Zhou JF, Ling TW, Bao ZF *et al.* Related axis: The extension to XPath towards effective XML search. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 27(1): 195–212 Jan. 2012. DOI 10.1007/s11390-012-1217-0

Related Axis: The Extension to XPath Towards Effective XML Search

Jun-Feng Zhou[1,2] (周军锋), *Member*, *CCF*, Tok Wang Ling[3] (林卓旺), *Senior Member*, *ACM*, *IEEE*
Zhi-Feng Bao[3] (鲍芝峰), and Xiao-Feng Meng[2] (孟小峰), *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*

[1]*School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China*

[2]*School of Information, Renmin University of China, Beijing 100872, China*

[3]*School of Computing, National University of Singapore, 117417, Singapore*

E-mail: zhoujf@ysu.edu.cn; {lingtw,baozhife}@comp.nus.edu.sg; xfmeng@ruc.edu.cn

Received September 6, 2010; revised July 22, 2011.

Abstract We investigate the limitations of existing XML search methods and propose a new semantics, related relationship, to effectively capture meaningful relationships of data elements from XML data in the absence of structural constraints. Then we make an extension to XPath by introducing a new axis, related axis, to specify the related relationship between query nodes so as to enhance the flexibility of XPath. We propose to reduce the cost of computing the related relationship by a new schema summary that summarizes the related relationship from the original schema without any loss. Based on this schema summary, we introduce two indices to improve the performance of query processing. Our algorithm shows that the evaluation of most queries can be equivalently transformed into just a few selection and value join operations, thus avoids the costly structural join operations. The experimental results show that our method is effective and efficient in terms of comparing the effectiveness of the related relationship with existing keyword search semantics and comparing the efficiency of our evaluation methods with existing query engines.

Keywords XML, XPath, related axis, entity graph, schema summary

1 Introduction

As a de facto standard for information representation and exchange over the Internet, XML has been used extensively in many applications and huge volumes of data are organized or exported in XML format. Extracting desired information from XML data collections using an effective and efficient approach is an important research issue. In practice, however, the feature that data can be flexibly organized with different structures results in two challenges: (C1) the underlying schemas may be too complex to be fully understood by most users; (C2) the given XML documents may be of structural heterogeneity. Together the two challenges make the task of retrieving desired information from the given XML documents no longer a trivial task.

Traditionally, a structured query language, e.g., XQuery① or XPath②, can convey complex semantics and therefore extract precisely the desired information from XML data. However, a prerequisite is that users must fully understand the underlying schema so as to correctly formulate their query expressions, which will impose great burden on most users and is likely infeasible in practice. Moreover, when searching heterogeneous XML documents, any single structured query expression is infeasible in such a case. Using tree pattern queries in conjunction with data integration mapping rules[1] is complex and error-prone, since maintaining the mapping relationship may involve extensive manual intervention. Therefore, structured query methods[2-6] are infeasible to C1 and C2.

Keyword search methods[7-23], on the other hand, free users from the great burden of understanding the underlying schema. However, as shown in Example 1, they may return too many irrelevant results for two reasons: 1) the meaningfulness of an answer is determined by only structure information, 2) structural constraints are prohibited from pruning irrelevant answers.

Example 1. Consider the document D in Fig.1. Using keyword search method, Q_1 and Q_2 in Fig.1 can

Regular Paper

This research was partially supported by the National Science and Technology Major Project of China under Grant No. 2010ZX01042-002-003, the National Natural Science Foundation of China under Grant Nos. 61073060, 61040023, 61070055, 91024032, the Fundamental Research Funds for the Central Universities of China, and the Research Funds of Renmin University of China under Grant No. 10XNI018.

①http://www.w3.org/TR/xquery/

②http://www.w3.org/TR/xpath20/

be written as {Mike, John, item} and {Mike, America, person}, respectively. The basic semantics of [7-9, 11, 13-15, 17-23] are based on *tree* model, thus they cannot capture the meaningful relationships conveyed by IDREF. They may consider f_1, f_2, f_4 (for Q_1) and f_5 (for Q_2) of Fig.2 as matched data fragments. The keyword search semantics of [10, 12, 16] are based on *graph* model (IDREF considered), thus they can find more answers. They may consider f_1 to f_4 as matched data fragments of Q_1, f_5 and f_6 as matched data fragments of Q_2. However, we cannot identify the relationships of the entity (entity has the same meaning as that of ER model) instances (video and person) in f_1 and f_5, since the two nodes (videos and persons) connecting the two *video* nodes in f_1 and the two *person* nodes in f_5 do not convey any useful information. f_2 and f_4 mean that both "Mike" and "John" provide "videos" that contain useful information about the same item named "bowlder" and "bow", respectively. f_3 means that "Mike" bought the item named "gem" which was sold by "John". f_6 means that "John" lives in "America" and he traded with "Mike" in an "auction".

