A New Indexing Technique XR+ Tree for Bio Informatics XML Data Compression

Dinh Duc Luong, Vuong Quang Phuong, Hoang Do Thanh Tung

Abstract*: *Informatics data is becoming huge due to the regular contribution of informatics community. Because bioinformatics problems are so diverse, documents for storing information need a structure that is easy to change, flexible, diverse and especially easy to share / contribute. For that reason, XML documents are the best solution for describing and storing this massive bioinformatics data. However, because XML documents have textual and semi-structured data, there is a lack of data reduction and data retrieval methods that are really relevant and effective.

In this paper, we apply a method of converting the number of name tags to numerical spatial coordinates to reduce the size of the document and propose an improved indexing method to increase query efficiency of XML documents. The idea of the paper is based on the characteristics of the XML data tree structure and the structure of the XR-tree index tree (R-tree improvement), we advance the algorithms to optimize for queries for XPath. Experiments have shown that the new algorithm has significantly increased performance compared to the previous method. And the results also point to some practical issues due to the variety of bioinformatics XML documents

Index Terms*: *bioinformatics, bioinformatics data, indexing method.

I. INTRODUCTION

Currently, data storage, transmission and query issues are becoming increasingly important as data collection is getting easier and faster from various sources. Data quickly becomes huge, complex, diverse. Typical of this problem is bioinformatics data. Bioinformatic data is diverse, describing all chemical reactions, protein structure to form Gene in living organisms, descriptions of DNA sequences / entire sequences for plants or animals. or describe the evolutionary tree of the world's living creatures. Thus, we need a data model that can store a variety of bioinformatics data structures.

The current solution for bioinformatics data is to store bioinformatic data in the form of an XML semi-structured document (eXtensible Markup Language). In recent years, XML is the most important platform for storing and transmitting data on the web. XML documents have an open structure, so they are highly flexible, easy to interact with and understand by many platforms, and are especially non-dependent, making XML very consistent with the requirements of bioinformatics data.

Revised Manuscript Received on June 12, 2019

Dinh Duc Luong, working at the Food Industrial College. Research Interest: bioinformatics, big data

Vuong Quang Phuong, working at the Institute of Information Technology (IOIT) - Vietnam Academy of Science and Technology (VAST).

Hoang Do Thanh Tung, working at the Institute of Information Technology (IOIT)-Vietnam Academy of Science and Technology (VAST)

Usually XML documents are not too large to transmit data over the network. But in the era of biotechnology, bioinformatics XML documents are very large, possibly up to Tetabyte due to the specificity of the bioinformatics industry with an increasing number of XML documents from the biological studies. As we know, XML documents are essentially textual data that is structured by name tags. The data section to describe accounts for 40-60% of the size of the XML data file [15]. With such large-sized XML text documents, there is still a lack of research on effective information retrieval of speed and size. Therefore, this paper wants to continue to study and propose methods for retrieving information in XML documents, efficiently querying and still compressing smaller XML data when querying, thanks to that is more efficient in storing and reducing access to the hard disk.

In the previous paper [11], we introduced algorithms to improve R-tree indexing methods to improve the efficiency of XPath queries, especially for sibling axes thanks to pointers. Experimental results have been impressive because the number of hard disk access times decreased significantly. In this paper, we will analyze data space after converting XML documents and propose algorithms to create and query XR-tree trees more efficiently, avoiding redundant tree browsing steps. With the new algorithms, we call the new XML document indexing method XR + tree.

The content of the paper is as follows, Part 2 will review recent research related to indexing bioinformatics XML documents. Part 3 will present the theorems and propose algorithms for the XR + tree method. Part 4 is experimental results and evaluations. Part 5 is the conclusion of the article.

II. RELATED WORK

Bioinformatic data compression has been studied for many years. Conventional bioinformatics XML documents include descriptive and DNA / Protein biological data. Therefore, there are two main research directions (1) compressing XML documents in general and (2) compressing biological data. We focus on compressing XML documents. Although there is no specific compression method for bioinformatic XML documents, recent research on computational XML document compresion has two directions, (1) compression methods that allow data to be retrieved without decompression, (2) compression method focuses on optimal compression ratio and only allows querying data after being decompressed. Because the data file size is very large, we think that the compression method allows a query to be necessary, so this paper focuses on the first one.

