MARS: A Matching and Ranking System for XML Content and Structure Retrieval

S. Alireza Aghili Hua-Gang Li Divyakant Agrawal Amr El Abbadi

Compute Science Department
University of California, Santa Barbara
Santa Barbara, CA 93106, USA.
{aghili,huagang,agrawal,amr}@cs.ucsb.edu

Abstract

Structural queries specify complex predicates on the content and the structure of the elements of tree-structured XML documents. Recent works have typically applied top-down decomposition of the twig patterns into (i) parent-child or ancestor-descendant relationships, or (ii) path expression queries, and then followed by a join operation to reconstruct matched twig patterns. This demonstration system is the implementation prototype of an efficient heuristic-based bottom-up approach named MARS (**M**atching **A**nd **R**anking **S**ystem for xml structure queries), for matching and ranking of structural query patterns for XML query processing. An efficient nearest common ancestor labeling scheme is applied to utilize fast bottom-up construction of the subtree matches from the potential keywords. MARS considers both the content and structure of queries and incorporates a variation of IR-based relevance ranking to report the top-k ranked results. The graphical user interface of MARS provides an interactive visualization of the twig query discovery.

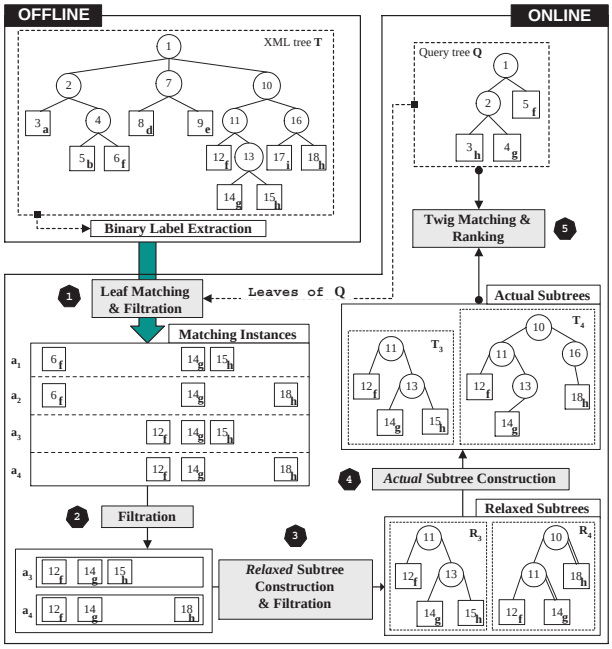

Figure 1: MARS procedure: a general overview.

1 Motivation of MARS

Structured twig queries specify complex patterns of selection predicates on element *labels* (keyword search) and their corresponding inherent *structural relationships*. XML query languages [1, 2, 3, 4] and the underlying relational [5, 7] or native [6, 8] databases must provide efficient and effective support for querying both the *content* and the inherent *structure* of XML documents. However, the support for structural queries is usually facilitated by augmenting a layer of structural search on top of the traditional path expression query language performing the following general steps: ***(i)*** Decompose the twig query structure into its corresponding path expression queries. For instance, (`/dblp/ /journal[author='Serge' AND title='XML']`) gets decomposed into (`/dblp//journal [author='Serge']`) and (`/dblp//journal[title='XML']`) path queries, ***(ii)*** Perform a top-down search in the XML document tree for each decomposed path expression of the query, ***(iii)*** Join the results of each of the query's path expressions to form the result to the original twig query.

However, the top-down traversal of the document tree results in scanning a large number of path combinations. In the worst case, the number of intermediate results (number of decomposed path expressions) is proportional to the size of the document tree which imposes serious scalability and efficiency drawbacks. For instance, the root node of the *dblp*[1] database has almost 3,288,858 immediate children which makes it impractical to inspect all the path combinations if the given query includes the root node. While some optimizations may be applied, it is inevitable that quite

[1] *http://www.informatik.uni-trier.de/∼ley/db/*

a large number of path expressions have to be generated and scanned. MARS employs a low-cost bottom-up tree matching technique to the given problem which *avoids* the decomposition of the twig query and inspecting the large number of intermediate results (path combinations).

The main contributions of the MARS system proposal are summarized as follows:

- An efficient *binary labeling* scheme based on the notion of *Nearest Common Ancestor (NCA)* is introduced for fast construction of a subtree from its leaves. Given a set S of leaves (or leaf keywords) of an XML document tree T, MARS constructs of the "*subtree of T induced by S*" in *constant time*.
- MARS supports approximate keyword search using string matching techniques, and approximate structure and path matching using a dynamic programming algorithm.
- A novel combination of an IR-based keyword and structure relevance ranking technique along with the path-level minimum tree edit distance is employed to compare and rank the subtree matching instances, based on their content and structure.
- Using the binary labeling scheme, various filtration techniques (schema-based, horizontal and vertical filtrations) are integrated to prune irrelevant matching instances and efficient reduction of false positives.