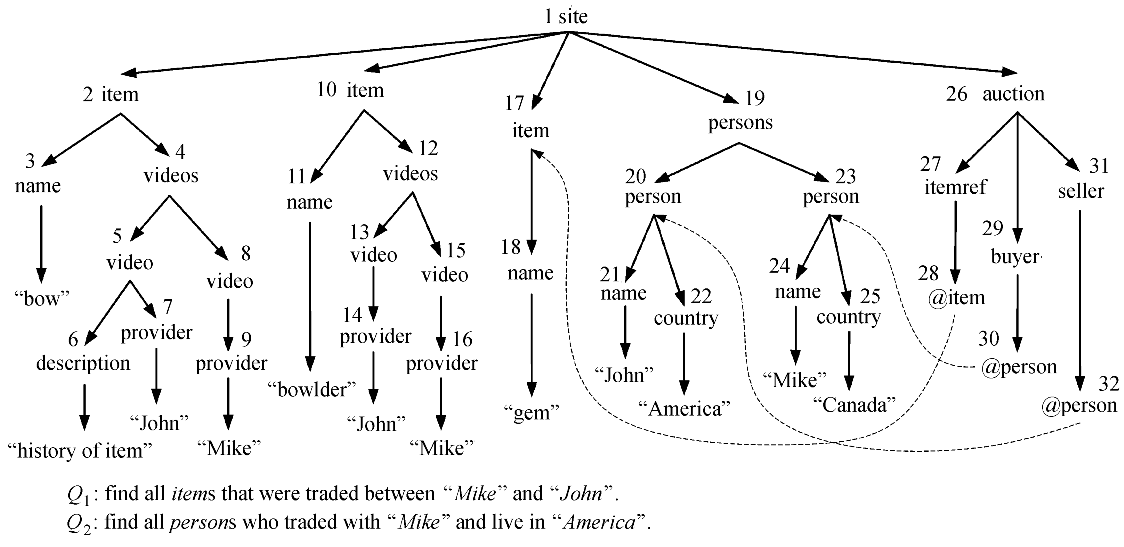


Fig.1. Example auction XML document D, where solid arrows denote the containment relationship, dashed arrows denote the reference relationship, the number beside each node is the id of this node.

