Semantic Based XML Context Driven Search And Retrieval System

[1,]M.V.V.Nagendra, [2,]K.Devi Priya, [3,]Vedula Venkateswara Rao

1,M.Tech(Final), Dept Of CSE
[2],Assistant Professor, Dept Of CSE
[3], Associate Professor, Dept Of CSE
[1,2,]Aditya Engineering College Kakinada, India
[3],Sri Vasavi Engineering College Tadepalligudem, India

Abstract

we present in this paper, a context-driven search engine called XCD Search for answering XML Keyword-based queries as well as Loosely Structured queries, using a stack-based sort-merge algorithm. Most current research is focused on building relationships between data elements based solely on their labels and proximity to one another, while overlooking the contexts of the elements, which may lead to erroneous results. Since a data element is generally a characteristic of its parent, its context is determined by its parent. We observe that we could treat each set of elements consisting of a parent and its children data elements as one unified entity, and then use a stack-based sort-merge algorithm employing context-driven search techniques for determining the relationships between the different unified entities. We evaluated XCD Search experimentally and compared it with other search engines. The results showed marked improvement .Research works propose techniques for XML Loosely Structured querying, where the user provides search terms consisting of label-keyword pairs. Computing the Lowest Common Ancestor (LCA) of elements containing keywords is the common denominator among these proposed techniques. Despite the success of the proposed search engines, they suffer recall and precision limitations. The reason is that they employ mechanisms for building relationships between data elements based solely on their labels and proximity to one another while overlooking the contexts of the elements. The context of a data element is determined by its parent, because a data element is generally a characteristic of its parent. We propose in this paper a search engine called XCD Search that avoids the pitfalls of non-context driven search engines. This paper presents a data-centric approach to XML information retrieval which benefits from XML document structure and adapts traditional text-centric information retrieval techniques to deal with text content inside XML. Document. Narrower contexts could be separate XML elements or their combinations. Our setting assumes.

Keywords- Searcht; XML;Contexte;Retrieval; LCA;Query.

1. INTRODUCTION

The World Wide Web (WWW) is a continuously expanding large collection of hypertext documents [1]. It represents a very large distributed hypertext system, involving hundreds of thousands of individual sites. It is a client-server based architecture that allows a user to initiate search by providing keywords to a search engine, which in turn collects and returns the required web pages from the Internet. Due to extremely large number of pages present on the web, the search engine depends upon crawlers for the collection of required pages. A Crawler [2] follows hyperlinks present in the documents to download and store web pages for the search engine.It is becoming increasingly popular to publish data on the Web in the form of XML documents. Current search engines, which are an indispensable tool for finding HTML documents, have two main drawbacks when it comes to searching for XML documents. First, it is not possible to pose queries that explicitly refer to meta-data (i.e., XML tags). Hence, it is difficult, and sometimes even impossible, to formulate a search query that incorporates semantic knowledge in a clear and precise way. The second drawback is that search engines return references (i.e., links) to documents and not to specific fragments thereof. This is problematic, since large XML documents (e.g., the XML DBLP) may contain thousands of elements storing many pieces of information that are not necessarily related to each other.

The popularity of the e**X**tensible **M**arkup **L**anguage (XML) has led large quantities of structured information to be stored in this format. Due to this ubiquity, there has lately been interest in information retrieval (IR) from XML. XML-IR presents different challenges than retrieval in text documents due to the semi-structured nature of the data. The goal is to take advantage of the structure of explicitly marked up documents to provide more focused retrieval results. For example, the correct result for a search query might not be a whole document, but a document fragment. Alternatively, the user could directly specify conditions to limit the scope of search to specific XML nodes. Previous work [2, 4] addresses several challenges specific to retrieval from XML documents: *Granularity of indexing units* (Which parts of an XML document should we index?) (2) *Granularity of the retrieved results* (Which XML nodes are most relevant?)(3) *Ranking of XML sub-trees* (How should the ranking depend on the type of enclosing XML element and term frequency/inverse document frequency (*tf-idf*)?Keyword search provides a user-friendly information discovery mechanism for web users to easily access XML data without the need of learning a structured query language or studying

possibly complex and evolving data schemas. How- ever, due to the lack of expressivity and inherent ambiguity, there are two main challenges in performing keyword search on XML data intelligently.

