TargetSearch: A Ranking Friendly XML Keyword Search Engine

Ziyang Liu, Yichuan Cai, Yi Chen

School of Computing, Informatics, and Decision Systems Engineering, Arizona State University

{ziyang.liu,yichuan.cai,yi}@asu.edu

***Abstract*— This demo illustrates an XML search engine TargetSearch that addresses an open problem in XML keyword search: given relevant matches to keywords, how to compose query results properly so that they can be effectively ranked and easily digested by users. The approaches adopted in the literature generate either overwhelmingly large results or fragmentary results, both of which may cause the ranking schemes to be ineffective. Intuitively, each query has a search target and each result should contain exactly one instance of the search target along with its evidence. We developed TargetSearch which composes atomic and intact query results driven by users' search targets.**

I. Introduction

Compared with text search engines where the returned results are static documents, XML keyword search engines are able to provide finer-grained query results than the whole XML documents due to the availability of the structure information, which provides opportunities to better satisfy the users' information needs. However, since everything is represented as a subtree in an XML database, how to identify the individual query result's granularity is a new challenge that is unaddressed. As shown in the following example, appropriate result composition is a key to ranking.

Example 1.1: A user seeking the *award* information of a student named *Mike Smith* in *Arizona* would issue query Q_1 in Table I, "*Arizona, Mike, Smith, award*". The schema of the XML tree is shown in Figure 1. Note that our system can handle XML documents without DTDs also.

Ideally each query result should contain one instance of *student*, along with the related keyword matches, such as the two query results shown in Figure 3. Besides, results should be properly ranked. For instance, most of the ranking schemes, such as [3], will rank the student that has two awards higher than the student with one award, as this is the only difference of these two query results. ■

Intuitively, each keyword search has a goal, which is usually the information of a real world entity or relationship among entities, as observed in [1], [2]. We use the term *search target* to refer to the information that the user is looking for, and *target instance* to denote each instance of the search target in the data. Each desirable query result should have *exactly one target instance* along with all associated evidence, so that ranking and top-k query processing can be based on target instances, and thus become meaningful. Specifically, query

TABLE I
SAMPLE KEYWORD SEARCHES

Q_1	Arizona, Mike, Smith, award
Q_2	Arizona, undergraduate
Q_3	ASU, SFAz Fellowship

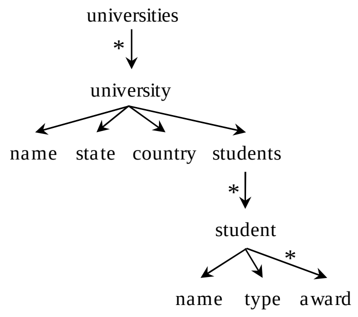


Fig. 1. Schema of an XML Document

results of an XML keyword search should satisfy the following two properties:

Atomicity. A query result should be *atomic*: it should consist of a single target instance. In the above sample query, each result should correspond to a distinct student. Atomicity enables the ranking method to rank target instances and show the top-k most relevant ones to the user.

Intactness. Each query result should be *intact*: containing the whole target instance as well as all its supporting information. In the above sample query, all keyword matches related to the same student should be in one result. With intactness, a ranking method has the whole view of each target instance to give a fair ranking.

However, the query composition methods adopted in existing XML keyword search engines, named as *Subtree Result* and *Pattern Match* respectively in this paper, fail to satisfy the atomicity and/or intactness properties. Subtree Result defines a query 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 the paths connecting them, as adopted in [7], [8], [9]. The results generated by Subtree Result generally fail to be atomic.

Example 1.2: For Q_1, a result produced by Subtree Result generally contains many target instances: the tree rooted at a *university* node that contains the match to *Arizona* and *all* the matches to *Mike*, *Smith* and *award*, such as the ones

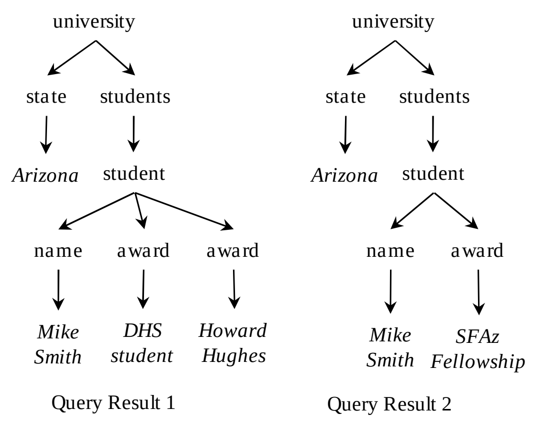


Fig. 3. Desirable Query Results of Q_1

shown in Figure 2(a). As we can see, Subtree Result violates atomicity. With many target instances (students) in a single result, ranking is not performed on target instances, and can be totally unreasonable. Suppose result 1 has more matches to query keywords than result 2, then result 1 is likely to be ranked higher by most existing ranking schemes. However, in result 1 there is no student named *Mike Smith*, and none of the student related to *Mike* or *Smith* has any *award*. In result 2, the student that matches Mike Smith is mixed with other students that only match one keyword. ■

On the other hand, *Pattern Match* defines a query result as a tree rooted at an LCA node consisting of *exactly one match to each query keyword* which are meaningfully related with each other and the paths connecting them, used in [3], [4], [5]. The results generated by Pattern Match Result generally fail to be intact.