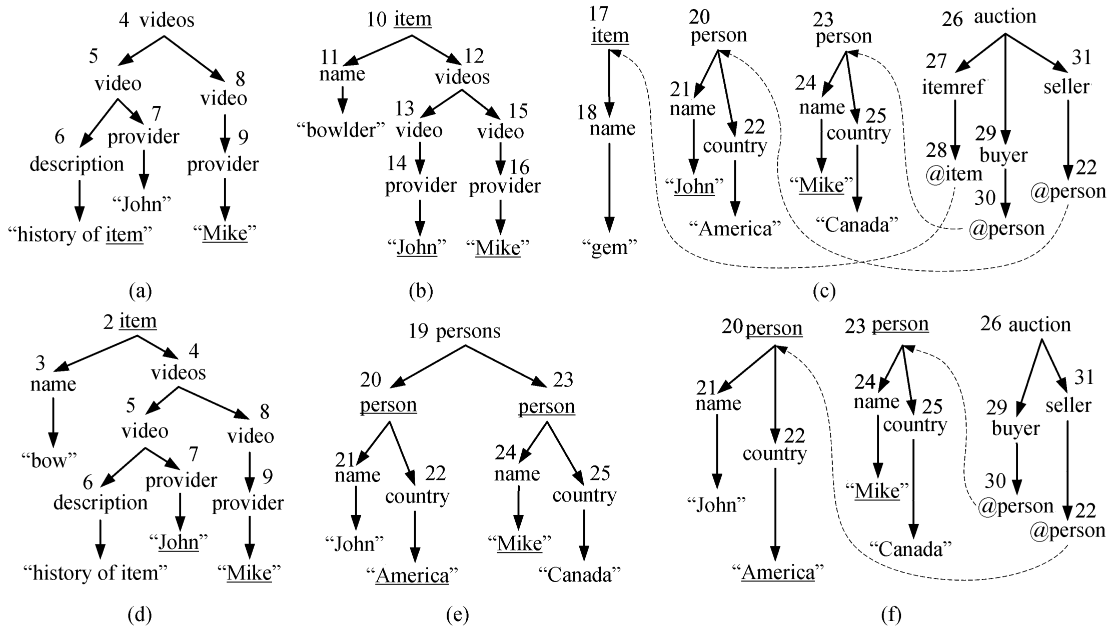


Fig.2. Example data fragments for keyword queries {Mike, John, item} and {Mike, America, person}. (a) f_1. (b) f_2. (c) f_3. (d) f_4. (e) f_5. (f) f_6.

According to the semantics of [16], each pair of entity instances cannot have *sibling* relationship, however, node 5 and node 8 in f_4, node 13 and node 15 in f_2 are entity instances and do not meet this condition, thus f_2 and f_4 are considered as meaningless data fragments. If the users' search intention is themeaning conveyed by f_2 and f_4, this method will lose meaningful answers.

By combining keyword search semantics and structured query language together, the methods proposed in [15-16, 24-26] provide users with a relaxed-structure query mechanism and return approximate answers. The effectiveness of such methods is affected by two factors: 1) keyword search semantics and 2) the requirements on users about the underlying schema. The limitations of the first factor are discussed in Example 1. For the second factor, [15, 24-26] need users to understand partial schema information so as to specify the value-join conditions in the where clause of the XQuery expression. For example, for Q_1, users are required to explicitly specify how *auction* is associated with *person* and *item*. [16] uses a schema summary[27] to help users to formulate their query expressions. The usability of the schema summary is very sensitive to the given query. In some cases, the expanded schema summary may be almost the same as the original schema and difficult for users to understand. Although these methods provide users with more flexibility, they all suffer from the same problem as the structured query methods do since structural constraints are indispensable in some cases.

We argue that when confronting challenges C1 and C2, a good query mechanism should be both *effective* and *flexible*. "*effective*" means that each answer should come from a data fragment that describes the meaningful relationships of entity instances. Such a requirement is based on the observation that users just care about the relationships of the representative data nodes (e.g., photo, video, item, person and auction in Fig.1) which we call entities in ER model and in most of the time their query semantics are based on the relationships of entities. The "*flexible*" means that users can freely incorporate whatever knowledge of the given schema into their query expressions to prune irrelevant answers, while system *does not* impose any requirements about the schema on users. From the above discussion we know that when confronting C1 and C2, no existing method is both *effective* and *flexible*, since they may either return meaningless results (e.g., keyword search methods) or require that users understand (at least partial) the schema (e.g., structured query methods and [15-16, 24-26]).

Motivated by C1, C2 and the above discussion, in this paper, we focus on providing users with an *effective* and *flexible* query mechanism to extract the desired information from the given XML documents.

For "*effective*", we propose a new semantics, *related relationship*, to capture the *meaningful relationships* of data elements from the given XML documents in the absence of structural constraints. For example, according to the related relationship, f_1 and f_5 in Fig.2 will not be considered as meaningful data fragments. The related relationship considers only semantic relationships of data elements, which may be organized with different structures in practice. When formulating query expressions, users can just focus on the desired semantics, rather than the complex or heterogeneous hierarchical structures. Therefore, $C1$ and $C2$ are no longer challenges for the related relationship.

For "*flexible*", we make an extension to XPath by introducing a new axis, *related axis*, to specify the related relationship of two query nodes. Thus users can freely incorporate whatever structural constraints, which are *not the necessary conditions*, into their query expressions to prune irrelevant answers. Further, an extended XPath expression can be seamlessly incorporated into an XQuery expression to convey complex semantic constraints as done in [15-16], and at the same time, have a more concise format.