1. First, unlike XQuery, where the connection among data nodes matching a query is specified precisely using variable bindings and *where* clauses, we need to automatically connect the keyword matches in a meaningful way.
2. Second, unlike XQuery, where the data nodes to be returned are specified using a *return* clause, we should effectively identify the desired return information. Several attempts have been made to address the first challenge [3, 2, 9, 7] by selecting and connecting keyword matches through a variant concept of lowest common ancestor, named as *VLCA* (such as SLCA [9], MLCA [7], interconnection [2], etc.). However, it is an open problem of how to automatically and effectively infer *return nodes*, the names of the data nodes that are the goal of user searches.

2. LITERATURE SURVEY

XML retrieval systems vary according to the query language, index structure, document preprocessing, indexing and scoring algorithms they employ. A great variety of XML query languages already exist. Standard ones proposed by W3C are XPath and XQuery. Unfortunately, they do not reflect IR properties such as weighing, relevance-oriented search, data types and vague predicates, structural relativism. A similarity based crawler that orders URLs having target keyword in anchor text or URL, was probably one of the first efforts towards focused crawling [9]. The basic focus was to crawl more important pages first i.e. to look at various measures of importance for a page such as similarity to a driving query, number of pages pointing to this page (back links), page rank, location, etc. The Page Rank algorithm [11] computes a page's score by weighing each in-link to the page proportionally to the quality of the page containing the in-link. Thus, a web page will have a high page rank, if the page is linked from many other pages, and the scores will be even higher if these referring pages are also good pages, i.e. having high Page Rank scores. In the HITS (Hyper-link-induced- topic search) algorithm [8], an authority page is defined as a high quality page related to a particular topic or search query and a hub page is one that provides pointers to other authority pages. Based upon this, a web page is associated with an Authority Score and a Hub Score that is calculated to identify the web page context. Another focused crawler [7] employs seed keywords which are used to find seed URLs from some standard search engine like Google. The seed URLs are used to fetch seed pages with the help of TF.IDF algorithm, based on iteratively calculating word frequency. This algorithm is used to find out some more number of keywords from seed web pages to represent the topic. Afterwards, vector similarity is computed between web page and topic keywords to see whether the page is relevant to the topic. A critical look at the available focused crawlers [5-9,11] indicates that these crawlers suffer from the following drawbacks :

[1]. The problem of iterative computation of word frequency for every web document renders the search process expensive.
[2]. The relevance of web page is not known until it is downloaded.
[3]. Associated context of the web page is unknown prior to search initiation.
[4]. The user interface of a search engine with keyword search is not flexible.

3. SYSTEM ARCHITECTURE

We implement our system in an XML retrieval library. Each document corpus has a separate XML index in a central *XML Index Repository*. Each index has its own *model*, defined in an *indexing schema.* The indexing schemas specify how to extract indexing units (XML subtrees) called *contexts* from XML documents with a common structure and defines how the separate *contexts* are related. Our basic assumption is that a *context* is a document fragment whose content is indexed in several text fields. A *field* is a name-value pair whose value is character data. *Contexts* and *fields* can be referred to in search queries. An indexing schema is added to an empty index on its creation. Existent XML indices are populated with documents according to the definitions in their corresponding indexing schema. For each context defined in the indexing schema we create and populate a full-text index called *context index.* The rest of the subsection gives an overview on the system architecture (Figure 1).

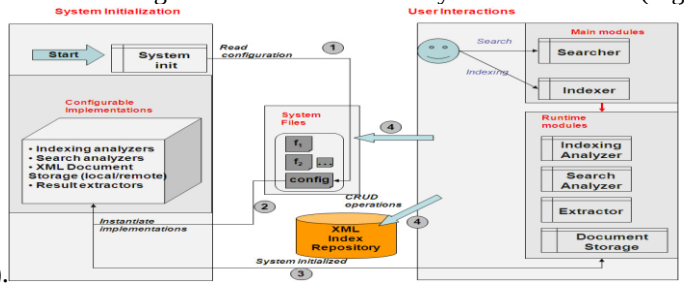

Figure1. System Architecture.

The search engine lifecycle has two sequential phases: *system initialization* and *user interactions*. When the system is started a processing module reads system configurations from a configuration file. This file specifies an indexing/search analyzer, document storage, and result extractor implementation classes. The opportunity to configure different implementations makes our framework highly adaptable to various search scenarios. We can use this to tune text analysis, document access policy, result formatting and extraction. Instances of the configured implementations are created by reflection and are made available at runtime. The indexing and retrieval tasks are performed by the *indexer* and *searcher* operational modules which manipulate the XML indices, system files and interact with the already instantiated objects. The Architecture of system also explained using following block diagram which is called as architectural diagram.

