ISSN 2277-3061

Retrieving information chunks from a repository of documents' SIT Collected from heterogeneous sources

Raad Alwan, Baydaa Al-Hamadani

Associat Prof., Dept. of Computer Science, Philadelphia University, Jordan

Ralwan@philadelphia.edu.jo

Asistant prof., . of Computer Science, Zarqa University, Jordan

bhamadani@zu.edu.jo

ABSTRACT

XML documents are generated from heterogeneous resources. They may share the same data but in different Schema, which make it difficult to retrieve information from them. In this paper we propose a new technique that first; minimizes the size of the XML documents by reducing the redundancy of the structure part and generate the repository for these documents, and second; relaxes and decomposes the XPath query in two stages to determine the relevant documents and the relevant part within these documents. The results show significant precision and recall comparing with the exact XPath queries.

Indexing terms/Keywords

Heterogeneous resources, data integrity, query relaxation, XML retrieval

Academic Discipline And Sub-Disciplines

Computer Science.

SUBJECT CLASSIFICATION

Information Retrieval, web technology.

TYPE (METHOD/APPROACH)

Query decomposition, generating SIT.

Council for Innovative Research

Peer Review Research Publishing System

Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Vol. 14 , No. 3

www.ijctonline.com , editorijctonline@gmail.com

ISSN 2277-3061

1. INTRODUCTION

Nowadays thousands of computer applications and software are used and most of them are generating different kinds of heterogeneous data. The integration of these data is becoming a significant issue to facilitate the interoperability and accessibility of these data.

There are several techniques to integrate data such as data warehousing [1-3], semantic integration [4, 5], ontology-based integration [6, 7], and other ways of technologies. The building block of most of the previous approaches is XML representation of the data. This paper focuses on the data that are represented in XML format for several reasons. The main reason is that XML is becoming the standard way of representing and transferring data through the internet. XML documents have several features that make them easy to be used such as: [8-10] readability by both human and machine, interoperability between different types of hardware and software, extensibility, generality, internationality, etc.

One of the main aims of data integration is the ability to retrieve information from these data. There are different types of query languages that are used to retrieve information from XML documents; one of them is XPath query language. Using XPath is easy, straight forward way, nevertheless, it has the ability to retrieve information from only a specific XML document. This paper focuses on two main aspects, (1) generate a repository of XML Structure Indexed Tree (SIT)s, and (2)expand the XPath query language to give it the ability to retrieve information from the generated repository.

Several techniques have been proposed to XML-IR. These techniques can be grouped into two parts, either Content-Only (CO) or Content-And-Structure (CAS) techniques [11, 12]. CO retrieval follows the same algorithms that retrieve information from traditional text files ignoring the structure part of the XML documents. On contrast, CAS retrieval takes into account the structure part of the document to retrieve more precise results.

2. GENERATING THE REPOSITORY

For the purposes of speeding up the retrieving process, our approach reforms the XML documents into SIT representing the structure part of the document. The data under each root-to-leaf path are stored in containers which are indexed by that path. Each tree is accompanied with a path-dictionary which has all the tags names and attributes in the document. This process will achieve several goals:

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress, and her own childhood to become queen
    of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology
    society in England, the young survivors lay the
    foundation for a new society.</description>
  </book>
</catalog >
```

Figure 1: An XML example (Book catalog)

ISSN 2277-3061

(1) reduce the size required to store each document by abridging the redundancy in the structure part, (2) combine all the data under the same path in the same container to fasten the CO retrieval process, (3) the path-dictionary used as a key to specify the relevant documents, and (4) the SIT is used to achieve CAS retrieval. For each XML document, the SIT; the path-dictionary; and the containers are added as one unit to the repository.

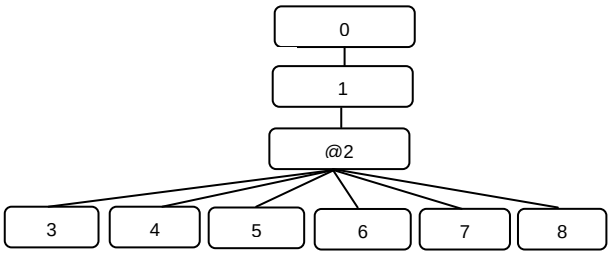
Example-1: *A repository unit*

This section exemplifies the process of generating one unit of the repository. As a very short example, Figure 1 is a small chunk taken from a catalog of books represented as XML document.

There are two ways to mange an XML document, DOM and SAX. Document Object Model (DOM) converts the XML document into tree-like structure and the document should be entirely stored in the memory to be processed. On the other hand, simple API for XML document (SAX) scans the document only once, throwing several events such as start-document, start-element, end-element, and end-document.

Since the repository is dealing with lots of documents, it will be memory consuming to store all the required documents in the memory and process them through DOM. SAX is such a better way. Through only one scan for each XML document, the structure is separated from the data; the structured tree is generated; the data in the leaves of the document are stored in their appropriate containers; and the path-dictionary produced. Figure 2a shows the SIT for the XML sample in Figure 1. Each node in the SIT is indexed with a unique number that represents the sequence of that node in the original XML document. The number of bits needed to store the SIT is $(\boldsymbol{N} * \mathbf{log}_2 \boldsymbol{N})$, where N is the number of nodes in the SIT.

To keep the consistency of data and the relationship between each datum and the other, each data chunk is accompanied with a unique ID to represent its sequence in the original document. These IDs are useful in the retrieval purposes and in the process of retaining the original XML document from the repository. The process of retaining the XML document from the repository form to its original form is beyond the scope of this paper. Figure 2b shows the containers for the XML document.



(a)

0	Catalog
1	Book
2	Id
3	Author
4	title
5	Genre
6	Price
7	Published-date
8	Description

(c)

/0/1/@2	"bk101" "bk102" "bk103"
/0/1/3	Gambardella, Matthew Ralls, Kim Corets, Eva
/0/1/4	XML Developer's Guide Midnight Rain Maeve Ascendant
/0/1/5	Computer Fantasy Fantasy
/0/1/6	44.95 5.95 5.95
/0/1/7	2000-10-01 2000-12-16 2000-11-17
/0/1/8	An in-depth look at creating applications with XML. A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world. After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.

(b)

Figure 2: SIT for the (Book) sample XML document.

Each container has an index which represents the complete path from the root to the leaves having the data inside that container. The path is build depending on the path-dictionary (Figure 2c).

The algorithm of transforming the XML documents to be added to the repository is shown in Figure 3. The algorithm has the main methods while using the SAX API. These methods are (1) *startElement:* during which the element names are added to the pathDictionary and a path is collected. (2) *endElement* : whenever an end element occurs, a complete path is constructed and added to SIT if it is not already did. (3) *endDocument:* at this stage, the SIT; pathDictionary; and the containers filled with the appropriate data are all added to the repository as one unit.

Algorithm Reform an XML document (**D**)
pathDictionary=[i_0, i_1, ..., i_n]
SIT=[j_0, j_1, ..., j_m]
pathStack=[k_p, k_{p-1}, ..., k_0]
R is the target repository
Using SAX to throw and implement the following functions
