ISSN 2047-3338

XML Keyword Search: Coarseness Evaluation under Ambiguity Concerns for Effective Results

K. Sampth kumar[1], Lokeshwari[2] and J. Srikanth[3]
Department of CSE, Aurora Engineering College, Bhongir, Nalgonda, A.P., India

***Abstract*– An XML search engine XSearch that addresses an open drawback in XML keyword search: given relevant matches to keywords, the way to compose question results properly in order that they will be effectively ranked and simply digested by users. The approaches adopted within the literature generate either overwhelmingly giant results or fragmentary results, each of which can cause the ranking schemes to be ineffective. Intuitively, every question features a search target and every result ought to contain precisely one instance of the search target together with its proof. We have a tendency to developed XSearch that composes atomic and intact question results driven by users' search targets.**

***Index Terms*– Keyword Search, XML and XSearch**

I. INTRODUCTION

COMPARED with text search engines where the came back results are static documents, XML keyword search engines are able to give finer-grained question results than the complete XML documents as a result of the supply of the structure data, that provides opportunities to higher, satisfy the users' data wants. However, since everything is represented as a sub tree in an XML database, a way to determine the individual question result's granularity could be a new challenge that's unaddressed. As shown within the following example, applicable result composition could be a key to ranking.

Example 1.1: A user seeking the award data of a student named Ramesh during a.P would issue question Q1 in Table I, "A.P, Ramesh, Rakesh, and award". The schema of the XML tree is shown in Fig. 1. Note that our system will handle XML documents while not DTDs conjointly.

Ideally every question result ought to contain one instance of student, at the side of the connected keyword matches, like the 2 question results shown in Figure three. Besides, results ought to be properly ranked. for example, most of the ranking schemes can rank the coed that has 2 awards over the coed with one award, as this is often the sole distinction of those 2 question results.

Intuitively, every keyword search incorporates a goal that is typically the knowledge of a true world entity or relationship among entities. we tend to use the term search target to seek advice from the knowledge that the user is longing for, and target instance to denote every instance of the search target within the knowledge. Every fascinating question result got to have specifically one target instance at the aspect of all associated proof, thus ranking and top-k question processing are usually based not off target instances, and thus become meaningful. Specifically, question results of an XML keyword search ought to satisfy the subsequent 2 properties: Atomicity. A question result ought to be atomic: it ought to encompass one target instance. Within the on top of sample question, every result ought to correspond to a definite student.

Sample Key Word Search:

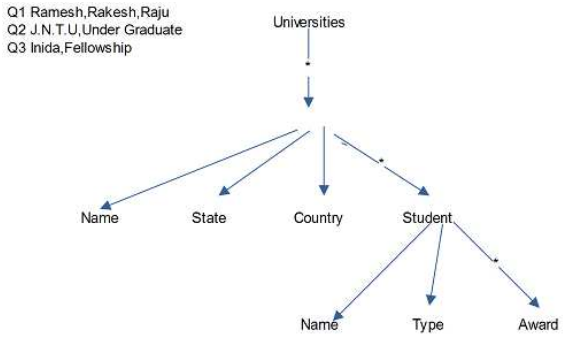


Fig. 1: Schema of an XML Document

Atomicity permits the ranking methodology to rank target instances and show the top-k most relevant ones to the user. Intactness: every question result ought to be intact: containing the entire target instance moreover as all its supporting data. Within the on top of sample question, all keyword matches associated with a similar student ought to be in one result. With intactness, a ranking methodology has the entire read of every target instance to provide a good ranking.

However, the question composition ways adopted in existing XML keyword search engines, named as Sub tree Result and Pattern Match respectively during this paper, fail to satisfy the atomicity and/or intactness properties. Sub tree Result defines a question result as a tree rooted at an LCA (Lowest common ancestor) node consisting of all relevant matches that are descendants of this LCA node and therefore the ways connecting them, as adopted. The results generated by Sub tree Result usually fail to be atomic.