Example 2. Using the extended XPath, Q_1 in Fig.1 can be written as *//item[/related::*$\ast \sim$*"Mike"][/related::*$\ast \sim$*"John"]*, where "$\ast$" is a query node of any name, "$\sim$" has the same meaning as the built-in function "contains()" of XQuery, which denotes that "Mike" or "John" is contained by the value of the "$\ast$" node. This expression will find all *items* that have the related relationship with the entity instances that contain "Mike" and "John" as their attribute value. For D in Fig.1, there are three entity instances (nodes 8, 15 and 23) containing "Mike", three entity instances (nodes 5, 13 and 20) containing "John". According to the related relationship, node 2 is related with nodes 5 and 8, node 10 is related with nodes 13 and 15, node 17 is related with nodes 20 and 23, i.e., f_2, f_3 and f_4 are considered as meaningful data fragments. Then nodes 2, 10 and 17 will be returned as matched results.

If we want to search data fragments in which "Mike" and "John" are contained by the value of the *name* attribute of two *person* nodes, we can incorporate structural constraints into the above expression, which is then rewritten as *//item[/related::person//name$\sim$"Mike"] [/related::person//name$\sim$"John"]*.

According to the related relationship, only f_3 (f_2 and f_4 are pruned) is considered as a meaningful data fragment, then node 17 is returned as a matched result.

As the related relationship captures meaningful relationships that may be organized in different forms, especially for heterogeneous XML documents, a query expression with related axes may correspond to multiple query expressions without related axes. We call the former *abstract tree pattern* (ATP) and each one of the latter a *query pattern* (QP), which consists of a set of *tree* (*or twig*) *pattern* (TP) queries connected together by reference edges. We show in Section 4 that to evaluate an ATP query, we need to solve the following problems.

P1: identifying all QPs from the underlying schema.

P2: evaluating all QPs against the given XML data.

For P1, we propose to use an entity graph as a schema summary that is generated from the original schema graph by removing the non-entity nodes and preserving the entity nodes and their connection relationships. Based on an entity graph, our method employs a delay-checking strategy so as to avoid the costly I/O operations (compared with [10, 28]) and avoid losing meaningful QPs (compared with [12, 16]).

For P2, we introduce two indices to improve the query performance. The first is an inverted list called *partial path index*. For each keyword k, partial path index stores 1) all entity instances that contain k as their attribute or attribute value and 2) paths from these entity instances to k. The second is also an inverted list called *entity path index*. Entity path index stores all matched entity instances of entity pairs, where for each entity pair, entity nodes are joined by an edge of the entity graph. Compared with [29], our indices are more flexible and materialize the ID/IDREF relationship. Based on the two indices, the costly structural join operations in *most* queries can be equivalently transformed into just a few selection and value join operations.

In summary, our contributions are as follows.

• We propose a new semantics, related relationship, to capture the meaningful relationships of data elements, then make an extension to XPath by introducing a new axis, related axis, to specify the related relationship of two query nodes so as to provide users with a query mechanism of both *effective* and *flexible*.

• We propose to use an entity graph as a schema summary, based on which our method employs a delay-checking strategy to avoid the costly I/O operations and reduce the CPU cost of computing QPs.

• We propose to use partial path index and entity path index to improve the performance of query processing. We further prove the high efficiency of our method, that is, the costly structural join operations of *most* queries can be equivalently transformed into just a few simple selection and value join operations.

• We conduct an extensive experimental study. The experimental results with datasets of various characteristics demonstrate that our method is effective and efficient in terms of various evaluation metrics.

The rest of the paper proceeds as follows. In Section 2, we introduce preliminaries. In Section 3, we introduce the related relationship and the extended XPath. In Section 4, we present our query evaluation method. In Section 5, we report our experimental results. Section 6 is dedicated to related work. We conclude this paper in Section 7.

2 Preliminaries

Schema S. We assume that the schema is always available, as we can use the methods proposed in [30-31] to infer the schema (if unavailable). We use a node labeled directed graph S to model a schema. Formally, $S = (V_S, E_S)$, where V_S denotes a set of schema elements each with a distinct tag name, E_S denotes a set of directed edges between schema elements. As shown in Fig.3, there are two kinds of edges in S. The first is the *containment edge*, which is drawn as a solid arrow from an element (e.g., person) to its child element (e.g., name). Containment edge denotes the *parent-child* (P-C) relationship of data elements in an XML document. The second is the *reference edge*, which is drawn as a dashed arrow from the attribute (e.g., @person) of referrer element (e.g., bidder) to referee element (e.g., person).

