query Enhancement

In the previous section, we saw that S-GRACE-2 performs better than the other two variants in most cases. Hence, we adopt it as the standard implementation of S-GRACE and test the performance enhancement it introduces in query processing. The data set we use is the XML DBLP records database [5], which contains about 200,000 XML documents composed of 36 elements. Most of the documents are described by either $inproceedings$ or $article$.[6] Others are postgraduate students' theses, white papers, etc. All

6. $Inproceedings$ and $article$ are two elements in the DTD of DBLP representing conference papers and journal articles, respectively.

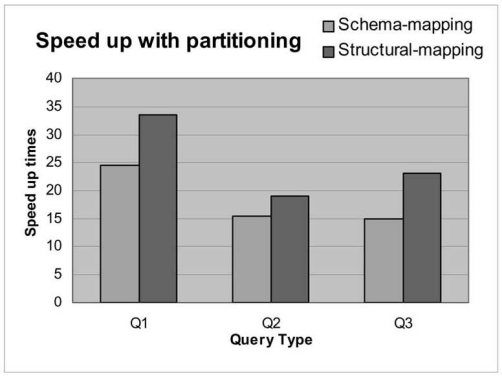


Fig. 16. Speed up ratios for Q1, Q2, and Q3.

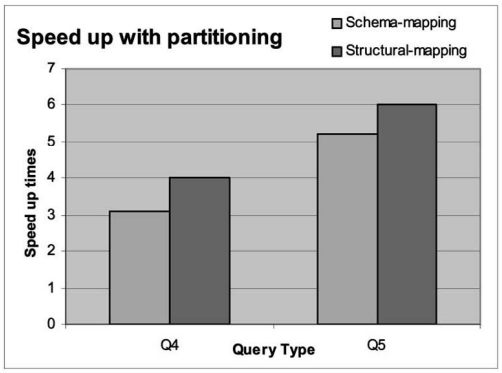


Fig. 17. Speed up ratios for Q4 and Q5.

documents contain elements such as $author$, $title$, and $year$. Overlap among documents' elements is a common scenario.

Our goal is to test whether a partitioned schema from S-GRACE brings in better query performance than the unpartitioned schema. We defined five types of queries based on the structure of existing documents. The first three are written in XPath, and the last two in XQuery. The five query classes are:

- Q1: $/A_1/A_2/\cdots/A_k$; all possible absolute XPaths in the documents.
- Q2: $/A_1/A_2/\cdots/A_k[text() = "value"]$; the same as Q1 except that one additional requirement is added to make sure the text value of the last element is equal to "value," which is a string randomly selected from the real data.
- Q3: $/A_1/A_2/\cdots/A_k[contains(.,"substring")]$; same as Q1 except that the additional requirement is to make sure that a randomly picked "substring" is contained in the text value of the last element.
- Q4: find the titles of articles published in the *VLDB Journal* in 1999.
- Q5: find the names of authors which have at least one journal article and one conference paper.

Because path expressions are the basic unit in composing XML queries, we used the first three queries to test the performance of processing path expressions. Comparatively, the resulting set of Q1 is very large, while that of Q2 is small and the size of the return of Q3 is somewhere in between. Hence, they can test our approach on queries with different selectivity. Q4 and Q5 are defined to test the joins among path expressions. Joins in Q4 occur only inside clusters while joins in Q5 are applied across clusters.

The RDBMS we used is the Oracle 8i Enterprise Edition release 8.1.5. All the above five queries are translated to SQL and executed on the RDBMS. S-GRACE is used to generate the clusters that define the partitioned database schema. Based on the experimental results, the parameters of S-GRACE (see Fig. 13) are set to: $\theta = 0.2$, $\alpha = 100/k$, and $k = 4, 5, 6, 8$. The clustering result depends on k, the number of expected clusters. For each value of k, we compared the overlap between the clusters found. The higher the overlap, the more the path expressions are that have answers in multiple clusters. In order to filter as many as documents while processing path expressions, we need a k that results in a low overlap. The average overlap is the lowest when $k = 4$. We used the four clusters found in this case to partition the documents in order to evaluate the query performance.

During the clustering, the parsing and construction of the array SG took 1,361 seconds for a total of 200,000 documents, the number of distinct s-graphs in SG is 233. Therefore, clustering is very fast and takes less than two seconds (we have excluded the element-attribute relationships in the s-graphs).

We use the schema mapping technique in [19] to create a schema for storing the documents. Tables in the schema are then projected into the partitions from the clustering to create the partitioned schema. Performance of the queries are compared between the original unpartitioned schema and the partitioned schema. We then use the structure mapping technique in [20] to create a schema and repeat the performance comparison.