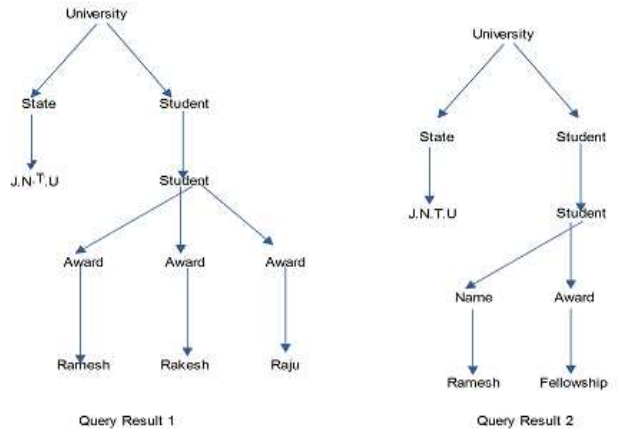


Fig. 2: Desired Query Results of Q1

Example 1.2: For Q1, a result made by Sub tree Result typically contains several target instances: the tree rooted at a university node that contains the match to A.P and every one the matches to Ramesh, Raeksh and award, like those shown in Fig. 2. As we are able to see, Sub tree Result violates atomicity. With several target instances (students) in a very single result, ranking isn't performed on course instances, and might be totally unreasonable. Suppose result one has additional matches to question keywords than result a pair of, then result one is probably going to be ranked higher by most existing ranking schemes. However, in result one there's no student named Ramesh, Rakesh and none of the scholar associated with Raemsh or Rakesh has any award. In result a pair of, the scholar that matches Ramesh is mixed with alternative students that solely match one keyword.

On the opposite hand, Pattern Match defines question |a question |a question} result as a tree rooted at an LCA node consisting of specifically one match to every query keyword that is meaningfully connected with one another and also the methods connecting them, used. The results generated by Pattern Match Result typically fail to be intact.

Example 1.3: the highest three results of Q1 generated by Pattern Match are shown in Fig. 2. Though every result's atomic, it's not intact: constant target instance (student) named as Ramesh with 2 awards is presented as 2 results, one for every match of award. This causes many issues. First, the highest k results typically contain data regarding but k target instance, since multiple results will describe constant target instance. This not solely wastes the user's time however makes it troublesome for a user to seek out the highest k ranked target instances. Second, from such results the user loses data, e.g., the scholar who has DHS student award and J.N.T.U award is truly constant person. Furthermore, separating the supporting data (award) of constant target instance (student) into multiple results can divide the ranking signals among these results. As an example, a student named Ramesh with 2 awards ought to intuitively be ranked above another Ramesh with one award; however Pattern Match invalidates this ranking issue by separating this student into 2 results. During this demo, we'll gift a XML keyword engine, XSearch that addresses the on top of challenges of result composition and allows effective ranking. Compared with

