Towards Adaptive Information Merging Using Selected XML Fragments

Ho-Lam Lau and Wilfred Ng

Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology, Hong Kong
{lauhl, wilfred}@cse.ust.hk

Abstract. As XML information proliferates on the Web, searching XML information via a search engine is crucial to the experience of both casual and experienced Web users. The returned XML fragments in the list is not directly usable, if not confusing, to the users, since in most cases the XML fragments extracted from a large XML repository are incomplete, scattered and redundant. Thus, it is necessary to re-iterate the searching process based on user preferences in order to obtain more complete, detailed and usable results. In this paper, we propose a unifying framework which takes searching, merging and user preferences into account. We view search queries and fragment labeling as an input in an on-going searching process, in which the relevant XML fragments are merged into a concise form and returned to the user a ranked result list.

1 Introduction

As the amount and use of XML data continue to grow, searching and ranking XML data has been an important issue studied in both database and information retrieval communities [1–4, 6, 7, 10]. Following the usual practice of handling results in Web search engines, the search results of these proposals are usually presented as a ranked list of small XML fragments to the users [1, 3, 11]. In practice, users do not have the schema knowledge of the underlying XML sources or have very little information of the data sources, therefore, highly structured XML queries such as XQuery FT expressions for searching are not easy for them to formulate. In addition, we recognize that the usual approach adopted by web search engines, which return a once-off list of items as the answers for a search query, is not adequate in XML setting. There are three reasons for the inadequacy. First, the target information may be scattered on the ranked list and thus it is not directly useful for the users. Second, the XML fragments can be duplicated in different ways. Third, a once-off query may not contain all desired information. In this paper, we propose a unifying framework which takes searching, merging and user preference into account.

Figure 1 shows the conceptual overview of our proposed framework. First, the user submits a query to the system and the system returns a list of fragments to the user. Then, the user selects the preferred fragments as feedback, the feedback will be merged and contribute as new search query which enlarge the

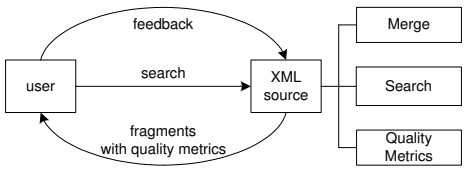

Fig. 1. A conceptual overview of the proposed framework

set of candidate fragments for the next iteration. Finally, the merged fragment and the new search results are returned to the user with our previously proposed notions of Quality Metrics (QMs) [9] which help users to judge the quality of fragments. We do not repeat the details in [9] here but mention that the QMs proposed are simple but effective metrics to assess the quality of individual data source or a combination of data sources, and are natural metrics to measure different dimension of the structure, data and subtrees. We contribute two main ideas related to searching XML information.

Unifying Framework We propose a unifying framework that searches and merges XML fragments in a ranked result list. The search is based on a fragment, which is viewed as a set of path-key pairs.

Adaptive Merging We propose an adaptive merging approach and four directional searching techniques, that are able to support progressive merging the search results according to the users' continuing feedback. With the combination of searching and merging, we provide flexibility on merging that match different users' preferences.

Paper Organization. Section 2 presents an overview of the unified framework for searching and merging techniques. Section 3 illustrates the merging techniques and introduces the merging approach for adopting the user feedback. We conclude the framework and discuss future work in Section 4.

2 The Unifying Framework for Searching and Merging

In this section, we present an overview of our unifying framework for searching and merging. A *path-key pair* is an ordered pair (p, k), where p is a path from the root to the parent node of the keyword, k. Thus, a path-key pair can be viewed as a simplified form of XPath [5]. An XML fragment is a sequence of non-repeated path-key pairs.

Figure 2 depicts the basic ideas of our framework which is able to incorporate the user preference and to support iterative searching and merging. Initially, the user submits a query to the search engine. We view the sources as the underlying XML database which collects XML fragments in a repository. Due to the space limitation, we do not describe the implementation details and the searching and ranking mechanism of the databases. However, we remark that our approach of searching and ranking techniques of XML fragments are similar to the recent work in [1, 3, 4, 7, 10].