Example 1.3: The top 3 results of Q_1 generated by Pattern Match are shown in Figure 2(b). Although each result is atomic, it is not intact: the same target instance (*student*) named as *Mike Smith* with two awards is presented as two results, one for each match of *award*.

This causes several problems. First, the top k results generally contain information about less than k target instance, since multiple results can describe the same target instance. This not only wastes the user's time but makes it difficult for a user to find the top k ranked target instances. Second, from such results the user loses information, e.g., the student who has *DHS student award* and *Howard Hughes award* is actually the same person. Furthermore, separating the supporting information (*award*) of the same target instance (*student*) into multiple results will divide the ranking signals among these results. For example, a student named *Mike Smith* with two *award*s should intuitively be ranked higher than another *Mike Smith* with one *award*, but Pattern Match invalidates this ranking factor by separating this student into two results. ■

In this demo, we will present a XML keyword engine, *TargetSearch*, which addresses the above challenges of result composition and enables effective ranking. Compared with existing keyword search engines, TargetSearch enables effective ranking based on search targets. Specifically, the technical contributions of this work include: (1) To the best of our knowledge, TargetSearch is the first system that composes ranking-friendly XML keyword search results, which are driven by inferred user search targets. (2) TargetSearch identifies user search targets by inferring the return specification in query keywords, modifying relationship among keyword matches and the data entities involved in the search. (3) TargetSearch adopts a novel query result composition approach towards achieving both atomicity and intactness properties.

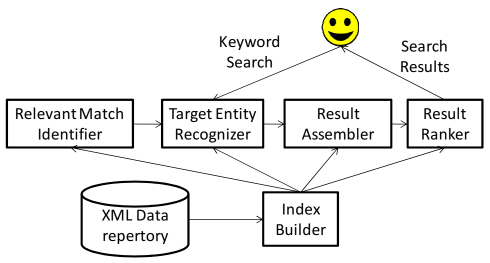


Fig. 4. Architecture of TargetSearch

II. System Overview

Figure 4 shows the architecture of TargetSearch. Users input a keyword query as well as an optional specification of the search target. First, the *Relevant Match Identifier* retrieves nodes in the XML document that match keywords, and identifies relevant matches using the approach proposed in [8]. *Target Entity Recognizer* classifies XML nodes into entities, attributes and connection nodes, and classifies keywords into return nodes and predicates. If the user does not explicitly specify the search target (which is likely 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 on node categories and the *modifying relationships* of entities and predicates. *Result Assembler* module composes and organizes the results based on the search target and relevant keyword matches. *Result Ranker* ranks the results based on their sizes and numbers of keyword matches, which are common ranking factors adopted in search engines such as [3]. All these modules use the indexes of the input XML data built by the *Index builder* module. Next we briefly introduce the key modules of the system.

Index Builder. The Index Builder builds three indexes to speed up query processing:

- A *node inverted index* is built to find the nodes matching each keyword.
- A *node category index* is built to retrieve the category of a node using node ID. We adopt the approach proposed in XSeek [7] to classify XML nodes into three categories: (1) entity, if a node is a *-node in the DTD; (2) attribute, if a node is not an entity and has only one leaf child. Its leaf child is called the attribute value; (3) connection node, if a node is neither an entity nor an attribute.
- A *modifier index*, which maps each attribute value to a list of entity types. An entity type E is in the list of