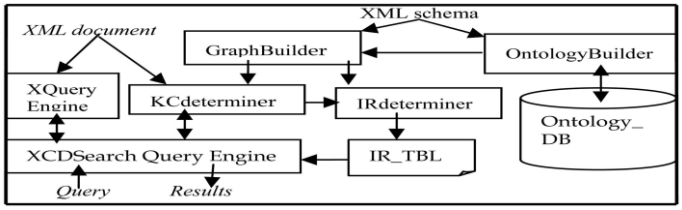

Figure2. Architetcure

The Architecture contains following sections.

1) Ontology Database.
2) Ontology Builder.
3) XQuery Engine 4) XCD Search Query Engine

4. METHODOLOGY

A. XCD Search

The structure of an XML document can be partitioned into multiple units (e.g., sub-trees), where each unit is associated with some document contents. Thus, the framework of XCD Search partitions XML trees to sub-trees, where each consists of a parent and its children data nodes and is treated as one unit. The idea of viewing each such set as one logical entity is useful as it enables filtering many unrelated groups of nodes when answering a query. Two real-world entities may have different names but belong to the same type, or they may have the same name but refer to two different types. To overcome that labeling ambiguity, we observe that if we cluster Canonical Trees based on the ontological concepts of the parent nodes' component of the Canonical Trees, we will identify a number of clusters.

Determining related keyword context

The process of answering a query goes through three phases. In this the system locates the KCs. In KC's each iteration of the algorithm, the current KC being processed is called the Current Context (CC) and prior KC processed is called Prior Context (PC).

Ontology label (OL)

Ontology label is nothing but the ontological concept of a parent node. The ontological concepts of "student" and "conference" are "person" and "publication proceedings, "respectively.

Canonical Tree (CT)

The structure of an XML document can be partitioned into multiple units (e.g., sub trees), where each unit is associated with some document contents. The document content consist of a parent and its children data nodes and is treated as one unit. These partition sub trees are called canonical tree.

Intended Answer Node (IAN)

IAN is a data node in the XML tree containing the data that the user is looking for.

The Following is an example of Ontology Hierarchy.For example, a student "is a" person. m' is the most general super class (root node) of m in a defined ontology hierarchy. If m is an interior node's label, m' is called the Ontology Label of m and is expressed as OL(m)=m'. Fig. 3 shows an example of ontology hierarchy.

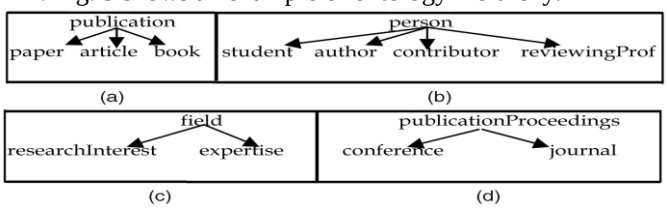

Figure 3 Ontology Hierarchy.

The XCD Search Query Engine Converts the xml document into xml tree using XML Parser and then constructs Canonical Tree Graph for finding relations between CT's. using CT's the Search is performed. The Following diagrams(Figure 4 & Figure) shows XML Tree and CT Graph.

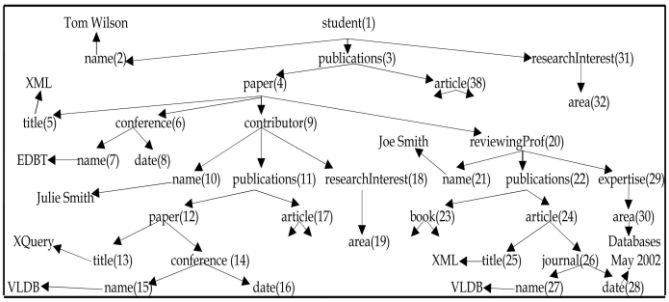

Figure 4 XML Tree

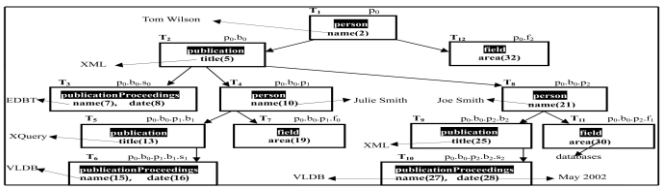

Figure5 CT Graph.