The search engine returns the list of ranked XML fragments as the raw list. The raw list is decomposed into "candidate path-key pairs" sorted by the frequency in the raw list. The top k path-key pairs is then displayed to the user (By

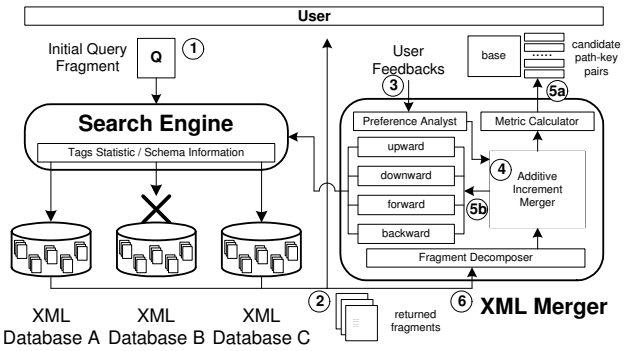

Fig. 2. Overview of searching and merging XML information via key-tags

default, $k = 10$). Initially, we categorize all the path-key pairs as "unclassified". The user feedback can be collected when he/she selects the preferred path-key pairs from the "candidate path-key pairs", which is similar to collecting the clickthrough data in the case of HTML data [8]. However, the main difference between searching HTML data in the mentioned work and searching XML data in our approach is that an XML fragment returned can be further used as a sample for re-querying. The user feedback is collected by the "Preference Analyst" and is re-classified into two categories as follows: **preferred**, and **unclassified**.

After the (re-classification) process, the two categories of path-key pairs are passed to the AIM. The preferred path-key pairs from the user contribute the merging process in twofold. First, the AIM establishes the "result fragment" by merging the "preferred" path-key pairs. The result fragment is then returned to the user. Second, they are served as new queries (i.e. re-queries) that are sent to the four searchers of *Upward*, *Downward*, *Forward* and *Backward*. The search results of the "re-queries" will be decomposed, added into the "candidate path-key pairs" and then displayed to the user in the next iteration. More details about AIM and Directional Searching will also be given in Section 3.

3 The Merging and Searching Approaches

In this section, we explain our approach, *the Adaptive Increment Merging (AIM) approach*, which supports further decomposing the selected fragments from users into path-key pairs. We also discuss the four directional searchings which are able to enrich the set of candidate path key pairs in the re-querying process.

3.1 The Adaptive Increment Merging Approach

The inputs of AIM are two lists: the "preferred" and "unclassified" path-key pairs and the outputs are the "result fragment", R, and a list of reordered candidate path-key pairs, C. R is an XML fragment resulting from merging the path-key pairs in the "preferred" list, which is possible to grow during the merging process. C is a list of path-key pairs displayed to the users for user feedback in each iteration. The path-key pairs in C are grouped according to their data sources and are sorted according to the their frequency among the list of fragments returned by the search engine. The AIM approach is shown in Algorithm 1.

Algorithm 1: Adaptive Increment Merging Approach

Input: R – the result fragment; P – a set of positive path-key pairs; U – a set of unclassified path-key pairs; ϱ – a quota variable; $S[\,]$ – an array of sets which represent the sources; $\varpi[\,]$ – an array of weights for the sources; // e.g. $\varpi[j]$ is the weight of source $S[j]$

Output: R – the result fragment; C – a list of candidate path-key pairs display for next iteration;

```
1  for each source S[j] do
2  |  S[j] = ∅ ;
3  |  Mark S[j] as negative source;
4  end
5  C = ∅;
6  for each path-key pairs p_i ∈ P do
7  |  R = R ∪ p_i;
8  |  if p_i is originated from source S[j] then
9  |  |  S[j] ∪ p_i;
10 |  |  Mark S[j] as positive source;
11 |  end
12 end
13 for each negative source S[j] do
14 |  ϖ[j] = ϖ[j]/2;
15 end
16 for each positive source S[j] do
17 |  ϖ[j] = (1 − Σ weights of negative sources) / (number of positive sources);
18 |  C = C∪ top-(ϱ × ϖ[j]) path-key pairs in S[j];
19 end
20 Perform Directional Searching using (P, U);
21 Return R and C;
```

Consider the following example, given a query, Q = (//author : Mary, //title : XML) and the fragments returned by the search engine is shown as trees in Figure 3(a).