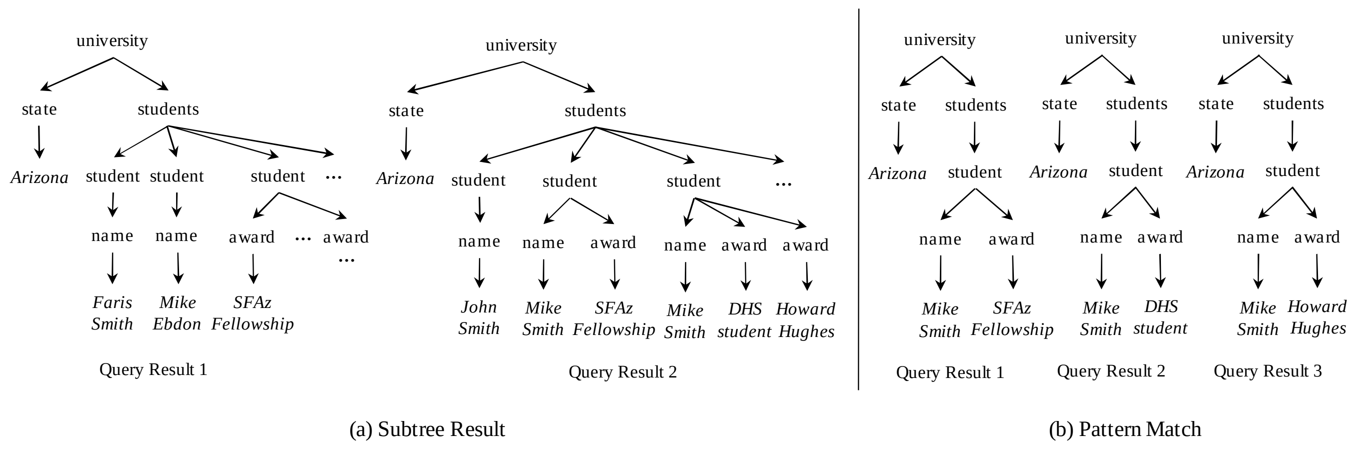


Fig. 2. Query Results of Q_1 Returned by Subtree Result and Pattern Match

attribute value A, if A *not* a *modifier* of E (to be defined in the *Target Entity Recognizer* part).

All three indexes are built offline. The modifier index can be efficiently built by a traversal of the XML data, the details of which can be found in [6].

Target Entity Recognizer. This is the key module of TargetSearch, which infers search target for a query. It inters the search target by analyzing the matches to input keywords and the XML data structure. Two scenarios are considered:

➢ **CASE 1.** In many queries, users provide hints about the XML nodes they are looking for as well as the conditions these nodes should satisfy. We call these XML nodes *return nodes* and the conditions *search predicates*. The entities associated with return nodes are considered as target entities.

Intuitively, if an entity is specified in a query without information about its associated attributes, then likely the information of its instances is the user's interest and this entity is considered as a return node. If a connection node or attribute node is specified in a query, but none of their value descendants matches any keyword, then probably the instances of these nodes along with the values are what the user is searching for. In this case, the entity associated with this attribute (which is considered as the nearest ancestor entity of the attribute), or the nearest descendant entity of the connection node, is considered as the target entity. For instance, for Q_1, *award* would be considered as a return node, and thus *student* as the target entity.

➢ **CASE 2.** In case the query keywords do not contain return nodes, we exam all the relevant entities and the modifying relationship between search predicates and these entities to identify target entities.

We follow two inferences to exclude some entities from being target entities. First, if an attribute value A appears in the query is and always related to an entity type E, then E is not likely to be the target entity, we call A the *non-modifier* of E. Otherwise, A is a *modifier* of E, or A modifies E. An entity type is a candidate target entity if all predicates in the query modify this entity type.

Example 2.1: Consider Q_2, "*Arizona, undergraduate*", where both keywords are search predicates. As there is no return nodes, we find all entities involved in this query: *university* and *student*. Let us judge two candidate semantics of this search that consider different relevant entities as the target entity: (1) Find the universities in 2006 that (i) locate in Arizona (ii) have an undergraduate student. (2) Find the students who (i) are undergraduate students and (ii) attend a university in Arizona. If we take a closer look, semantics (1) is counter intuitive: it specifies two conditions for search target university. However, every university has undergraduate students. The second condition does not modify (or restrict/constrain) the university entity at all, and is unlikely to be used by a reasonable user for searching universities. Therefore, the target entity of this query should be *student*, which is modified by both *Arizona* and *undergraduate*. ■

The second inference is to examine the *modification power* of the modifiers, in case multiple entity types are left after the first inference. The key attribute of an entity should have the strongest modification power, whose presence shadows/disables all other modifiers.