As a structured data model, the XML semi-structured data model also has the most popular private query languages, XPath and Xquery. XPath is the simple and basic form of Xquery. The working mechanism of the XPath query is divided into two forms: Sequential and indexed. With sequential form, XML data will not be preprocessed before querying. Each query will perform a full read of the data. The indexing form will pre-process XML data to build another data structure, so that the query does not require scanning the entire data.

Since the characteristics of the data structure after indexing are usually smaller in size, this is also a compression method that allows direct querying of compressed data. Here are some methods that have been implemented.

The first method is XGrind [1] by Tolani and Haritsa. This method compresses data by replacing attribute and element names into unique characters. The data section is encrypted using Huffman algorithm. The method allows direct querying on compressed documents but only supports simple path queries, incapable of supporting XPath queries. Similarly, Xpress [2] by Min et al. use inverse math method to encode XML document label paths into separate segments. Queries rely on each relationship between those segments to increase efficiency on compressed XML documents. The weakness of Xpress and XGrind is that the ratio of size between compressed and original data is linear, not supporting multiple queries. Therefore, Cheng and NG proposed the technique XQzip [3]. XQzip uses the Structure Index Tree (SIT) for the original XML document. And XQueC [4] by Arion et al. Use the structure summary tree to store XML documents. These methods give higher compression rates and faster queries. To increase the efficiency of XPath queries, Arroyuelo et al. [5] introduced the technique of creating a label tree from the XML tree structure, then using the array bit (array bit) index method to compress XML documents. The data section is compressed by any normal method. To reduce latency and optimize data transfer bandwidth, Qian et al. [6] provides a method of separating the XML document structure from the content and compression separately, giving better efficiency than the previous methods.

Compression methods tend to separate and compress XML document structures and data in XML documents. With the structure part, there are outstanding methods that have applied structural coding in the order of tree browsing.

Dietz [7] is the first method that uses structured coding techniques with algorithms to browse trees according to pre-order and post-order values. The weakness of this method is that it will have to recalculate the pre-order and post-order pairs each time there is new data. To overcome this problem, Li and Moon in XISS [8] used the B + tree method to index the element and attribute. However, the query does not perform well because of many intermediate results and every time a new node is created, it will have to be reorganized again.

In order to overcome the disadvantages of B + tree, some studies [12, 13] converted XML tree structure into 2-dimensional digital space, with pre-order and post-order pairs. In this space, the four main query paths for the XPath for an XML tag are the descendant, the ancestors, and the previous one (preceding sibling) and the following (following sibling) equally divided into four spaces. separate, symmetrical and evenly [14]. By discovering this distribution, we can find tags that relate to a given tag by creating search space windows to retrieve the required information. However, these windows are still very large spaces, which will adversely affect search performance. So far, two indexing methods for multidimensional spaces are grid (Grid) and tree (R-tree), these methods are very popular and available in database management systems. Grid method will be best with spatial data evenly distributed, whereas R-tree method is suitable for diverse distributed data. The XML document after converting to 2-dimensional space has a very diverse composition because the XML tree structure is unbalanced and heterogeneous [14]. Therefore, the R-tree method [9] is probably best suited for indexing XML documents. The biggest drawback of R-tree is that it must control the overlap problem between nodes of the tree, which causes redundant tree browsing steps. Moreover, this two-dimensional spatial data conversion method is still inherently flawed in that it will have to convert the space if there is a structural change of the XML document. However, in this paper we ignore this problem because it is assumed that bioinformed XML data is characterized by large size and almost unchanged when given to the community. And the main problem is to reduce the size and increase query efficiency.

Our previous paper [11] came up with the idea of increasing the efficiency of XPath axis queries for preceding tags (preceding sibling), the following (following sibling) and ancestors by adding pointers to leaf nodes of R-tree. Thanks to these pointers, other Xpath axis queries that result include sibling tags also benefit for better performance. However, the disadvantage of the method is that it is possible to create an unoptimal R-tree index tree. Therefore, the next content, we analyze the converted data space of the XML document and propose algorithms that can overcome some of the defects.