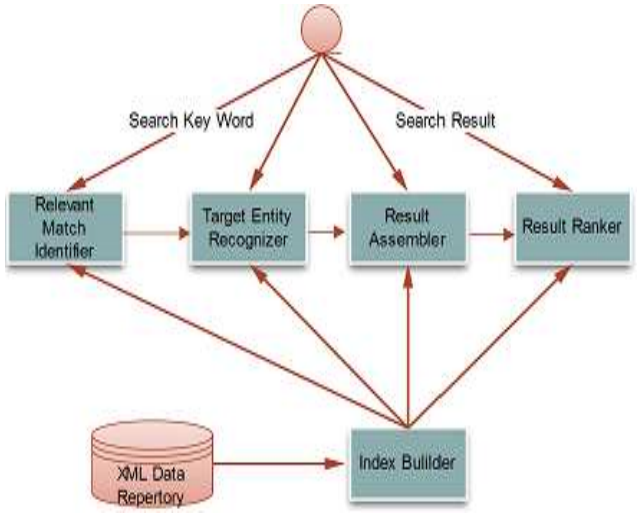


Fig. 3: Architecture of XSearch

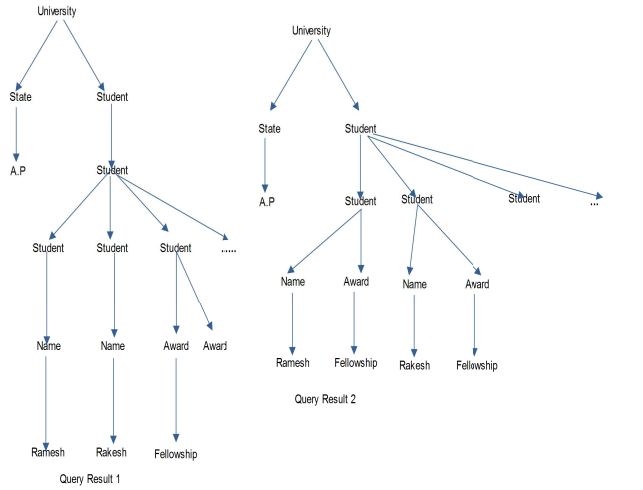
existing keyword search engines, XSearch allows effective ranking primarily based on search targets. Specifically, the technical con tributions of this work include: (1) To the most effective of our information, XSearch is that the initial system that composes ranking friendly XML keyword search results, that are driven by inferred user search targets. (2) XSearch identifies user search targets by inferring the comeback specification in question keywords, modifying relationship among keyword matches and also the information entities concerned within the search. (3) XSearch adopts a unique question result composition approach towards achieving each atomicity and intactness properties.

II. SYSTEM OVERVIEW

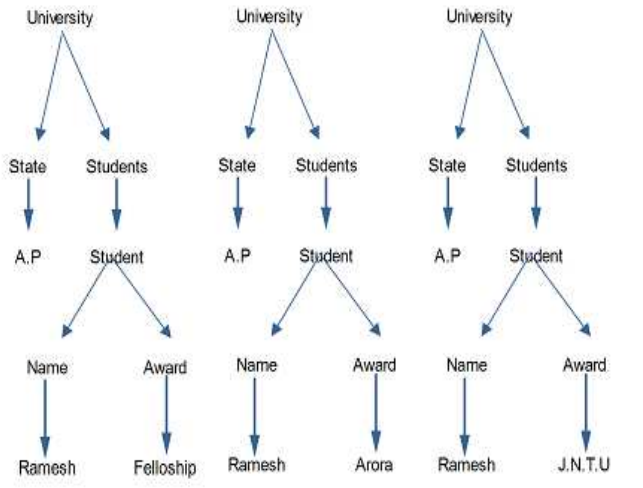
Fig. 3 shows the architecture of XSearch. Users input a keyword question similarly as an optional specification of the search target. First, the Relevant Match Identifier retrieves nodes within the XML document that match keywords, and identifies relevant matches. Target Entity Recognizer classifies XML nodes into entities, attributes and affiliation nodes, and classifies keywords into come back nodes and predicates. If the user doesn't explicitly specify the search target (which is probably going the case as most users are unwilling to perform advanced searches), Target Entity Recognizer infers the search target (if not specified by the user) based mostly on node classes and therefore the modifying relationships of entities and predicates. Result Assembler module composes and organizes the results based mostly on the search target and relevant keyword matches. Result Ranker ranks the results based mostly on their sizes and numbers of keyword matches that are common ranking factors adopted in search engines of these modules use the indexes of the input XML information engineered by the Index builder module. Next we have a tendency to briefly introduce the key modules of the system.

Index Builder: The Index Builder builds 3 indexes to hurry up question processing:

- A node inverted index is constructed to seek out the nodes matching every keyword.
- A node class index is constructed to retrieve the class of a node using node ID. we have a tendency to adopt the approach proposed in X obtain to classify XML nodes into 3 categories:
- entity, if a node may be a *-node within the DTD;
- Attribute, if a node isn't an entity and has just one leaf kid. Its leaf kid is named the attribute value;
- Connection node, if a node is neither an entity nor an attribute.
- A modifier index that maps every attribute worth to an inventory of entity sorts. An entity sort E is within the list of attribute worth A, if A not a modifier of E (to be outlined within the Target Entity Recognizer part).



(a) Sub Tree



(b) Pattern Matches

Fig. 4: Query Results of Q1 Returned by Sub tree Result and Pattern Match

All 3 indexes are engineered offline. The modifier index may be efficiently engineered by a traversal of the XML knowledge.