The first step of AIM is to decompose the fragments into candidates path-key pairs and allows user to select his/her desired path-key pairs as shown in Figure 3(b). In this example, we have three sources, F_1, F_2 and F_3 are from the sources, S_A, S_B and S_C respectively. We can see that the path-key pairs are sorted according to their frequency (i.e. the number of their appearance in the raw list). For example, three path-key pairs whose path equal to "/r/pub/author" are at position 1 to 2 of the source S_A. At the first iteration, the weights for the sources are $\{0.3333, 0.3333, 0.3333\}$, therefore the top-3 path-key pairs from each source will be displayed to the user in next iteration. Since there are only nine path-key pairs in this case, we need an additional path-key pair in order to have ten path-key pairs for user to select, we may simply add the fourth path-key pair from either S_A, S_B or S_C, and in this example, we add the fourth path-key pairs from S_A. The ten candidate path-key pairs for next iteration is shown in bold letters in Figure 3(b).

Now, assume the user selects all path-key pairs from S_B and S_C. The result fragment is shown in Figure 3(c). We can see that the result fragment is built as expected. With the user feedback, the weight of S_A is halved and S_B and S_C share the decreased weight of S_A, the new weights are $\{0.1667, 0.4167, 0.4167\}$. The candidate path-key pairs are shown in bold in Figure 3(d).

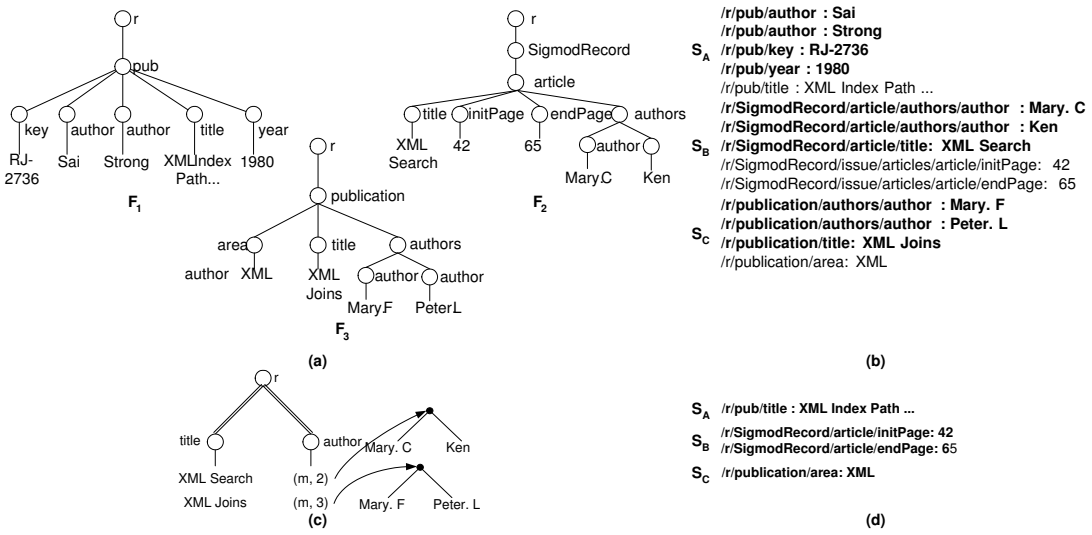

Fig. 3. (a)The returned fragments by the query Q, (b) the corresponding decomposed path-key pairs (c) merged result fragment and (d) candidate path-key pairs after the first iteration

3.2 Four Directional Searching Approaches

In this section we introduce four directional searching approaches used in the re-querying process. They are upward, downward, forward and backward searchings.