III. XR+ TREE METHOD

3.1 Analyzing the conversion data space of XML documents

Previous studies have shown that the distribution of XML data in space tends to focus on the two spaces of the previous tags (preceding sibling), the following (following sibling). Therefore, XR-tree has proposed to use more pointers to increase query efficiency.

In this section, we analyze the converted data space of the XML document and how to organize that data on the R-tree index tree. Suppose we have an XML document named X, to convert to a digital space, we need to use the "pre-order" algorithm to number each of the name tags of document X. See Figure 1 (round dot symbols are equivalent to 1 name tag in X, hierarchical structure is the same as the XML tree structure of document X), we see that the root node will have the order of 1, and the child nodes will have the order of increasing from left to right. Similarly, with the "post order" algorithm, the root node will have the last sequence number, and the leftmost descendant nodes will start from 1. After executing two tree browsing algorithms on document X, Figure 1 will show round dots with pairs of pre-order and post-order values, for example

the root node will have a value pair of 1-31, in That 31 is the total number of tag names of document X.

Thus, each tag name in document X has a Cartesian coordinate in 2-dimensional space. From now on, we can use spatial indexing methods for data of document X. In this paper, we use R-tree indexing method for 2-dimensional space. The traditional R-tree algorithm will group the space elements into each node on the R-tree and limit these groups to the smallest bounded rectangles of all elements of each group. In the previous article, to preserve the relationship between brother tags (sibling) of document X, we actively grouped brother tags (sibling) into leaf nodes of R-tree, and used the The pointer to connect the R-tree leaf nodes has brother (sibling) tags with the same father, this method is called XR-tree.

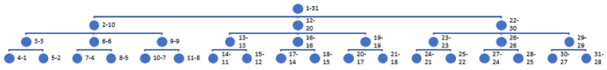

Figure 1. XML document tree is numbered

Figure 2 shows the leaf nodes of XR-tree in 2-dimensional space through the smallest bounding rectangles (MBR). Where, R is the MBR of the sub-tags of the original tag (1-31) of Figure 1. Similarly, R1, 2, 3 in Figure 2 are the MBRs of the sub-tags of the tags: 2-10, 12- 10, 22-30 in Figure 1, and the same for lower level tags. From the spatial visual description of Figure 2, we find the theorems about the MBR subtrees of the tags of document X when expressed in XR-tree space.

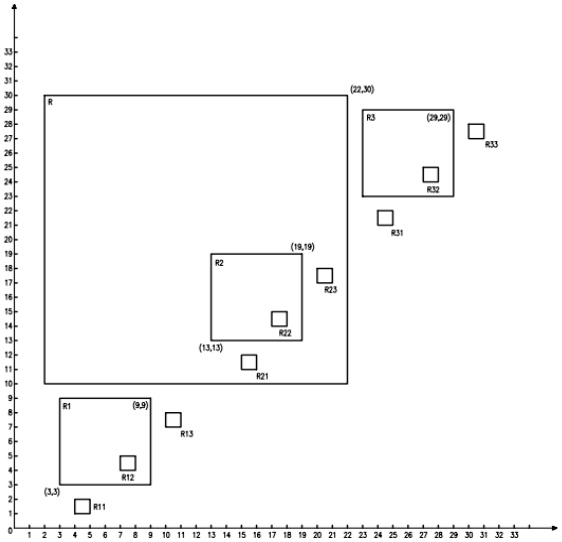

Figure 2. The MBRs of leaf nodes in the XR-tree tree

Theorem 1: Suppose in an XML document, the T tag is the father of the tags t1, t2, .., tn, they are brothers (sibling). The MBRs that contain subtrees of t1, t2, ..., tn on the XR-tree space are always separate (without intersection).

For example, R1, R2 and R3 are bounding rectangles of sub-trees that are always separate and spread from left to right on the space of Figure 2.

To prove this theorem, we easily recognize that the pre-order values of elements in the MBR for the sub tree of the brothers (sibling) tags on the left always have a value less than the pre-order values of the elements in the MBR for the sub tree of the sibling tags on the right. Similarly, on the contrary, the values on the left are always greater than the values on the right with the post order values. Therefore, MBRs for subtrees of sibling (sibling) tags in X documents on XR-tree space are always separate.