Target Entity Recognizer: this is often the key module of XSearch that infers search target for a question. It inters the search target by analyzing the matches to input keywords and also the XML knowledge structure. Two situations are considered:

CASE 1: In several queries, users offer hints regarding the XML nodes they're probing for moreover because the conditions these nodes ought to satisfy. we tend to decision these XML nodes come back nodes and also the conditions search predicates. The entities related to come back nodes are thought-about as target entities. Intuitively, if an entity is laid out in a question while not data regarding its associated attributes, then seemingly the data of its instances is that the user's interest and this entity is taken into account as a comeback node. If a affiliation node or attribute node is laid out in a question, however none of their worth descendants matches any keyword, then most likely the instances of those nodes in conjunction with the values are what the user is sorting out. During this case, the entity related to this attribute (which is taken into account because the nearest ancestor entity of the attribute), or the closest descendant entity of the affiliation node, is taken into account because the target entity for example, for Q1, award would be thought-about as a come back node, and so student because the target entity.

CASE 2: In case the question keywords don't contain come back nodes, we tend to exam all the relevant entities and also the modifying relationship between search predicates and these entities to spot target entities. We tend to follow 2 inferences to exclude some entities from being target entities. First, if an attribute worth A seems within the question is and invariably associated with an entity sort E, then E isn't seemingly to be the target entity, we tend to decision A the non-modifier of E. Otherwise, A may be a modifier of E, or A modifies E. An entity sort may be a candidate target entity if all predicates within the question modify this entity sort.

Example 2.1: think about Q2, "A.P, undergraduate", where each keyword is search predicates. As there are not any come back nodes, we discover all entities concerned during this query: university and student. Allow us to choose 2 candidate semantics of this search that think about totally different relevant entities because the target entity:

(1) Notice the colleges in 2006 that

(i) Find during a.P

(ii) Have an undergraduate student.

(2) Notice the scholars who

(i) Are undergraduate students and

(ii) Attend a university during a.P.

If we tend to take better look, semantics (1) is counter intuitive: it specifies 2 conditions for search Target University. However, each university has undergraduate students. The second condition doesn't modify (or restrict/constrain) the university entity in the least, and is unlikely to be employed by an inexpensive user for looking out universities. Therefore, the

target entity of this question ought to be student that is changed by each A.P and undergraduate.

The second inference is to look at the modification power of the modifiers, in case multiple entity sorts are left when the primary inference. The key attribute of an entity ought to have the strongest modification power, whose presence shadows/ disables all alternative modifiers.

Example 2.2: think about Q3" Fellowship" as an example. This question has 2 candidate semantics:

(1) Notice the university named J.N.T.U that has one or a lot of students who have received Fellowship.

(2) Notice the scholar who attend J.N.T.U and who have received Fellowship. Within the knowledge, J.N.T.U, that is that the worth of the key attribute of university uniquely identifies a university. If the user's search target may be a university, s/he doesn't want further keywords like Fellowship. Therefore, the second semantics a lot of seemingly reflects what the user really means that. As we are able to see, the presence of J.N.T.U disables Fellowship as a modifier of university, so university isn't thought-about as a target entity.

III. RESULT ASSEMBLER

The Result Assembler module composes atomic and intact search results primarily based on the target entities inferred by the Target Entity Recognizer module. If there's only 1 target entity for the question, then one question results generated for every instance of the target entity to create certain that it's atomic and intact. We tend to additionally embody into every result the matches to every keyword that are closest (compare to alternative matches for constant keyword) to the current target entity instance. When a question has multiple target entities, the user is probably going inquisitive about all target entities and their relationships. We tend to adopt sub tree end in this case, such that every result contains all the connected target entity instances. We have got performed a collection of experiments on real knowledge sets to verify the result quality, potency and scalability of XSearch.