Function startElement(String *eName*, String data)
IDOrder= the current order
if (*eName* ∉ *pathDictionary*)
pathDictionary=[i_0, i_1, ..., i_n] ∪ $eName_{n+1}$
Q"= *n+1*
else
Q"= q where i_q =*eName*
pathStack=[k_p, k_{p-1}, ..., k_0] ∪ [$Q"_{p+1}$]
currentPath ← $k_0 + k_1 + ... + k_p$
if (currentPath ∉ *structured-Tree*)
Add *currentPath* to *structured*-Tree
IDOrder++
if (data is not empty)
Add (*IDOrder*,*data*) to the leaf node of SIT
END.
Function *endElement*(String eName, String data)
If *data* ≠ Ø
Add (*IDOrder*,*data*) to the leaf node of pathStack
End.
Function *endDocument* ()
Add ***pathDictionary*** *to* ***R***
For all the ***N*** branches in ***structured-Tree***
index= Collect all the nodes [j_0, j_1, ..., j_k]
data= the contents of the leaf node for the path [j_0, j_1, ..., j_k]
Add a Container(**index**, **data**) to **R**
End.

Figure 3: Reforming an XML document algorithm

3. XML RETRIEVAL

XML documents are semi-structured and the retrieval techniques from such documents are more precise than retrieving information from structured documents [1]. Semi-structured document retrieval (SDR) techniques have the ability to retrieve information from the structure part and the content part of the XML documents. They focus on retrieving the very specific required part from the document rather that retrieving the entire document. As an example, SDR retrieve the required sections in a book rather than just retrieving the whole book. Moreover, SDR techniques provide their users the ability of determining the part of the document that has the required data. For instance, a user query could be “a graph about the compression time in compressing XML chapter”. In this case, the “graph” and “chapter” are retrieving the structure part, while “compression” and “XML compressing” are retrieving the content part of the XML documents.

4. HETEROGENEOUS XML RESOURCES & XPATH

Several query languages have been proposed to retrieve information from XML documents. Each of which use a special mechanism to reach their goal, but all of them are sharing t the feature in which the user should determine the XML document(s) that s/he would like to retrieve information from [2-6].

The standard XML query languages are XPath and XQuery and both of them are W3C recommendation [7-9]. XPath is used to retrieve parts of the XML document either directly or via another query language since it is the core to all other XML query languages [10]. XPath has the ability to CO and CAS retrieval. It depends on exact path matching to match the user query with the XML tree. If any missing node or misspelled tag name, there will be no matching and no retrieval information from the related document. For that reason several techniques have been proposed to relax and rewrite the XPath query to increase the efficiency and the flexibility of these queries [10-12].

Our work aims to retrieve information from a repository of XML document from heterogeneous sources. These documents have different structure and scheme even if they have the same data values.

Example-2: *heterogeneous resources*

Figure 4 exemplifies three SIT from the repository for three different XML documents. All these documents, and more, having the same data values but arranges in different structure depending on the source of these data. For clarity of the figure, the indexes of the nodes are replaced by the real tag names and a sample of the actual data is shown in the bottom of each SIT.

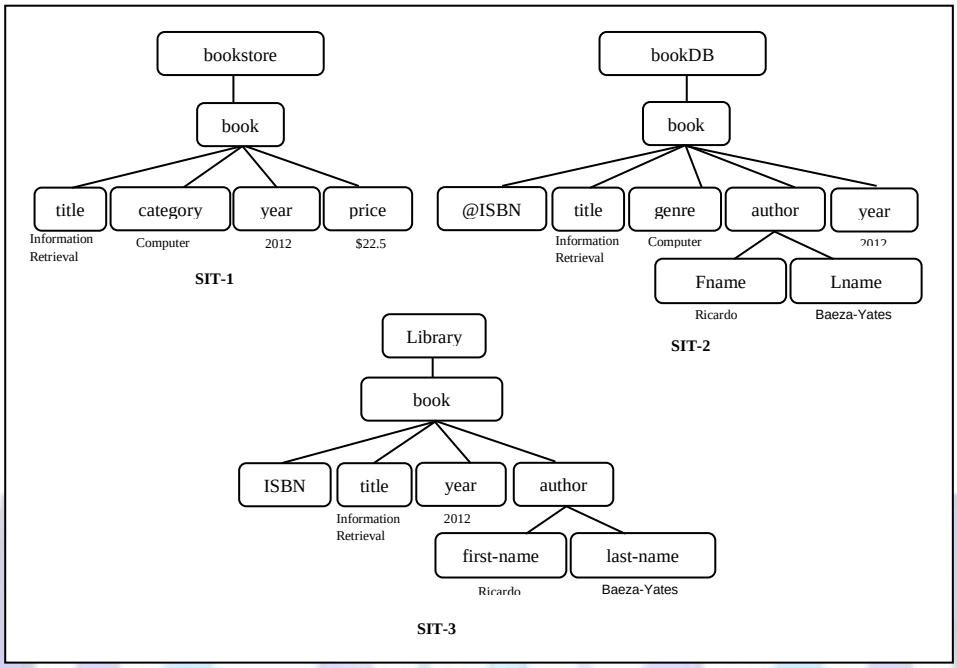


Figure 4: Three different SIT for the same data contents

Lots of XPath queries will not have any results when trying to retrieve information from the previous XML documents. Examples of these queries alongside with the reasons upon no results retrieved are:

1. //book[title = ”Information Retrieval”][price < $40]
//bookstore[genre = ”Computer”]
These two queries are not matching the structure of any document. The names of the nodes are different and the sequences of the tags are not matching.
2. //bookDB[ISBN = 12345]/author[firstname = ”Ricardo”]
In this query the (ISBN) node is an attribute in the original document, the thing that is not mentioned in the query.
3. //bookDB[@ISBN = 12345]/author[Fname = ”Ritchard”]