B. KeyWord Search Semantics

Identifying and Connecting Keyword Matches. XSeek adopts the approach proposed in for defining CT nodes and connecting keyword matches through their CT nodes. An XML node is named as a CT node if its subtree contains matches to every keyword in the query, and none of its descendants contains every keyword in its subtree. Keyword matches in a subtree rooted at a CT node are considered as closely related and are connected through the CT node; while the matches that are not descendants of any CT node are determined as irrelevant and discarded.

C. Analyzing Keyword Patterns

Analyzing Keyword Patterns. XSeek also analyzes key-word patterns to determine return information. It classifies input keywords into two categories: predicates and return nodes. Some keywords indicate *predicates* that restrict the search, corresponding to the *where* clause in XQuery. Others specify the desired output type, referred as *explicit return nodes*, corresponding to the *return* clause in XQuery.

D. Query Syntax

The query language of a standard search engine is simply a list of keywords. In some search engines, each keyword can optionally be prepended by a plus sign ("+"). Keywords with a plus sign must appear in a satisfying document, whereas keywords without a plus sign may or may not appear in a satisfying document (but the appearance of such keywords is desirable). The query language of XSEarch is a simple exten- sion of the language described above. In addition to specifying keywords, we allow the user to specify labels and keyword-label combinations that must or may appear in a satisfying document.

E. Query Semantics

This section presents the semantics of our queries. In order to satisfy a query Q, each of the required terms in Q must be satisfied. In addition, the elements satisfying Q must be meaningfully related. However, it is difficult to determine when a set of elements is meaningfully related. Therefore, we assume that there is a given relationship R that determines when two nodes are related. We then show how to extend R for arbitrary sets of nodes. We also give one natural ex- ample of a relationship, which we call interconnection. In our working system we use the interconnection relationship. However, it is possible to use a different relationship, with little impact on system efficiency.

F. Query Syntax and Search Algorithm

Search is performed within a single index in order to retrieve relevant content. The search criteria are specified in a query XML document whose format is presented below. *Contexts* and *fields*2 are referred to in search queries. Figure 6 (below) illustrates the query structure. Queries have a recursive structure similar to the indexing schema. The *context* and *field* elements in queries refer to corresponding elements in the indexing schema by ID. The parent-child element relationship for both contexts and fields **should** follow the order in the indexing schema. The element content of query *context* elements consists of: - *field* element(s) – their element content is transformed to a Lucene query. The supported

full-text queries for a field include term, phrase, fuzzy, Boolean, span, wildcard, range queries. Search results are sorted by *tf-idf.- AND; OR | and;or* elements – recursive multi argument operations denoting *intersection* and *union* of search results returned for sets of *context* arguments (*AND; OR*) or *field* arguments (*and; or*). In either case, results are sorted by *tf-idf.*

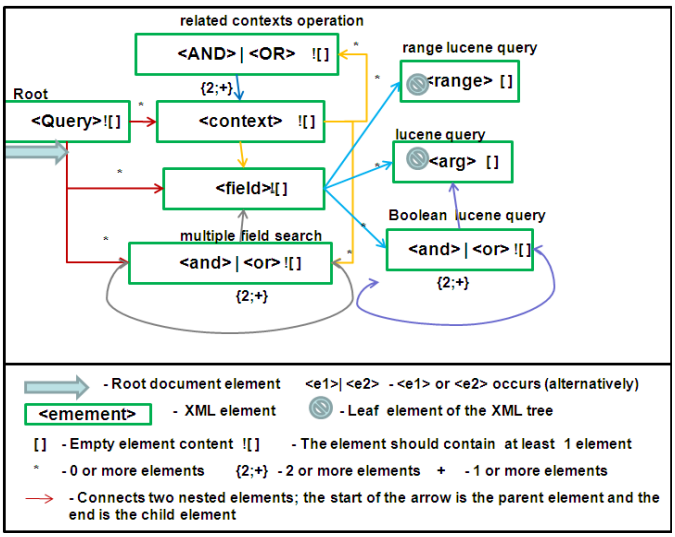

Figure6. Search Query Structure.