Thus, Theorem 1 shows that, forcing the sibling XML tags with the same parent into the same XR-tree leaf node may not have much effect on optimizing the R-tree tree structure for queries. Experimental results of the article on XR-tree have shown this observation.

Theorem 2: Suppose in an XML document, the T tag is the father of the tags t1, t2, .., tn, they are brother tags (sibling). Except for MBRs of subtrees of t1 and tn (first and last sibling), MBR including t1, t2, ..., tn will cover all MBRs of subtrees of t2, .., tn-1.

For example, R will cover all R2 and R21, R22, R23.

To prove this theorem, it is easy to recognize that pre-post values of t1 and tn are preceding and after the pre-post value of T. Therefore, pre-post values in the subtrees of t2, ..., tn are obviously within MBR range of t1 and tn.

From theorem 2, we draw a consequence 2.1 for the query algorithm to search for an XML tag in the XR-tree tree as follows

Consequence 2.1: Suppose when looking for a t tag of an XML document in the XR-tree search tree and finding the location of t located inside rectangles R1 and R2. If R1 is inside R2, the search algorithm will not need to continue browsing other subtrees of R2 to find t because it is certainly t in the subtrees of R1.

From the theorems and consequences, we redesigned the XR-tree algorithms to re-optimize queries, overcome structural weaknesses.

3.2 The proposed algorithm

From the conclusions obtained in Section 3.1, we propose new Insert and Query algorithms and we call the new extension method of XR-tree as XR + tree. The goal of the algorithm is to reduce redundant tree browsing steps to optimize query speed, while reducing the structural disadvantages of the XR-tree.

The XR + tree will remove the Parrent pointer (pointing to the parent node) because it has not really yielded the expected effect and consumes a lot of storage memory.

Insertion Algorithm

Algorithm Insert(N,E)
Invoke **FindNode(N,E)** to find a leaf node N' containing the sibling predecessor of the new context node entry E to be inserted (stage 1)
IF node N' is found,
IF N' is first node or last node
fullnode =m //m is minimum number of entries in a node
ELSE
fullnode =M //M is maximum number of entries in a node
IF |N'| <fullnode //has space, not full,
Insert new context node, E into N'.
ELSE
CreateNewLeafNode(E) //to create new leaf node for new context entry E, then insert the new leaf node into the tree
ELSE
CreateNewLeafNode(E) //to create new leaf node for new context entry E, insert the new leaf node into the tree
Creatpointers (pre, post, N')

The difference in this algorithm is that it will separate nodes containing the first / last sibling tags into separate leaf nodes in XR + tree. In particular, the nodes containing the first / last sibling tags will only contain up to m tags (minimum) instead of M (Maximum). Thus, the method will be able to reduce the size of the MBR, which covers all the sibling tags, so there will be fewer intersections between the MBRs of the XR + tree. Queries will benefit from this structure.

Querying Algorithm

Algorithm FindNode(N, E)
minN = N //only search N intersect or inside minN
IF N is NOT a leaf,
FOR EACH entry E' of N whose MBB (N') intersects with the
MBB of E
IF N' intersects or inside minN,
invoke **FindNode(N',E)**, where N' is the childnode ofN pointed to by E'.
IF N' inside N'
minN=N'
ELSE
IF N contains an entry E,
Return N.

This algorithm applies Theorem 2 and Consequence 2.1, which means that when searching for E, if R1 is inside R2, skip the subtree of R2 and save the R1 value.

At any intermediate node,

- If R contains R1, it will ignore the subtree
- If there is R inside R1, then save R for comparison in the next step, continue browsing R
- If R is not in the above 2 cases, continue browsing R.

IV. EXPERIMENT

To evaluate the actual effectiveness of the method, we experiment on bioinformatics XML documents of common and diverse origin.

We run experiments on computers with CPU configuration: Intel Xeon E5520 - 2.7 GHz, RAM: 20 GB, OS: Windows Server 2008 R2 Enterprise. The original documents are very large in size.