2 MARS: a general overview

In this section, we decompose the MARS into its primary component units and provide a general overview of the system, as depicted in Figure 1. The algorithm starts with an offline stage which associates unique binary labels to each node of the tree to facilitate fast detection of the nearest common ancestor (NCA) of the nodes. The main properties of the binary labeling procedure are: ***(i)*** The process of assigning labels to each node of the tree takes O(n) computation time which is performed as an *offline* procedure, ***(ii)*** The *unique* binary label assigned to each node of the tree is of length O($log\ n$) bits, ***(iii)*** Given the labels of any two nodes, the label of their NCA can be determined in *constant time*, ***(iv)*** Given a tree T and any subset S of its leaves in order of their *pre-order traversal rank*, the subtree of T restricted to the nodes of S can be constructed in O($|S|$) time.

Given the structural query pattern Q and XML document tree T, the online phase of MARS is performed in four stages as follows:

1) ***Leaf Matching, and Filtration***: Leaf matching, finds all occurrences of Q's leaf keywords in the leaf nodes of T resulting in the potential leaf-level matching instances. MARS also incorporates an efficient *structure filtration* when dealing with multiple XML document collections each having a different DTD (Document Type Definitions). It is much more likely that the answers to a query Q originate from those XML documents whose DTD resembles the structure imposed by Q. MARS employs the DTD-based filtration step for early pruning of irrelevant documents, which drastically reduces the size of search space.

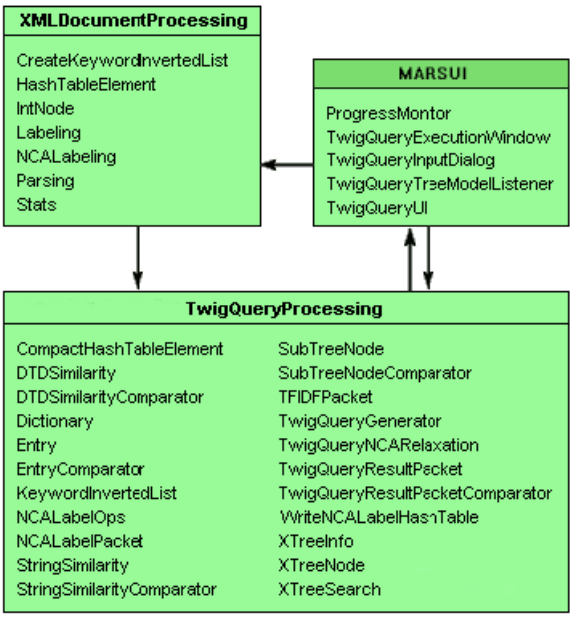

Figure 3: MARS System Architecture.

2) ***Filtration***: This is the second filtration step in MARS which attempts to remove most of the irrelevant matching instances using the extracted binary labels. This filtration step also uses the *horizontal* and *vertical* distances (*extent*) of the tree nodes to determine the relevance of the nodes and further pruning of the matching instances and reducing false positives.

3) ***Relaxed/Actual Subtree Construction, and Filtration***: For any remaining matching instance, this stage performs a bottom-up approach to construct a *relaxed* subtree of T restricted to the nodes of the matching instance using the binary labels of the nodes. The resulting relaxed subtrees are more compact than the actual subtrees of T restricted to the nodes of the matching instance. The *compactness* of the constructed relaxed subtrees depends on the internal nodes involved in the matching instance and is also a function of vertical distance among the corresponding nodes of the matching instance. This construction and filtration stage, benefits from the *compactness* of the relaxed subtrees and further uses the corresponding *NCA* label of the nodes for additional filtration of the intermediate results using schema comparison, vertical/horizontal distances of the nodes, and path matching on the potential results. Each remaining relaxed subtree is finally extended to construct the *actual* subtrees of T.

5) ***Twig Matching and Ranking***: The resulting actual subtrees are compared against the query twig for similarity comparison and ranking. A novel combination of an IR-based keyword and structure relevance ranking technique, and the path-level minimum tree edit distance is employed to compare each actual subtree against Q and ranking the subtrees among each other. This measure captures the *similarity* and *significance* of the structural results, by incorporating both the *content* and *structure* of the nodes.