Upward Searching. The objective of upward searching is to find a set of fragments with similar structure but different data values. Given a query, Q, and the list of preferred path-key pairs in previous iteration, P. We formulate a re-query, q_i, for each path-key pair in P, $f_i = (p_i, k_i) \in P$, where p_i is the path from the root to the parent node of the keyword, k_i. We check if p_i is located at the root of the document. If yes, we stop, since we cannot go up anymore, otherwise, the re-query is given by "$\rho_0//\rho_n : *$", where ρ_0 is the root of p_i and ρ_n is the parent node of k_i.

Downward Searching. The objective of downward searching is to find a set of fragments which can provide further details according to user preference. We formulate a re-query which aims at the children or siblings of the "preferred" path-key pairs. Given a query, Q, and the list of preferred path-key pairs in previous iteration, P. We formulate a re-query, q_i, for each path-key pair in P, $f_i = (p_i, k_i) \in P$, where p_i is the path from the root to the parent node of the keyword, k_i. The re-query, q_i is given by "$\rho_0//\rho_n/* : k_i$", where ρ_0 is the root of p_i and ρ_n is the parent node of k_i.

The Forward Searching. The core idea of forward searching is to search relevant fragments that are ignored in the initiate query (i.e. the query submitted by the user at the very beginning) by providing more detailed query for more accurate results. Given the initiate query, Q, and the preferred path-key pairs in previous iteration, P. For each path-key pairs $f_i = (p_i, k_i) \in P$, if f_i does

not exactly match with any path-key pairs in Q, we submit the re-query, r_i, as "$//\rho_n : k_i$", where ρ_n is the parent node of k_i.

The Backward Searching. The backward searching is similar to forward searching but in "opposite direction". Backward searching aims to find information that match the initiate query, Q, but are different from the path-key pairs in P. Given a query, Q, and the list of preferred path-key pairs in previous iteration, P. For each path-key pairs $f_i = (p_i, k_i) \in P$, if f_i does not exactly match with any path-key pairs in Q, we submit the re-query, r_i, as " $Q \cup //\rho_n : (NOT\ k_i)$", where ρ_n is the parent node of k_i.

4 Conclusions

An interesting contribution in our proposed framework is to unify the processes of searching XML fragments and merging the users' preferred XML fragments returned from the ranked result list. We suggest rewriting the queries using path-keys of the set of core paths in order to increase the searching coverage. We proposed the approaches of Additive Increment Merging and Directional Searching in order to generate more usable results in a progressive manner.

The ideas presented in this short paper pave the way to promote a wider use of XML data, since fragment search is simple enough for existing users to search the XML information systems. In addition, the merger provides more usable and quality information according to the users' preferences. This paper is a ground work for many interesting issues for further study. For example, we can further examine several schemes in order to estimate path-key similarity in the merging process. This also allows us to extend our framework for searching and merging XML and HTML data, which serves as a more useful tool for searching heterogenous Web data.

References

1. S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for xml. In *Proc. of VLDB*, 2005.
2. J. Bremer and M. Gertz. XQuery/IR: Integrating XML document and data retrieval. In *WebDB*, 2002.
3. D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *SIGIR*, pages 151–158, 2003.
4. T. T. Chinenyanga. Expressive and efficient ranked querying of XML data, 2001.
5. World Wide Web Consortium. Xquery 1.0 and xpath 2.0 full-text.
6. N. Fuhr and K. Großjohann. XIRQL: An extension of XQL for information retrieval, 2000.
7. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents, 2003.
8. T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD '02*.
9. Ho-Lam Lau and Wilfred Ng. A unifying framework for merging and evaluating XML information. In *DASFAA*, pages 81–94, 2005.
10. A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in XML. In *ICDE*, pages 162–173, 2005.
11. Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An efficient and versatile query engine for topx search.