The whole query is matching the path of SIT-2, but misspelled the content of the document results on no content matching.
4. //boostore/title/category
The (title) and (category) nodes are siblings in the XML tree, while the query structure shows that they are parent and child respectively.
5. //library[YEAR > 2013]
The case letters of one node is not matching (XPath is a case sensitive language)

Relaxing XPath query is one remedy to solve this problem. The relaxation process depends on decomposing the XPath query into several sub queries. This process passes through two stages, (1) determine the relevant documents and (2) determine the relevant containers within these documents [13].

4.1 Query Decomposition: Stage-1

In this stage, the XPath query is decomposed into several sub-queries according to the relevancy between the elements in the query and the elements in the XML documents from the repository.

Definition- 1 *(relevant document):* If $\boldsymbol{Q} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \dots \boldsymbol{e_n}\}$ is the set of elements in the user's query, and $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \dots \boldsymbol{x_m}\}$ is the set of all the transformed XML documents in the repository. $\boldsymbol{x'} \in \boldsymbol{X}$ is considered to be relevant document if $\exists \boldsymbol{e_i} | \ \boldsymbol{e_i} \in \boldsymbol{x_j'}.\boldsymbol{PD}$, where $\boldsymbol{PD}$ is the *path-dictionary* for the specified document. If so, add $\boldsymbol{x_j'}$ to the set of relevant documents **D'** and add $\boldsymbol{e_i}$ to $\boldsymbol{q'_j}$.

■

According to definition-1, a XML document is considered to be relevant if it has one or more of the query elements in its path-dictionary. All the relevant documents $X' = \{x_1', x_2', \ldots x_l'\}$ are collected in a small repository for relevant XML documents each of which is accompanied with its corresponding sub-query $Q' = \{q_1', q_2', \ldots q_l'\}$. Note that the number of sub-queries is equal to the number of the relevant documents. All the elements and attribute names in Q' are replaced with its location in the path-dictionary of the relevant document. This process is done to prepare the sub-queries for the second decomposition stage.

4.2 Query Decomposition: Stage-2

After specifying the relevant documents, the role of this stage is to specify the relevant containers within these documents. This process causes further decomposition to the sub-queries

Definition-2 (Relevant Container): Given $C = \{c_1, c_2, \cdots c_n\}$ represents the set of n containers for a relative document and each of these containers has an index with k elements $P = \{p_1, p_2, \cdots p_k\}$, and $q_i' = \{e_1, e_2, \cdots e_m\}$ represents the set of m elements in the sub-query accompanies **C**, to select the relevant containers, follow the steps:

$\forall q_i' \in Q$: **for** $i = k$ **to** 1

$\forall c_i \in C$ **if** $p_k \in q_i'$

Add c_i to **C'**| **C'** is the list of the relevant containers

$\forall e_k$ if $e_k \in P$, copy e_k and add it to the list of the elements in $q_{i,j}'$ to denote a new sub-query.

■

At the end of this stage only the relevant containers taken from the relevant documents are uploaded into the memory for ranking process. Each of these containers is accompanied with its sub-query.

Example-3: *Query decomposition*

Suppose the following XPath query (**Q**) in which the user requires to retrieve the ISBN and the published year of books titled "Information Retrieval" for the author's first name is "Richardo" and the price does not exceed $40:

//book/ISBN [title = "Information Retrieval"]/published-year [firstName = "Richardo"][price<$40]

In stage-1 the query is decomposed to three sub-queries in case of the three relevant SITs in Figure 4. Each sub-query will accompany its relevant document for the purpose of stage-2 decomposition. The three subqueries are as follows:

SIT_1 ~ **Q_1** : //book[title = ***"Information Retrieval"]/published-year [price < 40]***

SIT_2 ~ **Q_2** : ***//book/ISBN[title = "Information Retrieval"]/published-year [firstName = "Richardo"]***

SIT_3 ~ **Q_3** : ***//book/ISBN[title = "Information Retrieval"]/published-year [firstName = "Richardo"]***