The search algorithm is presented below.

```
==========================================================================================
                                XML Index search
==========================================================================================
Iterate query root children
    1. For all children = <context> → call F1 : result map with key-value pairs <context ID, result list>
    2. If child= <and> | <or> | <field> → call F2 : result list
Return → result map + <global context, result list> entry
==========================================================================================
                                F1:Context search
==========================================================================================
Foreach context element:
    Iterate context children
            1. If child = <AND> | <OR> → call F3: context result list
            2. If child = <and> | <or> | <field> → call F2 → result list
            3. result = result + context result list + result list (sort by tf-idf)
Return → result
==========================================================================================
                                F2: Field Search
==========================================================================================
If current element = <and> | <or> → push operator in stack
    1. Iterate current element children
            1.1. For all <field> elements → lucene search : push result list in stack
            1.2. For all <and> | <or> elements → push operator in stack → call F2
    2. Process stack → pop the peak: if peak = result list → add it to temp list, else process operation <and> |
    <or> on temp list (sort by tf-idf) and push back the result {do while the stack is not empty}
    3. Return → temp
Else process lucene search → Return result
==========================================================================================
                                F3: Context operations search
==========================================================================================
If current element = <AND> | <OR> → push operator in stack and iterate current element children
            1.1. For all <context> elements from children → call F1 : push result list in stack
            1.2. For all <AND> | <OR> elements from children → push operator in stack → call F3
    2. Process stack → pop the peak: if peak = result list → add it to temp list, else process operation <AND> |
    <OR> on temp list (sort by tf-idf) and push back the result {do while the stack is not empty} →
    3. Return → temp
Else call F1 → Return result
// !NOTE: the leaf context elements should have only <and> | <or> | <field> child elements, else endless cycle
```

Figure7. Search Procedure.

5. EXPERIMENTAL RESULTS

We performed extensive experimentation with the XSEarch system, which was implemented in Java. The experiments were carried out on a Pentium 4, with a CPU of 1.6GHZ and 2GB of RAM, running the Windows XP operating system.The following shows the XML Query Results.

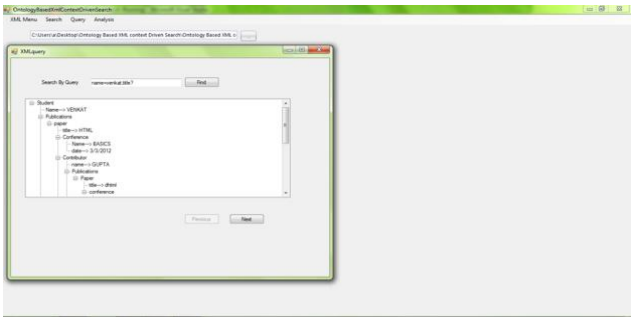

Figure 8 XML Query Result.

The following shows the XML Ontology Based Query Results.

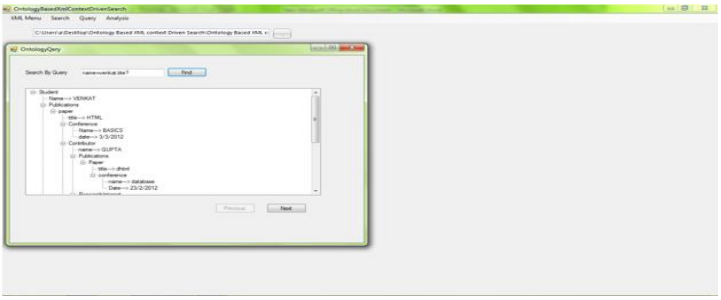

Figure 9 XML Ontology Query Result.

The following shows the comparison of XML Query(XCD) and XML Ontology Query for their time complexity.

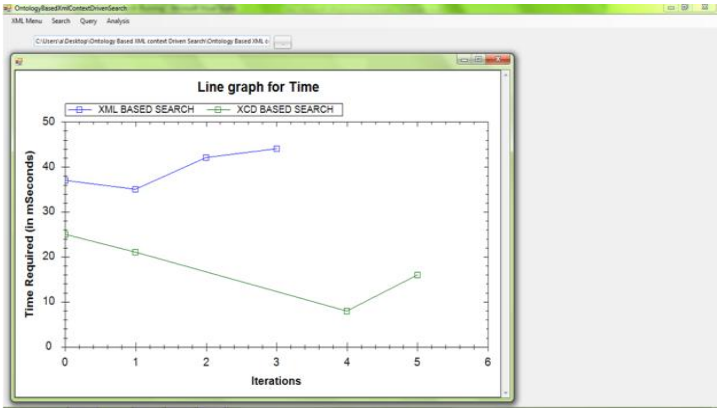

Figure 10 Comparison of time complexity.