We use 4 different bioinformatics materials from reputable data sources. They describe different biodiversity, DNA, Protein, and description of subspecies, as follows

- **DNACorn:** Contains information describing Corn DNA such as authors, published years, genes, ... originating from NCBI,
- **DNARice:** Contains descriptive information about Rice DNA such as authors, published years, genes, ... originating from NCBI
- **Swissprot:** Collection of functional information on proteins, with accurate, consistent and rich annotation. Source: https://www.uniprot.org. From (swissprot.xml), which represent the description of all proteins.
- **Allhomologies:** contains information on the subspecies of the mouse family. Source from ftp://ftp.ensembl.org/

4.1 Experimental compression of bioinformatics XML documents

To appreciate the practical effect of compressing data by transferring the bioinformed XML document to digital spatial data, we have experimented on actual documents as described above. The results are shown in the graph in Figure 3. Figure 3 shows quite good compression ratios, particularly with DNA XML documents. However, with the document Allhomologies describing subspecies information, it is quite surprising that there is a low compression ratio.

To understand why the compression ratio differs between documents, we have analyzed the XML files and realized that the Allhomologies document describes species tree information, so the data has a lot of Attribute tags. The feature of the Attribute tags is to be declared as a collection in the same string. Therefore, the conversion algorithm when meeting the Attribute tag set will have to separate each Attribute into multiple tags. This may be the reason for increasing the size of conversion documents.

Thus, this conversion method does not always have a good compression ratio because it depends on the structure of the XML document.

In this paper, we only contribute to the actual discovery of the compression algorithm by converting to digital space. Later research, we will find ways to solve the problem.

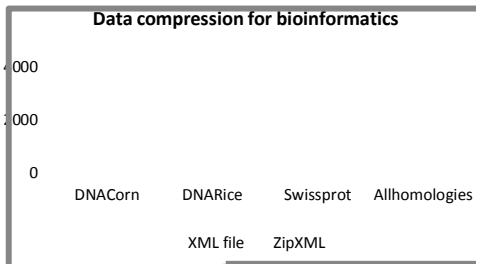

Figure 3. Compare XML documents and conversion documents to digital space

4.2 Experimental queries

The experimental scenario is to execute queries according to the key and important axes of XPath. You can split these queries into two groups

- Point queries: queries for sibling, children, Parent.
- Regional queries: queries for the other XPath axis.

To evaluate average performance of point type queries, we generate 200 random queries for index trees for each XML document, to compare the performance of XR-tree and XR + tree. The result is evaluated based on the number of nodes accessed by the query, in other words, the number of access nodes is the number of accesses to the hard drive to get the data to the memory. The lower the average number of hard disk visits, the higher the query's performance.

a. Sibling queries

The query must find all the brother tags (sibling) of any tag in the XML document. Figure 4 shows that the average XR + tree performance result is always better than XR-tree for all XML documents. And it looks with material Allhomologies, experimental for best results.

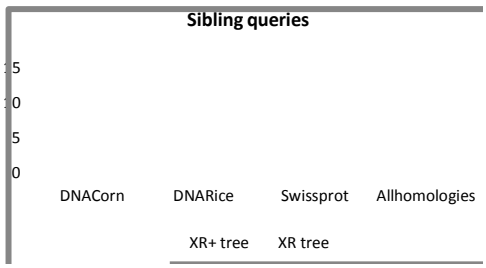

Figure 4. Sibling query result

b. Preceding Sibling queries

The query only looks for brother tags (sibling) in front of any tag in the XML document. Figure 5 also shows that the average XR + tree performance results are always better than XR-tree for all XML documents. And it seems that with allhomologies material, experiments still give the best results.

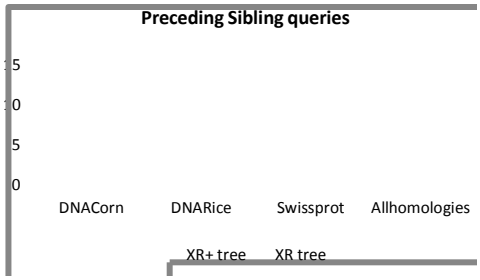

Figure 5. Query for Preceding sibling

c. Following Sibling queries

The query only looks for brother tags (sibling) behind any tag in the XML document. Figure 6 also shows that the average XR + tree performance results are always better than XR-tree for all XML documents. However, the results for DNARice documents seem much different. This proves that there are XR + tree situations that will give very good results when the query removes many steps of redundant tree browsing according to Theorem 2 and consequence 2.1.