For the decomposition stage-2, each sub-query is decomposed into several sub-queries. The number of the new sub-queries depends on the relative containers from the repository. As an example of mapping the query **Q_1** to SIT-1, Figure 5 shows the tree representation for the sub-query **Q_2** (to the left) and its mapping the SIT-2 (to the right).

The mapping process M :Q $\xrightarrow{M}$ SIT depends on the exact matching and approximate matching between the query elements and the SIT nodes. The approximate matching is derived from the string-similarity algorithm [14] in order to match the misspelling in the elements and attribute names.

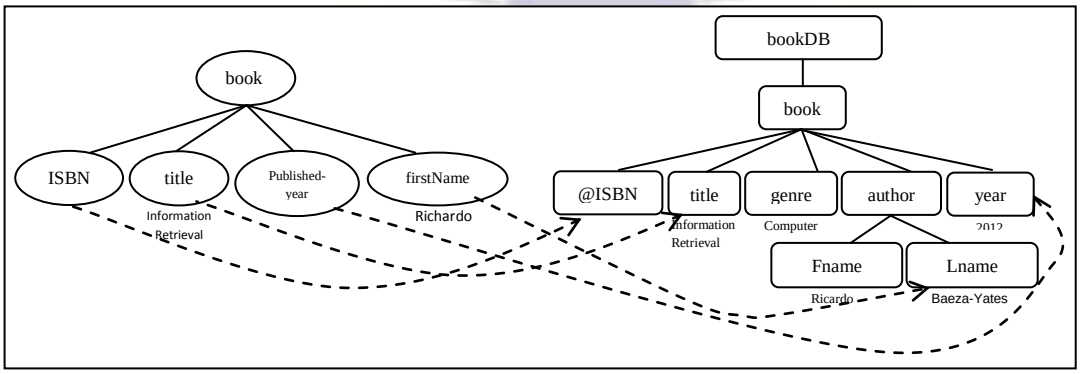


Figure 5: Example of the mapping technique.

Definition-3: *String-similarity:*

let st1 = [s1s2s3 ... sn] and st2 = [c1c2c3 ... cm]

st1Set = [s1s2][s2s3][s3s4]...[sn-1sn]

st2Set = [c1c2][c2c3][c3c4]...[cm-1cm]

$$\boldsymbol{SimilarityRatio} = \frac{\mathbf{2} \times |\boldsymbol{st1Set} \cap \boldsymbol{st2Set}|}{|\boldsymbol{st1Set}| + |\boldsymbol{st2Set}|}$$

The choice of string-similarity algorithm is made on the ground that it meets most of XCVQ-QP needs since this algorithm has the following features:

1. If two strings have minor differences, they are considered to be similar (ex: heap, heard).
2. If two strings have the same words but in different order, they are considered to be similar (ex: data base management system, managing data base).

The resulting sub queries from stage-2 for $\mathbf{Q_1}$ are as follows:

$\mathbf{Q_{2,1}}$ **:** book/ISBN

$\mathbf{Q_{2,2}}$ **:** book[title = ”*Information Retrieval*”]

$\mathbf{Q_{2,3}}$ **:** book/year

$\mathbf{Q_{2,4}}$ **:** book[Fname = ”Richardo”]

4.3 Query relaxation

In this stage, the list of sub-queries $\mathbf{Q_{i,j}}$ is going to be relaxed to determine the relevancy of each of these queries to the document. The relaxation types and their costs are listed below:

1. Node insertion: This type of relaxation is done by inserting one or more nodes in the list of available query nodes. To reach this goal, a comparison between the sub-query $\mathbf{Q_{i,j}}$ and the index of the relevant containers is made.

Definition-4 (Node-Insertion): *Given a container $c_j \in C'$ has an index with $\boldsymbol{k}$ elements $\boldsymbol{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \cdots \boldsymbol{p_k}\}$ and given $\boldsymbol{q_{i,j}} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \cdots \boldsymbol{e_m}\}$ represents the set of m elements in the sub-query accompanies the $\boldsymbol{i^{th}}$ relevant document and $\boldsymbol{j^{th}}$ relevant container. The relaxed sub-query $\boldsymbol{q'_{i,j}} = \boldsymbol{q_{i,j}} \cup (\boldsymbol{P} \ominus \boldsymbol{q_{i,j}})$.* ***the cost of inserting node(s) in a sub-query is***

$$\boldsymbol{cost_{In}}(\boldsymbol{q}, \boldsymbol{q'}) = \frac{|(\boldsymbol{P} \ominus \boldsymbol{q'_{i,j}})|}{|\boldsymbol{P}|}$$

■

2. Node renaming: After completing the first stage of relaxation, each sub-query is going to pass through the following procedure:

Let $\boldsymbol{q'_{i,j}} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \cdots \boldsymbol{e_m}\}$ be the set of elements in the current sub-query, $\boldsymbol{P} = \{p_1, p_2, \ldots, p_k\}$ be the set of elements of the index for the current container associated with $\boldsymbol{q'_{i,j}}$.

$\forall\, \boldsymbol{e_i} \in \boldsymbol{q'_{i,j}}$,

if $\boldsymbol{e_i} \notin \boldsymbol{p_j} | \ \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{k}$ then find the *string-similarity*$(\boldsymbol{e_i}, \boldsymbol{p_j})$
if *(SimilarityRatio > 50%)* then

$changes = \sum_{i=1, j=1}^{m,k} SUB\ \boldsymbol{e_i}\ with\ \boldsymbol{p_j}$

And the cost of the node renaming process is:

$$\boldsymbol{cost_{Ren}}(\boldsymbol{q}, \boldsymbol{q'}) = \frac{\boldsymbol{changes}}{|\boldsymbol{q'_{i,j}}|}$$

3. Node deletion: After inserting all the required nodes from the index of a container, the extra nodes from the query should be removed.