Node Categories. In the following discussion, *entity* and *attribute* refer to the notions defined in ER-model,

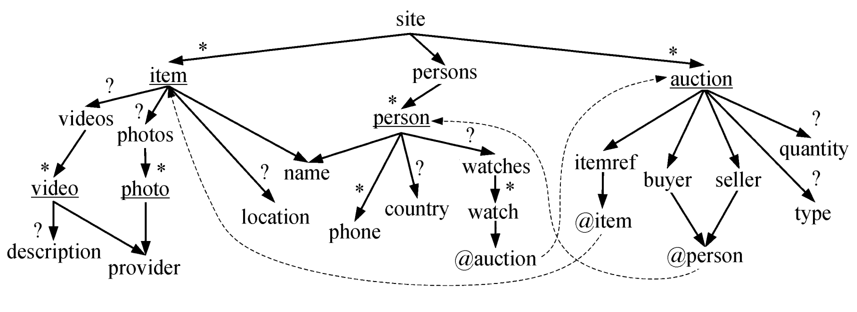


Fig.3. Example schema S derived from XMark.

rather than that defined in XML specification[3]. To facilitate our discussion, *entity* denotes the entity-type, which is an entity node in a schema graph; *entity instance* is the instance of *entity* in XML data. Generally speaking, two kinds of methods can be used to specify the category of schema elements, which are 1) automatic methods using heuristic inference rules[13,16] and 2) manual method done by database administrator (DBA) or domain expert. The inference rules of [13, 16] are as follows.

1) A node represents an entity if it corresponds to a $*$-node in the document type definition (DTD).

2) A node denotes an attribute if it does not correspond to a $*$-node, and only has one child, which is a value.

3) A node is a connection node if it represents neither an entity nor an attribute. A connection node can have a child that is an entity, an attribute or another connection node.

The automatic method can avoid the cost of manual intervention, but it may not be quite correct. The example, for the schema in Fig.3, *person* is a $*$-node, thus by rule 1), *person* represents entity, instead of the attribute of *site*. However, according to the above inference rules, *phone* will be considered as an entity, which is unreasonable. By using manual method from scratch, we can get the accurate category of each schema node, but it may impose great burden on DBA or domain experts.

Therefore, to achieve accurate node categories while paying minimum manual intervention, we first employ the above rules to get an approximate categorization, followed by a minor manual adjustment from DBA or domain expert. In this way, they can accomplish this task easily, even if for heterogeneous schemas. For example, there are twenty-two XML datasets provided in the website http://www.cs.washington.edu/research/projects/xmltk/xmldata/www/repository.html, using our method, all the schema information can be analyzed easily, including XMark benchmark, which has a complex schema[16]. In the following discussion, we take *phone* and *watch* as multi-value attributes of their entities, and the underlined nodes as entities in Fig.3.

Such a method is similar to [32], where users are required to participate in the task of identifying entities, the difference is that in this method, users are isolated from the task. Further, this method is different from the "entity extraction" of [33], where entity is assumed to be loosely defined on the contents of web pages, not schema, by a generic category.

In this paper, we mainly focus on designing an *effective* and *flexible* query mechanism based on the results of existing classification methods, not how to improve the accuracy of existing classification methods. The notations used in our discussion are shown in Table 1.

[3]http://www.w3.org/TR/REC-xml/

Table 1. Summary of Notations

Notation	Description
MEW	Meaningful entity walk
ATP	Abstract tree pattern
QP	Query pattern
TP	Tree (or twig) pattern
PPI	Partial path index
EPI	Entity path index

3 Related Relationship

Before formally defining the related relationship, having an intuitive impression on the data organization of meaningful relationship will facilitate understanding our definitions. From Fig.1, we know that person "Mike" bought the item "gem", which can be expressed as $R1$: "person⇠@person←buyer←auction→itemref→ @item⇢item". $R1$ denotes that a meaningful relationship between entity instances may contain edges of different directions (*mixed directions* problem) and cannot be identified by simply traversing the document. Even if we omit the direction of each edge, directly traversing an XML document of large size is usually time consuming. An alternative way is finding such relationship from a schema graph, which has much smaller size. However, some relationships produced in this case may be meaningless (*meaningfulness* problem). For example, $R2$: "item→name←person" is a possible relationship produced by traversing the undirected schema graph S_u of S in Fig.3, such relationship is meaningless since according to S, person and item must have different child element named "*name*" in XML documents, which contradicts $R2$. Thus, we need an effective way to capture both "*mixed directions*" and "*meaningfulness*", so as to avoid losing meaningful relationships (e.g., $R1$) by traversing directed graph and producing meaningless relationships (e.g., $R2$) by traversing undirected graph.