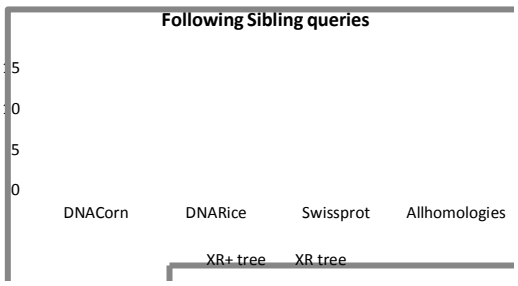

Figure 6. Query for Following sibling

d. Children queries

These queries need to find the child tags of any tag in the XML document. To do this, it is best to estimate a sub-tag (first or last), then use the sibling query to find those sub-tags. As such, this query will become the sibling query as above. However, in this paper, we create a zone query with a limited search window that can find subtags of any tag to evaluate the performance of the regional queries.

Figure 7 also shows that the average XR + tree and XR-tree performance results are basically the same for all XML documents. It seems that the limited search window is good enough to not see the difference between the two tree structures.

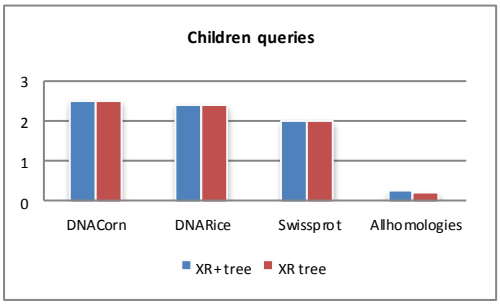

Figure 7. Query for Children

e. Regional queries

In addition to the queries on the XPath axes as above, the remaining queries use zone windows that have limited scope of space, we experiment with these zone queries together. Experiments on these queries will not take advantage of pointers that connect leaf nodes of XR tree and XR + tree. Thus, the results are evaluated completely based on the optimization of the tree structure. To objectively compare query performance between two XR-tree and XR + tree structures, we use 50 fixed (non-random) queries to experiment with.

The results in Figure 8 show that not all documents give different query results, but there are still XML documents where the XR + tree structure provides better performance than XR tree structure.

From the above experiments, it has been shown that the construction of algorithms according to the proposed theorems and consequences is effective, not only queries related to brother tags (sibling) but also and for Common queries thanks to more optimal tree structure.

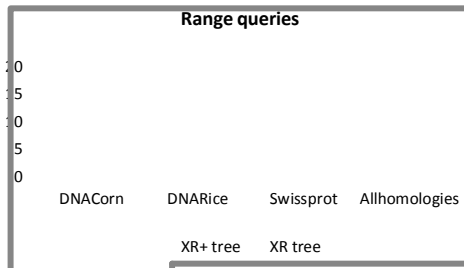

Figure 8. Query practice for Range query

V. CONCLUSION

The goal is to find new indexing methods that can retrieve information of large bio-sized XML documents and at the same time reduce storage size. This article analyzed the correlation between XML tree structure after conversion to digital spatial data and XR tree tree structure. From there, give theorems and consequences to apply the construction of new algorithms, the new method is XR + tree.

Experimental results have shown that the new XR + tree indexing method is superior to XR-tree in most XPath axes and regular queries. The experiments have used bioinformatic data sets from reputable sources and described different biodiversity, the purpose to confirm the objectivity and practicality of the algorithms.

Although the original R-tree algorithm has many other variants such as R * tree, R + tree ... these variants only apply to specific and special situations that are not applicable in commercial database management systems. Therefore, we have upgraded this original algorithm so that it can be applied in practice.

Further research will continue to improve the original algorithms and apply on database systems that support R-tree such as SQL server, Big data to be able to test on very large bioinformatics databases.

ACKNOWLEDGMENT

This paper is supported by project PTNTĐ19.05 of Key lab of Network technology and Multimedia, Vietnam Academy of Science and Technology (VAST), Hanoi, Vietnam.