Definition-5 (Node-deletion): *Given a container $\boldsymbol{c_j} \in \boldsymbol{C'}$ has an index with $\boldsymbol{k}$ elements $\boldsymbol{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \cdots \boldsymbol{p_k}\}$ and given $\boldsymbol{q_{i,j}} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \cdots \boldsymbol{e_m}\}$ represents the set of m elements in the sub-query accompanies the $\boldsymbol{i^{th}}$ relevant document and $\boldsymbol{j^{th}}$ relevant container. The relaxed sub-query $\boldsymbol{q'_{i,j}} = \boldsymbol{q'_{i,j}} - [\boldsymbol{q''_{i,j}} \ominus \boldsymbol{P}]$. The cost required to delete node(s) from a sub-query is:*

ISSN 2277-3061

$$cost_{Del}(q, q') = \frac{|q'_{i,j} \ominus P|}{|q'_{i,j}|}$$

■

4. Order relaxation: This is the last relaxation process which arranges the order of the nodes in each resulted sub-queries. The cost of this relaxation is shown in equation (11) such that changes represent the number of changing in the order of the elements in the sub-query.

$$cost_{Order}(q, q') = \frac{\sum n_i | i = changes\ in\ order}{|Q'|}$$

After relaxing all the sub-queries, the process of finding the similarity between the containers' index and the sub-queries is:

$$sim(x, Q) = 1 - (cost_k(q, q') \ \forall q \in \{Q'\}) / (k)$$

■

5. EXPERIMENTAL EVALUATION

To test the performance of the proposed system, a set of different types of XML documents has been chosen. These documents have different sizes, number of elements, number of nodes, the depth of the longest path, and the data ratio (DR) which is calculated as follows (Sakr, 2009):

$$DR_d = (D_d / Si_d) / 100$$

Where DR_d is the data ratio for the XML document (d), (D) is the data, and (Si) represents the size of the XML document. The XML set has DR range between textual documents with DR exceeds 70% to structural document with DR less than 30% passing through regular documents. The overall size of the data was around 850MB grouped into categories according to their topic to ease the process of testing the precision and the recall of the retrieved documents.

All the testing were carried out on a personal computer with Intel(R) Core(TM)2 Due CPU processor that has the speed of 5.50 GHz. The RAM memory of the tested environment is 4.00GB and 300GB of hard disk drive. It has 32-bit Windows Vista operating system.

During this stage both Query Functional Test (QFT) and Query Performance Test (QPT) [13] were investigated. For this purpose, a query benchmark was tested from XPathMark [15]since it covers all the required categories of XML queries. The benchmark is extended to be 100s of queries scattered among axes, node test, operators, and functions concepts. To explain the testing results, we choose two queries as a sample from each of the four given concepts. The precision and the recall of the retrieved documents according to the selected queries are shown in Figure 6. The results show a good precision and recall for all the tested queries, but more experiment needed to compare the relaxed query with the XPath original query in terms of precision and recall to magnify the significance of our approach.

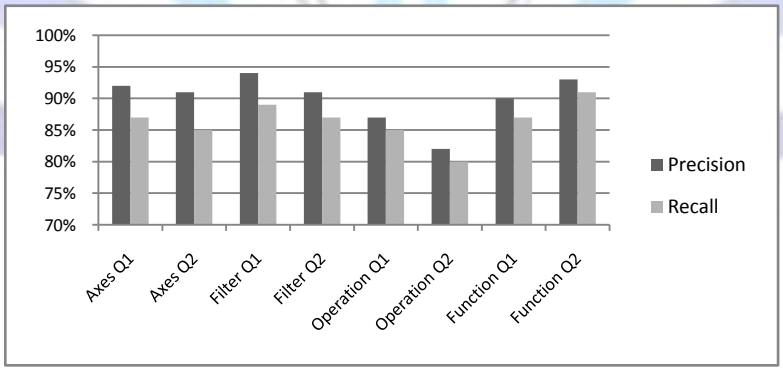
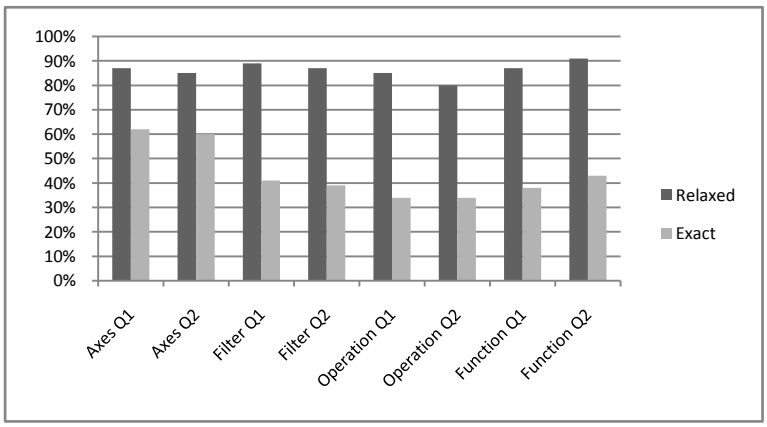


Figure 6: Precision and recall test for selected queries.

Figure 7 illustrates the recall of the retrieved documents using the relaxed queries and the exact XPath queries. It is clear that the relaxation of the query make the recall much higher with average 43%. This test shows clearly that relaxing the queries increase retrieving the relevant documents significantly.



Although the recall of the retrieved documents is significant, but the time needed to process a relaxed query (Figure 8) is more than the time needed to process the exact XPath query. This difference in time is due to the time required to relax the query and to the matching techniques.

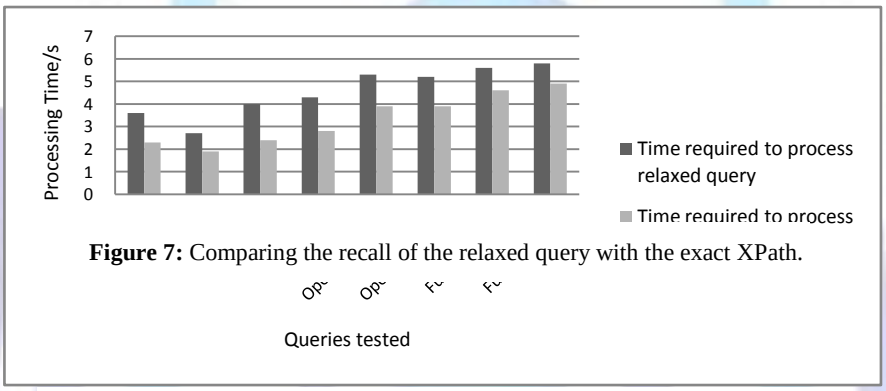


Figure 7: Comparing the recall of the relaxed query with the exact XPath.

Figure 8: Sample test of precision and recall for the retrieved **documents**

When the complexity of the query increases, the time required to process it increases as well. Axes queries are much simpler than function queries and they need less time than the others.

6. RELATED WORK