Definition 1 (Walk). *A $v_0 - v_k$ walk W : $v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k$ of the undirected schema graph S_u is a sequence of vertices of S_u beginning with v_0 and ending at v_k, such that each two consecutive vertices v_{i-1} and v_i are joined by an edge e_i of S_u. The number of edges of W is called the length of W, which is denoted as $L(W)$. For any two nodes u and v of S_u, if there exists at most one edge joining u and v in S_u, W can be written as*

W : $v_0, v_1, \ldots, v_k$ after removing edges. Any $v_i - v_j$ walk W' : $v_i, e_{i+1}, v_{i+1}, \ldots, v_{j-1}, e_j, v_j$ $(0 \leqslant i \leqslant j \leqslant k)$ extracted from W is called a sub-walk of W, which is denoted as $W' \subseteq W$.

Intuitively, a walk denotes a possible connection relationship of two schema nodes where direction is not considered. Note that Definition 1 does not require the listed vertices and edges to be distinct, there may be more than one walk between two nodes. We define walk so as to avoid the problem of "*mixed directions*", and Definition 2 is used to avoid the problem of "*meaningfulness*" and capture the meaningful connection relationship of two entities.

Definition 2 (Meaningful Entity Walk, MEW). *Let S be a schema graph, a $v_0 - v_k$ walk W of the undirected schema graph S_u is a meaningful entity walk of S if both v_0 and v_k are entity nodes and*

• *$L(W) \leqslant 1$, or*

• *W does not contain a sub-walk W' that has the form $u \rightarrow v \leftarrow w$ in S, where "$\rightarrow$" denotes a solid arrow from $u(w)$ to v in S. Moreover, if W' has the form $u \leftarrow v \rightarrow w$ in S, v must be an entity node.*

Example 3. Consider the walks in Table 2, where *person*, *photo*, *video*, *item* and *auction* are five entities. As two keywords may be attributes or attribute values of the same entity instance, according to Definition 2, W_1 is an MEW. W_2 is not an MEW according to Definition 2, the reason was discussed in the first paragraph of this section, where W_2 is denoted as $R2$. W_3 means that a person is watching an auction, according to Definition 2, W_3 is an MEW. W_4 means that both photo and video belong to the same item, according to Definition 2, W_4 is an MEW. We cannot explain intuitively the relationship of *person* and *item* in $W5$ as they are connected together through a connection node, i.e., *site*. According to Definition 2, W_5 is not an MEW since W_5 has the form "item←site→person" and site is not an entity. W_6 means that both the two items are sold in the same auction. According to Definition 2, W_6 is an MEW.

Table 2. Example Walks

W_1	person
W_2	person→name←item
W_3	person→watches→watch→@auction$\dashrightarrow$auction
W_4	photo←photos←item→videos→video
W_5	item←site→person
W_6	item$\dashleftarrow$@item←itemref←auction→itemref→@item$\dashrightarrow$item

Note that for simplicity, we do not mention in Definition 2 other constraints of the given XML documents and the schemas. For example, both W_4 and W_6 are MEWs according to Definition 2. However, from Fig.1 we know that no instance of W_4 exists, from Fig.3 we know that each auction corresponds to just one item, which contradicts W_6. Thus W_4 and W_6 should be discarded, which will be further discussed in Examples 5 and 6.

In practice, users may just provide attributes or attribute values rather than entity names when submitting their queries. For example, for query Q_1 in Fig.1, the meaningful relationship is not based on *item* and "*Mike*", but item and entities that have entity instances containing "Mike" as their attribute value. Thus if a node is not an entity, we need to identify which entities it belongs to.

Definition 3 (Self Entity Node). *Let u, v be schema elements, u an entity node. We say u is a self entity node of v if $v = u$ or v is an attribute node of u in the schema graph, which is denoted as $u \in selfE(v)$. For a text value of an XML document, $selfE(k)$ consists of entities that have entity instances with attribute values directly containing k.*

According to Definition 3, D in Fig.1 and S in Fig.3, we have selfE(auction)={auction}, selfE(name)={item, person} and selfE("Mike")= {person, video}. Then we formally define the related relationship and the combination of the related relationship and XPath grammar as follows.