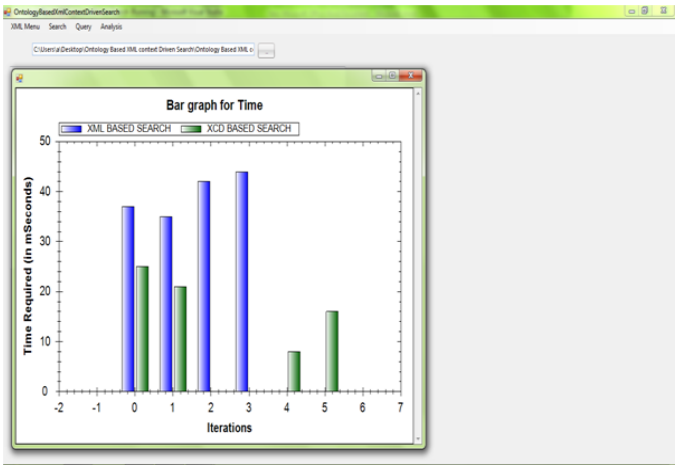

Figure 11 Comparison of time complexity.

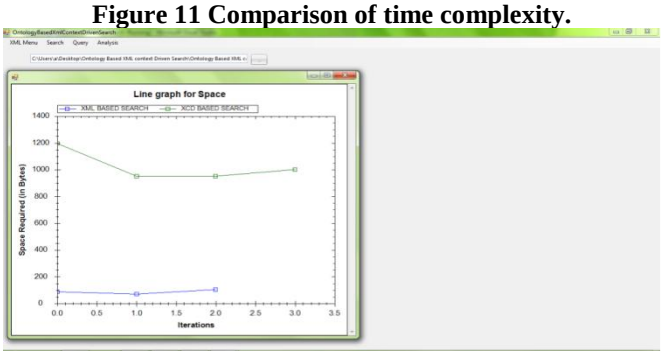

Figure 12 Comparison of space complexity.

REFERENCES

[1] Kamal Taha, Ramez Elmasri , "XCDSearch: An XML Context-Driven Search Engine," IEEE Transactions of Knowledge and Engineering, December 2010.

[2] S. Amer-Yahia, E. Curtmola, and A. Deutsch, "Flexible and Efficient XML Search with Complex Full-Text Predicates," Proc. ACM SIGMOD '06, 2006.

[3] D. Alorescu and I. Manolescu, "Integrating Keyword Search in XML Query Processing," Computer Networks, vol. 33, pp. 119-135, 2000.

[4] C. Agrawal and G. Das, "DBXplorer: A System for Keyword- Based Search over Relational Databases," Proc. Int'l Conf. Data Eng. (ICDE '02), 2002.

[5] B. Aditya and S. Sudarshan, "BANKS: Browsing and Keyword Searching in Relational Databases," Proc. Int'l Conf. Very Large Data Bases (VLDB '02), 2002.

[6] B. Balmin, V. Hristidis, and Y. Papakonstantinon, "Keyword Proximity Search on XML Graphs," Proc. Int'l Conf. Data Eng. (ICDE '03), 2003.

[7] B. Balmin and V. Hristidis, "ObjectRank: Authority-Based Keyword Search in Databases," Proc. Int'l Conf. Very Large Data Bases (VLDB '04), 2004.

[8] C. Botev, L. Guo, and F. Shao, "XRANK: Ranked Keyword Search over XML Documents," Proc. ACM SIGMOD '03, 2003.

[9] Books24_7, http://www.books24x7.com/books24x7.asp, 2010.

[10] S. Cohen, J. Mamou, and Y. Sagiv, "XSEarch: A Semantic Search Engine for XML," Proc. Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

[11] S. Cohen and Y. Kanza, "Interconnection Semantics for Keyword Search in XML," Proc. Int'l Conf. Information and Knowledge Management (CIKM '05), 2005.

[12] B. Balmin, V. Hristidis, and N. Koudas, "A System for Keyword Proximity Search on XML Databases," Proc. Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

6. CONCLUSION

Non-context-driven XML search engines build relationships among data nodes based solely on their labels and proximity to one another while overlooking their contexts (parents), which may cause these engines to return faulty answers. In this paper, we have introduced XCD Search, an XML context-driven search engine, which answers Keyword based and Loosely Structured queries. XCD Search accounts for nodes' contexts by considering each set consisting of a parent and its children data nodes in the XML tree as one entity (CT). We proposed mechanisms for determining the semantic relationships among different CTs. We also proposed an efficient stack-based sort-merge algorithm that selects from the set of CTs containing keywords (KCs) subsets, wherein each subset contains the smallest number of KCs that are closely related to one another and contain at least one occurrence of each keyword. We took as samples of non-context-driven XML search engines and compared them heuristically and experimentally with XCD Search. The experimental results show that XCD Search outperforms significantly the three other systems.