Several approaches have been proposed to retrieve information from heterogeneous XML documents [16-21]. Most these approaches depended on some semantic web mapping evaluation with the condition of Schema presence, either by query translation or by using global-as-view and local-as-view integration technique. Other approaches [22-26] does not necessarily require the global schema to achieve their goals, instead they introduced mapping algorithms to translate the user's query.

Latest approaches [27-30] proposed ontology based approached to find some semantic mapping between the user query and the heterogeneous documents' relational schema.

Our previous work [31] focused on compressing the XML documents and retrieve information from the compressed version through vague user queries. To the best of our knowledge, this is the first work that relaxes the user query and retrieves information from a repository of SIT instead of keeping the original XML documents with their high redundancy in their structure.

7. CONCLUSION & FUTURE WORK

In this paper we proposed a new approach for retrieving chunks of information from XML documents by transforming the XML documents into SIT and relax the XPath queries to increase the precision and recall for the retrieved documents. The transformation process minimizes the redundancy in the structure of the XML documents, which reduces the size of these documents. The process of relaxing the queries, which is done after two stages of decomposition, were tested and the results show that more recall and precision were gained. The retrieved information according to a specific query is structured as XML documents and can be return back to the repository for further retrieval. More development will be made to our approach including add more functions to the list of XPath functions. (Synonyms) is one of the suggested functions which uses WordNet to find the synonyms of the nodes or the content to maximise the retrieved information and simplify the user query.

ACKNOWLEDGMENTS

Our thanks to the experts who have contributed towards development of the template.

REFERENCES

[1] Manning, C.D., P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. 1st ed: Cambridge University Press. 2009,

[2] Norbert, F. and G. Kai," *XIRQL: An XML query language based on information retrieval concepts".* ACM T INFORM SYST, vol. **22** no 2: pp. 313-356.2004

[3] Paparizos, S., S. Al-Khalifa, A. Chapman, H.V. Jagadish, L.V.S. Lakshmanan, A. Nierman, J.M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. "*TIMBER:A Native System for Querying XML"*. in Proceeding of the *ACM SIGMOD International Conference on Management of Data*. pp 274-291. 2003

[4] W3C. *XML Linking Language (XLink) Version 1.0*. 2001 [cited 2014 Available from: http://www.w3.org/TR/xlink/.

[5] W3C. *XML Pointer Language (XPointer)*. 2002 [cited 2014 Available from: http://www.w3.org/TR/xptr/.

[6] W3C. *XML Path Language (XPath) 2.0*. 2010 [cited 2014 19/10]; Available from: http://www.w3.org/TR/xpath20/.

[7] Kay, M., *XPath 2.0 Programmers Reference*. 1st ed. Canada: Wiley Publishing, Inc. 2004,

[8] Andrew Watt, *XPath Essentials*. 1st ed: John Wiley & Sons Publishing. 2002,

[9] msdn. *XPath Syntax*. 2010 [cited 2014 Available from: http://msdn.microsoft.com/en-us/library/ms256471(v=vs.110).aspx.

[10] Cautis, B., A. Deutschy, and N. Onose. "*XPath Rewriting Using Multiple Views: AchievingCompleteness and Efficiency"*. in Proceeding of the *11th International Workshop on Web and Databases (WebDB 2008)*. pp 15-19. 2008

[11] Cautis, B., A. Deutsch, N. Onose, and V. Vassalos. "*Efficient Rewriting of XPath Queries Using Query Set Specifications"*. in Proceeding of the *VLDB '09*. pp 301-312. 2009

[12] Fazzinga, B., S. Flesca, and F. Furfaro," *XPath Query Relaxation through Rewriting Rules".* IEEE T KNOWL DATA EN, vol. **23** no 10: pp. 1583-1600.2011

[13] Al-Hamadani, B.T., "Retrieving Information from Compressed XML Documents According to Vague Queries", PhD thesis, University of Huddersfield, UK, 2011

[14] White, S. *How to Strike a Match*. 2008 [cited 2010 15/10]; Available from: http://www.catalysoft.com/articles/StrikeAMatch.html.

[15] Franceschet, M.," *XPathMark: Functional and Performance Tests for XPath".* Lecture Notes in Computer Science, Springer, vol. **3671** no 1: pp. 129-143.2005