The four clusters returned from S-GRACE-2 have the following properties: The first cluster contains about 80,000 $article$ documents and its s-graph contains 14 elements: dblp, article, author, title, pages, year, journal, volume, number, month, url, ee, cdrom, and cite. The second cluster contains about 73,000 $inproceedings$ documents and its s-graph contains eight elements: dblp, inproceedings, author, title, booktitle, pages, year, url. The third cluster contains about 39,000 $inproceedings$ documents and its s-graph contains 16 elements; besides the eight tags that appear in the second cluster, it contains another eight tags: ee, cdrom, cite, crossref, sup, sub, i, and number. The fourth cluster is the outlier set, which has about 7,000 documents and its s-graph contains 36 elements. We should mention that the s-graph of the second cluster is entirely contained in the third cluster—not only all the nodes, but also all the edges. It would be difficult to spot these two clusters by manual inspection. This clearly demonstrates the effectiveness of S-GRACE in XML document collections like DBLP.

Figs. 16 and 17 show the query performance speed-up when the original schema is compared with the partitioned schema. Each distinct path expression conforming to Q1, Q2, and Q3 in the documents is submitted as a query to the original schema and the partitioned schema. The speed-up ratios for each query type are averaged and the results are plotted in Fig. 16. The average improvement on path expressions is quite large. We should mention here that the

TABLE 6
Query Response Time

Method	Q1	Q2	Q3	Q4	Q5
UP-Sa	12,713	17,682	23,720	42,248	92,3691
P-Sa	2,460	4,787	6,952	13,462	177,697
UP-St	137,336	2,693	29,934	54,984	4,560,031
P-St	12,344	533	4,500	13,527	761,327

speed-up ratio of the distinct path expressions in Q1 in fact ranges from 1.4 to 44 because some paths may need to join more tables than the others. The improvement for Q4 and Q5 in Fig. 17 is smaller. Comparing the queries in Q4 and Q5 with Q1, observe that the path expressions in Q4 and Q5 involve less joins than those of Q1, on the average. This reduces the improvement ratio. In fact, we observe that the speed-up of the individual XPaths in Q4 and Q5 are, in general, less than two.

Table 6 summarizes the average query response times for Q1 to Q5 in milliseconds. In the first column, methods "UP-Sa" and "P-Sa" denote the unpartitioned original schema and the partitioned schema, respectively, with schema mapping [19]. "UP-St" and "P-St" are corresponding cases for structure mapping [20].

Observe that in both Figs. 16 and 17, the speed-up of structure-mapping method is always larger than that of the schema-mapping method. This is because, in the structure mapping, only four tables are used to keep the content of a document. Except the $Path$ table, the sizes of the other three tables are very large. For example, the number of tuples in the $text$, $element$, and $attribute$ tables are 1,918,589, 2,244,838, and 273,841, respectively. The join among them in the original schema is very expensive. In the partitioned schema, the tables become much smaller. Hence, the speed up is obvious and larger.

Our experiments on the synthetic data show that S-GRACE is effective in identifying clusters and scalable. The results on the DBLP data reveal several additional advantages of the clustering algorithm. First, it is fast, requiring only one scan of the documents. The time for clustering on the array SG is $O(m^2 \log m)$ and m is small, in general. Second, after applying S-GRACE to partition the database schema, the query processing cost drops dramatically since many unnecessary joins between irrelevant parts of the original tables are avoided. Finally, a qualitative benefit of the clustering method is revealed; it can discover subclusters, which are not easy to spot manually.

5.4 Comparison with Tree-Distance-Based Algorithm

Besides studying the performance of S-GRACE, we also compared it with the clustering algorithm ESSX proposed in [17]. ESSX hierarchically merges clusters of documents using the tree-edit distance. At each step of the algorithm, the pair of clusters with the smallest average distance between the documents in them is merged. The edit distance between two trees is defined by the minimum cost required to transform one tree to the other. This cost is computed by summing up the cost of the primitive operations (i.e., node insertion, node deletion, node renaming, subtree insertion, and subtree deletion) involved in the transformation. However, since the cost of computing tree distance on XML documents is very high, we could only run ESSX on a random sample of 40 documents from the DBLP database.[7]

In the 40 documents, there is a natural partitioning: 10 documents belong to $proceedings$, 10 to $phdthesis$, 10 to $journals$, six to $books$, and four to $incollections$. By setting the number of clusters k to five, S-GRACE-2 discovered five clusters that exactly match the original partitioning. Note that the same k value was given to ESSX as well. When running ESSX, the cost of node relabeling, insertion, and deletion were all set to 1, whereas the cost of subtree insertion and deletion, ranged from 0 to 10. Interestingly, the clustering results were the same for all the values in this range. However, the clusters generated were very different from the original partitioning. One of the five clusters contains 30 documents from $proceedings$, $phdthesis$, $books$, and $incollections$. The remaining 10 $journal$ documents are distributed into four clusters A, B, C, and D with $|A| = |B| = |C| = 1$ and $|D| = 7$. We found that all the documents in D do not contain the tag $cite$, while documents in A, B, and C contain many instances of $cite$. In ESSX, according to [17], a subtree can be inserted into the source tree to transform it to the target tree only if it has already appeared in the source tree. Therefore, it is not possible to use subtree editing operation to convert a document in D to a document in A, B, or C. Only node insertion or deletion can be used to convert source tree to target tree in this case. This explains why the different cost parameters of the subtree editing operation has no effect in the clustering. Since node insertion has a positive cost associated with it, the difference on the number of $cite$ tags between two documents would affect the edit distance between them. Thus, journal documents form four clusters because some of them have many more $cites$ than the others.