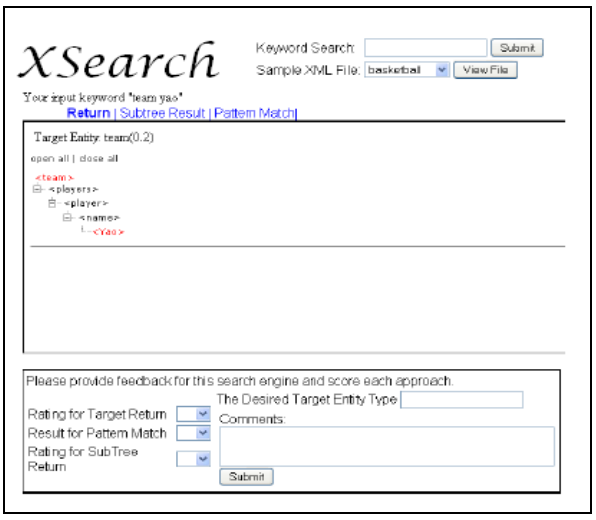


Fig. 5: Snap Shot of the XSearch

IV. CONCLUSION

Through this demo, we have a tendency to aim at showing users a crucial nonetheless unstudied step of processing keyword searches on XML: composing leads to a meaningful manner. As mentioned in Section I, result composition has crucial effects on meaningful result ranking. The event and demonstration of Target- Search shows the importance of this step and offers a sound answer to the challenges of composing XML search results. We have a tendency to conduct experiments on many sample knowledge sets, together with a knowledge regarding baseball groups and players, a geographical knowledge (mondial), and a bookstore knowledge with recursive schema. The input will be an easy keyword question, like "Arora, undergraduate". Users may also specify the search targets using "*", e.g., "JNTU, undergraduate, student*".

To method the question, Xsearch adopts existing XML keyword search techniques [8] to spot relevant keyword matches, then automatically identifies the search target (if it's not specified with the question) and composes query results in keeping with the techniques mentioned in Section II. The results are presented as XML fragments, within which components will be expanded/collapsed. If the user isn't glad with the results, s/he will specify the fascinating target entity sort, based mostly on that Xsearch can generate new results.

Through the results generated by Xsearch, the user ought to be ready to simply notice the specified info within the top- k results, as they correspond to precisely the top-k target entity instances, i.e., they're atomic and intact. By being atomic and intact, every target entity instance will be fairly ranked, while not considering an excessive amount of or too very little info. Besides, the results also are straightforward for users to know and can not contain irrelevant info of a target entity instance in a very result, or split the data of a target entity instance in multiple results. Note that a way to rank results of XML keyword search is an orthogonal problem; any ranking theme will be incorporated into Xsearch.

For comparison purpose, the results made by Subtree Result and Pattern Match also will be shown upon clicking the corresponding tabs, and ranked using identical ranking theme. Through comparison, the users can perceive the disadvantage of failing to supply atomic or intact results, as illustrated in Examples 1.1 and 1.2. This helps users notice the importance of result composition in XML keyword search and therefore the advantages a decent methodology of result composition brings to results ranking and user search expertise.

Users will rate the results generated by every approach within the scope of 1-10, and supply feedbacks to the developers of Xsearch. Through the demonstration of Xsearch, we have a tendency to show the importance of fastidiously composing results for XML keyword search, a drag so far largely ignored. The publicity of Xsearch within the database community can attract a lot of analysis on this subject, which can create the XML search engines a lot of intelligent and so more profit the users.

REFERENCES

[1] T. Cheng and K. C.-C. Chang. Entity Search Engine: Towards Agile,Best-Effort Information Integration over the Web. In CIDR, 2007.
[2] T. Cheng, X. Yan, and K. C.-C. Chang. EntityRank: Searching Entities Directly and Holistically. In VLDB, 2007.
[3] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In VLDB, 2003.
[4] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword Proximity Search in XML Trees. IEEE Transactions on Knowledge and Data Engineering, 18(4), 2006.
[5] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In VLDB, 2004.
[6] Z. Liu, Y. Cai, and Y. Chen. Ranking Friendly Result Composition for XML Keyword Search. Technical Report TR-09-016, A.P State University, 2008.
[7] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML,Keyword Search. In SIGMOD, 2007.
[8] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In VLDB, 2008.
[9] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In SIGMOD, 2005.

K. Sampath kumar is pursuing his M. Tech in Software Engineering (Dept. of CSE) in Aurora Engineering College, Bhongir, Nalgonda, A.P and India. His areas of interests are Software Engineering, Testing, Mining and Data Engineering, Email: sampath.kilaru@gmail.com

Mrs. Lokeshwari senior Associate Professor in the Department of Computer Science & Engineering, Aurora Engineering College, Bhongir, Nalgonda, A.P and India, Email: lokeshwari@yahoo.co.in

Srikanth Jatla working as Associate Professor and Head of the Department of Computer Science and Engineering at Aurora's Engineering College with a teaching experience of 12years.He is a B.E and M. Tech in Computer Science and pursuing his PhD in Data Stream Mining at JNTU, Hyderabad. His areas of interest includes data structures, principles of programming languages, algorithm analysis and compiler design, Email: jsrikanth@aurora.ac.in