REFERENCES

1. Tolani, P.M., J.R. Haritsa. XGRIND: A Query-friendly XML Compressor. IEEE 18th international conference on Data Engineering, 2000.
2. Min, J.K., M.J. Park, C.W. Chung. XPRESS: a queriable compression for XML data. Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM, San Diego, California, 122133, 2003.
3. Cheng, J., W. Ng. XQZip: Querying Compressed XML using Structural Indexing. International Conference on Extending Data Base Technology (EDBT) , 2004.
4. Arion, A., A. Bonifati, I. Manolescu, A. Pugliese. XQueC: A query-conscious compressed XML database. ACM Trans. Internet Technol., 7: 10, 2007.
5. Arroyuelo, D., F. Claude, S. Maneth, V. M¨Akinen, G. Navarro, K. Nguyen, J. Sir´En, N. V¨Alim¨Aki. Fast In-Memory XPath Search using Compressed Indexes. In Proceedings of the IEEE Twenty-Sixth International Conference on Data Engineering (ICDE 2010), California, USA, 2010.
6. Qian, B., H. Wang, J. Li, H. Gao, Z. Bao, Y. Gao, Y. Gu, L. Guo, Y. Li, J. Lu, Z. Ren, C. Wang, X. Zhang. Path-Based XML Stream Compression with XPath Query Support Web-Age Information Management. Springer Berlin / Heidelberg, 2012.
7. P. Dietz. Maintaining order in a linked list. Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, ACM, 1982, 122–127.
8. Q. Li, B. Moon, et al. Indexing and querying XML data for regular path expressions. Proceedings of the International Conference on Very Large Data Bases, 2001, 361–370.
9. Guttman. R-Trees: A dynamic index structure for spatial searching. Proceedings of SIGMOD, Boston, Massachusetts, 1984.
10. Norah Saleh Alghamdi, Wenny Rahayu, Eric Pardede. Semantic-based Structural and Content indexing for the efficient retrieval of queries over large XML data repositories. Journal of Future Generation Computer Systems. July 2014, 212–231.
11. Hoang Do Thanh Tung, Dinh Duc Luong "An Improved Indexing Method for XPath Queries", Indian Journals of Science and Technology, Vol 9(31), 10.17485/ijst/2016/v9i31/92731, August 2016.
12. T. Grust, M. Van Keulen. Tree awareness for relational DBMS kernels: Staircase join. Journal of Lecture Notes in Computer Science, 2003, 231–245.
13. T. Grust, M.V. Keulen, J. Teubner. Accelerating XPath evaluation in any RDBMS. Journal of ACM Trans. Database Syst, 2004, 91–131.
14. Torsten Grust. Accelerating XPath loation steps. In SIGMOD Conference, 2002
15. Baydaa Taha, Raad Alwan. Bioinformatics Data Compression and Retrieval Based on XML Structured Indexed Tree. Australian Journal of Basic and Applied Sciences 8(83):35-42. April 2014

AUTHORS PROFILE

Born on December 4, 1978. Graduated from Dongdo University (2001). Master of Science thesis in mathematics and computer science at University of Science, Vietnam National University (2010). Currently working at the Food Industrial College. Research Interest: bioinformatics, big data.
Email: luongdd@fic.edu.vn or luongdd.tp@gmail.com

Born on July 10, 1988 in Hanoi. Graduated from Hanoi University of Industry (2010). Master of Science thesis in Computer Science at Posts and Telecommunications Institute of Technology (2014). Currently working at the Institute of Information Technology (IOIT) - Vietnam Academy of Science and Technology (VAST). Research Interest: Bioinformatics, Big Data, Indoor location
Email: phuongvq88@gmail.com

Born on May 9, 1976 in Hanoi. Graduated from Vietnam National University, Hanoi, Vietnam (1998). Master of Electrical & Computer Engineering, Chungbuk National University, South Korea (2004). PhD of Electrical & Computer Engineering, Chungbuk National University, South Korea (2007). Currently working at the Institute of Information Technology (IOIT)-Vietnam Academy of Science and Technology (VAST). Research Interest: Business intelligence/Model, Analysis and Decision support, Big data, Bioinformatics, Mobile technology, In-memory database, Real time database, Continuous queries. Email: tunghdt@ioit.ac.vn or tunghdt@gmail.com