Definition 4 (Related Relationship). *Let u and v be two schema nodes. We say u and v have related relationship if there exists a node $u' \in selfE(u)$ and $v' \in selfE(v)$, such that there exists at least one MEW between u' and v'.*

According to Definition 4, if u and w are sibling nodes and they have a common entity ancestor v, then u and w have the related relationship; even if u and w have not a common entity ancestor, if they are entity nodes and referenced by another entity node, they are still related. Our method cares about only which schema node is entity, not the structural discrepancy, thus we do not need a single common schema and users are freed from understanding the underlying schema, especially when querying heterogeneous XML documents.

Definition 5 (RelatedStep). *Let V be the set of entities of the currently being processed context node, a RelatedStep returns a sequence of nodes that are reachable from an entity instance of $v \in V$, via a related axis.*

Definition 6 (Related Axis). *The related axis contains data elements that have related relationship with the entity instances of the self entity nodes of the context node.*

The related axis has a very simple syntax that can be seamlessly integrated into XPath. Fig.4(a) is the EBNF grammar for the axis, where the newly introduced symbols are underlined. There are totally 82 rules in the current XPath grammar, among which only the 26th and 28th rules have to be modified. RelatedStep is further defined by the new rules [$n1$] and [$n2$] and

[26] RelativePathExpr	::=	StepExpr ((“/”|“//”|”~>”) StepExpr)*
[27] StepExpr	::=	FilterExpr | AxisStep
[28] AxisStep	::=	(ReverseStep | ForwardStep | ***RelatedStep***) PredicateList
[n1] ***RelatedStep***	::=	***RelatedAxis*** NoteTest
[n2] ***RelatedAxis***	::=	(“related” “::”)

(a)

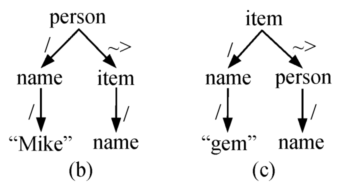


Fig.4. EBNF grammar for the extended XPath with related axis and two ATP queries Q_3 and Q_4. (a) EBNF grammar for the extended XPath with related axis. (b) Q_3. (c) Q_4.

represented by the related axis. Besides the related axis, we also introduce a separator, “~>”, to indicate the semantic constraint of the related relationship, thus a path expression consisting of a series of step expressions may be separated by “~>”, e.g., “person~>item” is short for “child::person/related::item”.

Definition 7 (Abstract Tree Pattern, ATP). *An abstract tree pattern is a tree $Q = (V, ED)$ where*:

- *associated with each node $v \in V$ is P_v, which specifies the predicate for individual node match*;
- *associated with each edge $e \in ED(e = (u, v))$ is SR_e, which specifies the relationship between u and v ($SR_e \in \{/, //, \sim>\}$)*;
- *associated with one and just one node $u \in V$ is a rectangle, which denotes that u is the return node.*

Example 4. Consider the two ATP queries in Fig.4, Q_3 is used to find the name of items that are related with the person whose name is “Mike”, which can be written as “//person[name=“Mike”]~>item/name”. Similarly, Q_4 can be written as “//item[name= “gem”]~>person/name”, which is used to find the name of persons that have the related relationship with the item which has name “gem”.

Although most existing studies[2-5] on twig pattern query focus on returning the entire twig results, in practice, returning the entire twig results is seldom necessary and may cause duplicate elimination or ordering problems[6]. This problem also holds for an ATP query, as a result, a formal definition of ATP matching is as below.

Definition 8 (ATP Matching). *Let Q be an ATP query, D the given XML document, $\mathcal{Q}$ the set of QPs of Q. A match of Q in D is identified by a mapping from nodes in any $Q' \in \mathcal{Q}$ to nodes in D, such that*: 1) *query node predicates are satisfied by the corresponding database nodes*; 2) *the structural relationships between query nodes are satisfied by the corresponding database nodes. The answer set $\mathcal{R}$ to Q consists of the distinct database nodes from the matches of all queries in D, where each answer $d \in \mathcal{R}$ is the text value of the return node of Q.*

4 Query Processing

The computation of a related relationship, e.g., $A \sim> B$, is finding from the given schema graph the set of MEWs between each pair of entities of which one comes from selfE(A) and the other selfE(B). Such an operation equals to finding from the given schema graph all QPs for keyword query $Q = \{A, B\}$. Then a natural question is:

why not consider an ATP query Q as a keyword query and directly apply existing keyword search methods to find matched results that satisfy the structural constraints of Q?