The time to run ESSX to cluster the 40 documents is 530 seconds, while S-GRACE-2 runs in less than two seconds, including the I/O cost. This demonstrates that S-GRACE is not only more effective but also more efficient than ESSX in performing clustering.

6 Discussion

6.1 Schema Design for XML Documents

In this paper, we have not advocated any new schema design method for storing XML documents. Neither do we claim that S-GRACE can *always* discover some nicely structured clusters to improve a schema. The clustering quality depends heavily on whether the collection of documents has some inherently good structure like that of the DBLP database. However, given a large collection of documents, it would be beneficial to run an algorithm like S-GRACE to identify potential clusters. These clusters could be useful not only for database schema redefinition, as we demonstrated here, but

7. Forty documents are already larger than the data set used in [17], which contains only 20 documents. We did try an experiment with 1,000 documents, however, ESSX was impractical for this case. The average time to compute the tree distance between two documents is about 0.6 seconds; computing the distances between all pairs of the 1,000 documents would require about four days.

also for other applications like data analysis and DTD extraction from large collections of XML data.

Notice that the number of clusters k generated by S-GRACE can be controlled. If the method is intended to be used for partitioning the schema of an XML database, k should not be too large for practical reasons. Moreover, the tables in the partitioned schema could be further optimized for query purposes.

6.2 Other Clustering Algorithms

As have been pointed out, the framework in S-GRACE does not preclude the use of other clustering algorithms. To validate the applicability of this framework on other clustering algorithms, we implemented the density-based clustering algorithm DBSCAN [9] and tested it on the s-graphs.

We ran DBSCAN on the s-graphs extracted from the documents in DBLP with different settings of parameters and discovered clusters similar to those reported in Section 5.3. In particular, besides the clusters on $inproceedings$ and $articles$, DBSCAN also dug out three rather small clusters (containing about 500 documents each), which hid inside the "outlier" cluster in the experiment performed with S-GRACE. Due to the three smaller clusters, both the outliers ratio and the average similarity are reduced when DBSCAN is used. The result of this experiment shows that our methodology is generic and can be used with different clustering algorithms. Most importantly, the fact that nearly the same clusters are discovered shows that the s-graph is a robust "feature" for clustering semistructured data.

7 Conclusion

We have proposed a framework for clustering XML data. We have shown that clustering based on the notion of edit distance between the tree representations of XML data is too costly to be practical. Hence, an effective summarization, which can distinguish documents among different clusters would be highly desirable. Based on this direction, we developed the notion of s-graph to represent XML data and suggested a distance metric to perform clustering on XML data. We have shown that the s-graph of an XML document can be encoded by a cheap bit string and clustering can then be efficiently applied on the set of bit strings for the whole document collection. With the structural information encoded, clustering of XML data becomes efficient and scalable using the proposed S-GRACE algorithm. As an application of the proposed framework, we have shown that clustering a large collection of XML documents by structure can alleviate the fragmentation problem of storing them into relational tables.

Our experiments on synthetic data show that S-GRACE is effective and efficient, whereas the performance studies on the real DBLP data set show that S-GRACE can discover clusters that could not be easily spotted by manual inspection. Moreover, the query performance on DBLP data, after using the clustering results to partition the database schema, is significantly improved. Although, in our test cases the DTDs of the data sets cover tree-structured documents only, S-GRACE can be applied as well for document collections of arbitrary (graph) structure. Thus, the distance metric on s-graph representations is also more generic than other metrics based on tree edit distance.

References

[1] S. Abiteboul, S. Cluet, and T. Milo, "Querying and Updating the File," *Proc. 19th Int'l Conf. Very Large Data Bases,* pp. 73-84, 1993.

[2] A. Aboulnaga, J.F. Naughton, and C. Zhang, "Generating Synthetic Complex-Structured XML Document," *Proc. Fifth Int'l Workshop Web and Databases,* 2001.

[3] H. Bunke and K. Shearer, "A Graph Distance Metric Based on the Maximal Common Subgraph," *Pattern Recognition Letters,* vol. 19, no. 3, pp. 255-259, 1998.