[16] Bikakis, N., N. Gioldasis, C. Tsinaraki, and S. Christodoulakis, *Semantic Based Access over XML Data*, in *Visioning and Engineering the Knowledge Society. A Web Science Perspective*. Springer Berlin Heidelberg. pp. 259-267.2009

[17] Camillo, S., C. Heuser, and R. Mello, *Querying Heterogeneous XML Sources through a Conceptual Schema*, in *Conceptual Modeling - ER 2003*. Springer Berlin Heidelberg. pp. 186-199.2003

[18] Chaitan, B., G. Amarnath, L. Bertram, scher, M. Richard, P. Yannis, V. Pavel, and C. Vincent," *XML-based information mediation with MIX".* SIGMOD Rec., vol. **28** no 2: pp. 597-599.1999

[19] Cong, Y. and P. Lucian. "*Constraint-based XML query rewriting for data integration"*. in Proceeding of the *2004 ACM SIGMOD international conference on Management of data*. pp 371-382. 2004

[20] Damjanovic, V., T. Kurz, and R. Westenthaler. "*Semantic Enhancement: The Key to Massive and Heterogeneous Data Pools"*. in Proceeding of the *20th International Electrotechnical and Computer Science Conference (ERK 2011)*. pp 413-416. 2011

[21] RodrÃŒguez-Gianolli, P. and J. Mylopoulos, *A Semantic Approach to XML-based Data Integration*, in *Conceptual Modeling* H. S.Kunii, S. Jajodia, and A. Sølvberg, Editors., Springer Berlin Heidelberg. pp. 117-132.2001

[22] Cindy, X.C., A.M. George, P. Sriram, and M.R. Isabelle. "*Query translation scheme for heterogeneous XML data sources"*. in Proceeding of the *7th annual ACM international workshop on Web information and data management*. pp 31 - 38 2005

[23] Jayant, M., A.B. Philip, and R. Erhard. "*Generic Schema Matching with Cupid"*. in Proceeding of the *27th International Conference on Very Large Data Bases*. pp 49-58. 2001

[24] Sanz, I., "Flexible Technique for Heterogeneous XML Data Retrieval", PhD Thesis, The Jaume I University, Spain, 2007

[25] Zhou, X., S. Su, M. Papazoglou, M. Orlowska, K. Jeffery, F. Mandreoli, R. Martoglia, and P. Tiberio, *Approximate Query Answering for a Heterogeneous XML Document Base*, in *Web Information Systems (WISE 2004)*, X. Zhou, et al., Editors., Springer Berlin Heidelberg. pp. 337-351.2004

[26] FAZZINGA, B., S. FLESCA, and A. PUGLIESE," *Retrieving XML Data from Heterogeneous Sources through Vague Querying".* ACM T INFORM SYST, vol. **9** no 2: pp. 1-35.2009

[27] Schober, D., M. Boeker, J. Bullenkamp, C. Huszka, K. Depraetere, D. Teodoro, N. Nadah, R. Choquet, C. Daniel, and S. Schulz," *The DebugIT core ontology: semantic integration of antibiotics resistance patterns".* Stud Health Technol Inform, vol. **160** no 2: pp. 1060-1064.2009

[28] Wang, J., J. Lu, Y. Zhang, Z. Miao, and B. Zhou," *Integrating Heterogeneous Data Source Using Ontology".* J SOFTW, vol. **4** no 8: pp. 843-850.2009

[29] Yusuf, J.C.M., M.M. Su'ud, P. Boursier, and M. Alam. "*Extensive overview of an ontology-based architecture for accessing multi-format information for disaster management"*. in Proceeding of the *Information Retrieval & Knowledge Management (CAMP)*. pp 294-299. 2012

[30] H.Wache, T. V¨ogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. H¨ubner. "*Ontology-Based Integration of Information — A Survey of Existing Approaches"*. in Proceeding of the *In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*. pp 108-117. 2001

[31] Al-Hamadani, B., *Retrieving Information from Compressed XML Documents According to Vague queries.*, in *Design, Performance, and Analysis of Innovative Information Retrieval*, P.J. Lu, Editor., IGI Global. pp. 91-178.2012

Author' biography with Photo

Raad Alwan is an associated prof. in Philadelphia University in Jordan. He completed his undergraduate from the University of Baghdad, Iraq. He finished his MSc. and PhD in University College Dublin (1980) and Trinity College Dublin (1990), respectively. His research interests are algorithm analysis and design, information retrieval, and data mining.

Baydaa Al-hamadani is an assistance prof. in Zarqa University in Jordan. She got her Bachelor degree from the University of Technology in Iraq in 1994, followed by MSc. in the same University in 2000. In 2011, she got her PhD from the University of Huddersfield in the UK from the School of Informatics. Her fields of interest are XML, compressing and retrieving information, and ontology.