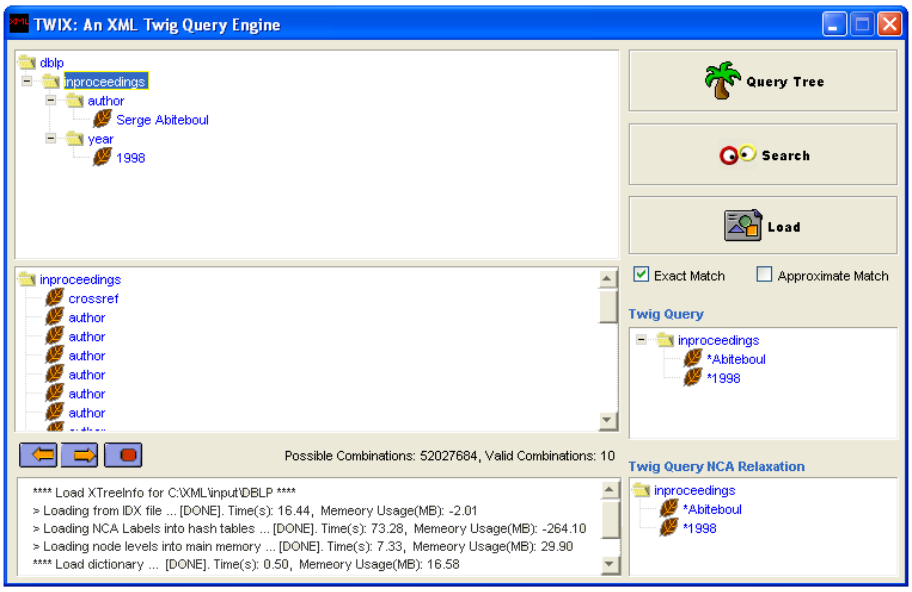

Figure 2: MARS Graphical User Interface.

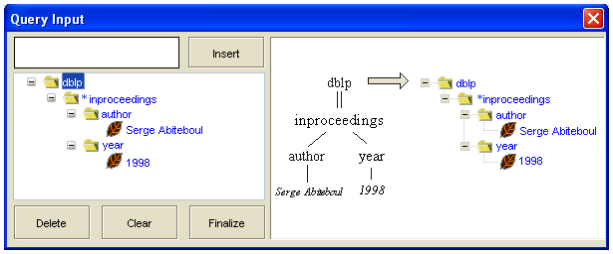

Figure 4: Query Data Entry Console.

3 System Architecture

Figure 3 depicts a general overview of the system architecture and components of MARS. MARS consists of three main components which are summarized as follows:

1. XMLDocumentProcessing: This is an *offline* component responsible for parsing, and creating the necessary index structures on each node of the XML document tree. It proceeds by associating each node of the tree with a unique binary label which will be used in the subsequent components for subtree construction and filtration. It also provides an inverted list index structure along with a keyword dictionary to efficiently maintain the location of keywords in the system.

2. MarsUI: The graphical user interface (GUI) of the system is handled through MarsUI component which is used for the offline loading of the data, online entry of the twig query and browsing through the results.

3. TwigQueryProcessing: This component is the core of the twig query structure matching which incorporates various modules and data structures to facilitate filtration, subtree generation, string/path similarity, and ranking of the results. The auxiliary index files and data structures that were generated in the XMLDocumentProcessing component are used in this stage for manipulation and traversal of the document tree. TwigQueryProcessing interactively communicates with the MarsUI component for browsing through and further refinement of the produced ranked results. The arrows are used to browse through the discovered patterns and the bottom-center box displays the content information and the statistics of each search.

4 Demonstration Overview

We will demonstrate a functional implementation of the MARS system. MARS and its components are implemented in *Java 1.4.2* with a graphical user interface. The following features of the MARS system will be demonstrated:

- **XML Document Entry and DTD Visualization:** Users may browse, visualize and select various XML documents for querying. A parallel program provides a visualization of the chosen document's DTD to the user, as well. We will have some known real XML documents (e.g. `dblp`, `SIGMOD Record`, `NASA`, ...) pre-loaded for more efficient demonstration purposes. Figure 2 is the main GUI window of the MARS, which is used for loading, search and browsing through the ranking of the structural queries.

- **Query Tree Data Entry Console:** Users may use any of the various pre-loaded queries or may specify their own queries using the user-friendly JTree input interface of MARS, as shown in Figure 4. The right side of the window is a static figure showing an example of the how the query should be entered, and the left portion of the window is blank in the initialization. Users input the queries to the system, node by node which will be displayed in the left portion of the window, and the edges are inserted between each pair