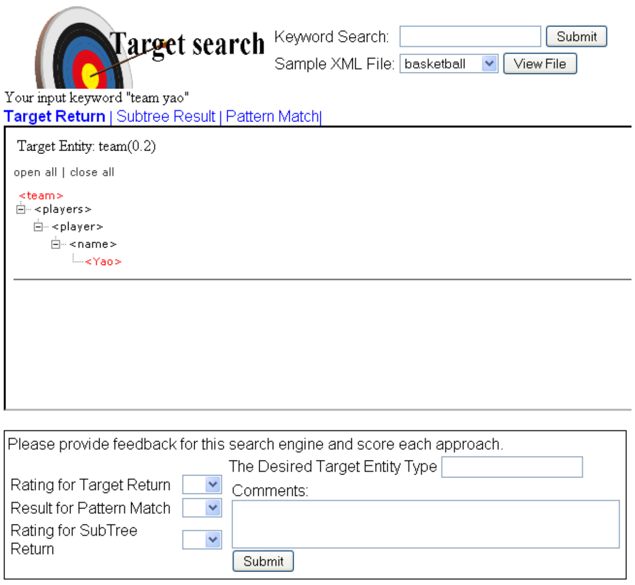
Example 2.2: Consider Q_3 *"ASU, SFAz Fellowship"* as an example. This query has two candidate semantics: (1) Find the university named *ASU* that has one or more students who have received *SFAz Fellowship*. (2) Find the student who attend *ASU* and who have received *SFAz Fellowship*. In the data, *ASU*, which is the value of the key attribute of *university*, uniquely identifies a university. If the user's search target is a university, s/he does not need additional keywords like *SFAz Fellowship*. Therefore, the second semantics more likely reflects what the user actually means. As we can see, the presence of *ASU* disables *SFAz Fellowship* as a modifier of *university*, thus *university* is not considered as a target entity. ■

Result Assembler. The *Result Assembler* module composes atomic and intact search results based on the target entities inferred by the *Target Entity Recognizer* module.

If there is only one target entity for the query, then one query result is generated for each instance of the target entity to make sure that it is atomic and intact. We also include into each result the matches to each keyword that are *closest* (compare to other matches for the same keyword) to this target entity instance.

When a query has multiple target entities, the user is likely interested in all target entities and their relationships. We adopt subtree result in this case, such that each result contains all the related target entity instances.

We have performed a set of experiments on real data sets to verify the result quality, efficiency and scalability of TargetSearch, the details of which can be found in [6].



Sample XML File: basketball

Your input keyword "team yao"

Target Entity: team(0.2)

```
<team>
<players>
  <player>
    <name>
      <Yao>
```

Please provide feedback for this search engine and score each approach.

Fig. 5. Snapshot of Target Search

III. Demonstration

Through this demo, we aim at showing users an important yet unstudied step of processing keyword searches on XML: composing results in a meaningful way. As discussed in Section I, result composition has crucial effects on meaningful result ranking. The development and demonstration of TargetSearch shows the importance of this step and gives a sound solution to the challenges of composing XML search results.

The demo of TargetSearch is available online at http://wsdb.asu.edu/target and a snapshot is shown in Figure 5. We provide several sample data sets, including a data about baseball teams and players, a geographical data (mondial), and a bookstore data with recursive schema. The input can be a simple keyword query, such as "*Arizona, undergraduate*". Users can also specify the search targets using "*", e.g., "*Arizona, undergraduate, student**".

To process the query, TargetSearch adopts existing XML keyword search techniques [8] to identify relevant keyword matches, then automatically identifies the search target (if it is not specified with the query) and composes query results according to the techniques discussed in Section II. The results are presented as XML fragments, in which elements can be expanded/collapsed. If the user is not satisfied with the results, s/he can specify the desirable target entity type, based on which TargetSearch will generate new results.

Through the results generated by TargetSearch, the user should be able to easily find the desired information in the top-k results, as they correspond to exactly the top-k target entity instances, i.e., they are atomic and intact. By being atomic and intact, each target entity instance can be fairly ranked, without considering too much or too little information. Besides, the results are also easy for users to understand and will not contain irrelevant information of a target entity instance in a result, or split the information of a target entity instance in multiple results. Note that how to rank results of XML keyword search is an orthogonal problem; any ranking scheme can be incorporated into TargetSearch.

For comparison purpose, the results produced by *Subtree Result* and *Pattern Match* will also be shown upon clicking the corresponding tabs, and ranked using the same ranking scheme. Through comparison, the users will understand the disadvantage of failing to produce atomic or intact results, as illustrated in Examples 1.1 and 1.2. This helps users realize the importance of result composition in XML keyword search and the benefits a good method of result composition brings to results ranking and user search experience. Users can rate the results generated by each approach in the scope of 1-10, and provide feedbacks to the developers of TargetSearch.

Through the demonstration of TargetSearch, we show the importance of carefully composing results for XML keyword search, a problem to date largely ignored. The publicity of TargetSearch in the database community will attract more research on this topic, which will make the XML search engines more intelligent and thus further benefit the users.

IV. Acknowledgement

This material is based on work partially supported by NSF CAREER award IIS-0845647, IIS-0740129, and IIS-0915438.

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, Arizona 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.