[4] D. Coppersmith and S. Winograd, "Matrix Multiplication via Arithmetic Progressions," *Proc. 19th Ann. ACM Symp. Theory of Computing,* 1987.

[5] DBLP XML records, http://www.acm.org/sigmod/dblp/db/index.html, Feb. 2001.

[6] S. DeRose, E. Maler, and D. Orchard, "XML Linking Language (XLink), Version 1.0" W3C Recommendation, http://www.w3.org/TR/xlink/, June 2001.

[7] A. Deutsch, M. Fernandez, and D. Suciu, "Storing Semistructured Data with STORED," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 431-442, 1999.

[8] A.L. Diaz and D. Lovell XML Generator, http://www.alphaworks.ibm.com/tech/xmlgenerator, 1999.

[9] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining,* pp. 226-231, 1996.

[10] Excelon, http://www.odi.com/excelon, 2001.

[11] D Guillaume and F Murtagh, "Clustering of XML Documents," *Computer Physics Comm.,* vol. 127, pp. 215-227, 2000.

[12] S. Guha, R. Rastogi, and K. Shim, "ROCK: A Robust Clustering Algorithm For Categorical Attributes," *Proc. 15th Int'l Conf. Data Eng.,* pp. 512-521, 1999.

[13] International Press Telecommunications Council, News Industry Text Format(NITF), http://www.nift.org, 2000.

[14] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data," *Proc. 18th Int'l Conf. Data Eng.,* 2002.

[15] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore: A Database Management System for Semistructured Data," *SIGMOD Record,* vol. 26, no. 3, pp. 54-66, Sept. 1997.

[16] R.T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. 20th Int'l Conf. Very Large Data Bases,* pp. 144-155, Sept. 1994.

[17] A. Nierman and H.V. Jagadish, "Evaluating Structural Similarity in XML Documents," *Proc. Fifth Int'l Workshop Web and Databases,* June 2002.

[18] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[19] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proc. 25th Int'l Conf. Very Large Data Bases,* pp. 302-314, 1999.

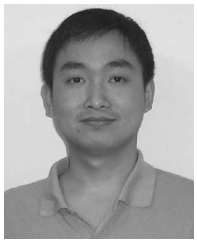
[20] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases," *Proc. 10th Int'l Conf. Database and Expert Systems Applications,* pp. 206-217, 1999.

[21] World Wide Web Consortium, "XML Path Language (XPath) Version 1.0,"http://www.w3.org/TR/xpath, Nov. 1999.

[22] World Wide Web Consortium, "XQuery: A Query Language for XML," W3C Working Draft, http://www.w3.org/TR/xquery, Feb. 2001.

[23] O. Zamir, O. Etzioni, O. Madani, and R.M. Karp, "Fast and Intuitive Clustering of Web Documents," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining,* pp. 287-290, 1997.

[24] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," *SIAM J. Computing,* vol. 18, no. 6, pp. 1245-1262, 1989.



Wang Lian received the BEng degree in computer science from Wuhan University, Wuhan, China in 1996 and the MPhil degree in computer science from The University of Hong Kong in 2000. He is currently a PhD candidate at the final stage in the Department of Computer Science and Information Systems at The University of Hong Kong. His research interests include semistructured data management and query processing, data mining, data warehousing, information dissemination, and Web semantic.

David Wai-lok Cheung received the BSc degree in mathematics from the Chinese University of Hong Kong and the MSc and PhD degrees in computer science from Simon Fraser University, Canada, in 1985 and 1989, respectively. From 1989 to 1993, he was a member of the scientific staff at Bell Northern Research, Canada. Since 1994, he has been a faculty member in the Department of Computer Science and Information Systems at The University of Hong Kong. He is also the director of the Center for E-Commerce Infrastructure Development. His research interests include data mining, data warehouse, XML technology for e-commerce, and bioinformatics. Dr. Cheung is the program committee chairman of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2001). He is the program chairman of the Hong Kong International Computer Conference 2003. Dr. Cheung is a member of the ACM and the IEEE Computer Society.

Nikos Mamoulis received the diploma in computer engineering and informatics in 1995 from the University of Patras, Greece, and the PhD degree in computer science in 2000 from the Hong Kong University of Science and Technology. Since September 2001, he has been an assistant professor in the Department of Computer Science, University of Hong Kong. In the past, he has worked as a research and development engineer at the Computer Technology Institute, Patras, Greece, and as a postdoctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), the Netherlands. His research interests include spatial, spatio-temporal, multimedia, object-oriented and semistructured databases, and constraint satisfaction problems.

Siu-Ming Yiu received the BSc degree in computer science from the Chinese University of Hong Kong, the MS degree in computer and information science from Temple University, and the PhD degree in computer science from the University of Hong Kong. He is currently a teaching consultant in the Department of Computer Science and Information Systems at the University of Hong Kong. His research interests include data mining and computational biology.