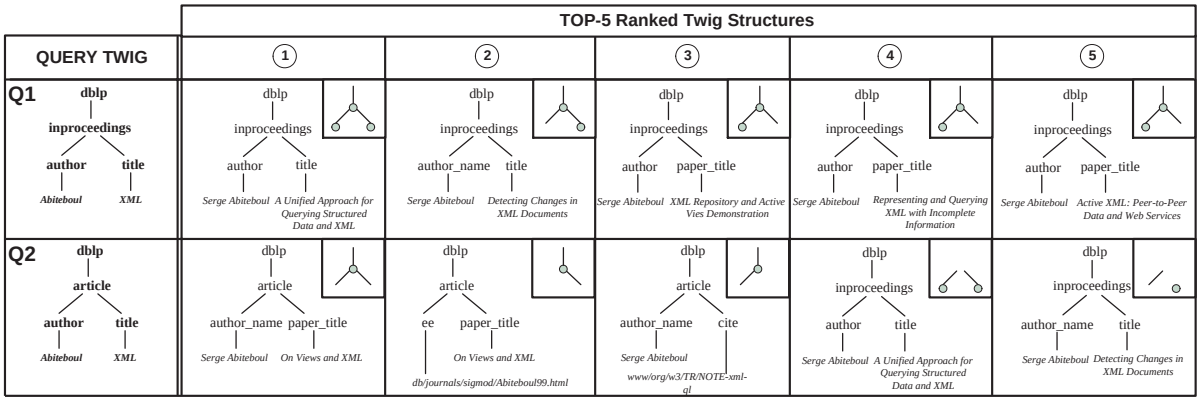

Figure 5: TOP-5 ranked twig answers to the given queries.

of nodes depending on the location of the insertion. The input query is displayed in browsable tree format where the modification of the query may easily be performed at any level of the tree. For instance, in Figure 4 the `dblp` node (on the left window) is being highlighted/selected. *Insertion*, *deletion* and *modification* of the nodes are provided in the user interface. The parameterized ranking values/thresholds of the algorithm may also be entered to the system, if the user decides to incorporate more sophisticated/weighted ranking parameters. Once the query entry is done, pressing on the button `Finalize` will take the user to the main GUI of MARS (Figure 2) and the query twig would be displayed in the upper-left portion of the screen.

- **Result Ranking:** Figure 2 depicts a snapshot of the main GUI component of MARS used for browsing through the ranked subtree results. The total number of combinations and the number of valid combinations are displayed and users may use the provided arrows to inspect the discovered patterns. The upper-left window shows one of the twig query matches and more details on each node of the this matching instance may be shown in the lower window, by simply highlighting the corresponding node. For instance, in Figure 2, the nodes `inproceedings` is highlighted and as a result its components are displayed in the middle window. The Figure 2 also provides the choice of performing *exact* or *approximate* matching of the query tree. Approximate matching incorporates bottom-up relaxation of the query tree and returns the ranked answers, as well as, the exact answers. The ranking of the results are only used in the approximate matching option of the MARS and more information regarding the rank values and their corresponding orders are displayed accordingly.

  Moreover, Figure 5 depicts a visual perspective and analysis into the highly ranked matching instances returned by MARS on a collection of 14 versioned `dblp` datasets. The matching instances are listed in the decreasing order of their corresponding relevance ranks (rank 1 denoting the *best match*). A visual legend on the upper-right corner of each twig match depicts the parts of the result, *edges (structure: line)* and *nodes (content: circle)*, which match with the query and hence contributing to the ranking computation. Moreover, MARS clusters the keywords into *conceptual groups* (e.g. `author`, `author_name`, ..., `authors`) covering the semantics of the keywords into account. This feature facilitates a flexible ranking scheme, not imposing much constraints on the user's knowledge of underlying database. The Queries are shown in the left portion of the figure followed by their corresponding top-5 ranked results. The internal nodes of the queries were particularly chosen to capture the *element label* differences among the XML documents of the collection. The first answer to `Q1` is an exact match and the similarity of the matches decrease traversing through the ranked matching list. However, for `Q2`, the first answer is an approximate match. Going down the ranking results, validates the quality of MARS ranking in distinguishing matching instances from each other.

References

[1] D. Chamberlin, J. Robie and D. Florescu, Quilt: An XML query language for heterogeneous data sources. WebDB (Informal Proceedings), 53–62 (2001).

[2] D. Chamberlin, Daniela Florescu, Jonathan Robie, Jérôme Siméon and Mugur Stefanescu, XQuery: A Query Language for XML. W3C Working Draft, http://www.w3.org/TR/xquery (2001).

[3] J. Clark, XSL Transformations (XSLT), Version 1.1 W3C Recommendations, http://www.w3.org/TR/xslt11 (2001).

[4] A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu, XML-QL: A query language for XML. QL, http://www.w3.org/TR/NOTE-xml-ql (1998).

[5] D. Florescu and D. Kossman, Storing and querying XML data using an RDBMS. IEEE Data Engineering Bulletin **22(3)**, 27–34 (1999).

[6] J.F. Naughton et al., The Niagara Internet query system. IEEE Data Eng. Bulletin **24(2)**, 27–33 (2001).

[7] J. Shanmugasundaram et al., Efficiently Publishing Relational Data as XML Documents. VLDB, 133–154 (2000).

[8] Xyleme, Available from *http://www.xyleme.com*.