Since an ATP query may contain *ancestor-descendant* edges, existing keyword search method cannot return all matched results when involving recursive nodes. For example, for XML document “$A1/B1/A2/B2$”, query “$A//B$” will return both $B1$ and $B2$ as matched results. However, for keyword query $Q = \{A, B\}$, existing keyword search methods, e.g., MLCA[15] or XSEarch[9], will only contain $B2$ in the matched results. Therefore an ATP query cannot be equivalently represented using a restricted keyword query.

Therefore, to process a given ATP query Q, the basic idea of our method can be simply stated as: firstly identify all QPs of the given ATP query by replacing each related edge (~>) of Q by one of its corresponding MEW; then evaluate these QPs to find matched results, which correspond to Algorithm 1 and Algorithm 2, respectively. To find all QPs of the given ATP query, we propose to use a simplified schema graph, i.e., entity graph, as a schema summary to accelerate the computation in Subsection 4.1. For each QP of Q, we propose two efficient indexes, i.e., partial path index and entity path index in Subsection 4.2, based on which the costly structural join operations of most queries can be equivalently transformed into just a few simple selection and value join operations.

As shown in Section 6, identifying all QPs from the given schema graph is not a trivial task, we introduce a new schema summary in the following subsection to solve this problem.

4.1 Computation of Query Patterns

Definition 9 (Entity Path). *Path p: $v_1, v_2, \ldots, v_k$ ($2 \leqslant k$) of schema graph S is called an entity path if it* *is a directed path from v_1 to v_k, where only v_1 and v_k are entity nodes, and for any v_i $(2 \leqslant i \leqslant k-1)$, $v_i \neq v_j$ $(1 \leqslant j \leqslant k \wedge i \neq j)$.*

Definition 10 (Partial Path). *A partial path p is a path starting with an entity that is the only entity node of p.*

Intuitively, an entity path describes the direct relationship of two entities, a partial path denotes containment relationship of an entity and one of its attributes. To get the partial path information of any node, we maintain an inverted index H that stores the set of partial paths (*not their instances*) for each keyword k appearing in the given XML document D, which can be got after parsing D. Thus from H, we can get for each keyword k, the set of partial paths and selfE(k) easily. For example, H stores "video/provider, person/name" for "Mike". According to H, selfE("Mike")={video, person}.

Definition 11 (Entity Graph). *Let S be a schema graph, $\mathcal{P}$ the set of entity paths of S. The* entity graph *of S is represented as $G = (V, E)$, which consists of only entity nodes of S and for each entity path $p \in \mathcal{P}$ that connects u and v in S, there is an edge in G that joins u and v.*

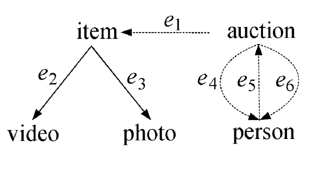
As shown in Fig.5, in an entity graph, two entity nodes may be joined by two or more edges, e.g., person and auction are joined by e_4, e_5, e_6. Each edge of an entity graph may be a containment edge (solid arrow) or reference edge (dashed arrow). Each containment edge of G denotes an entity path that consists of just containment edges in S and each reference edge denotes an entity path that consists of at least one reference edge in S. Compared with the original schema graph S, entity graph G is much smaller and it captures the related relationship among entities without any loss. The construction of an entity graph is very simple according to Definition 11, we omit it for limited space. According to Definition 11, we have the following lemma, from which we know that all MEWs can be computed from an entity graph.

Lemma 1. *There exists a one-to-one mapping between the edges of an entity graph and the entity paths of the original schema graph.*

As shown in [28], a joining sequence of two data elements in an XML document is data bound[4], so is the MEW. Considering this, users are usually required to specify the maximum size C that equals the number of edges in a joining sequence in existing methods[10,28].

However, such a method may result in two problems, 1) returning results of very weak semantics, 2) losing meaningful results. For example, W_3 and W_4 in Table 2 are two MEWs and $L(W_3) = L(W_4) = 4$, thus the semantic strongness of the two MEWs should be equal to each other, if all edges have the same weight. Obviously, the semantic strongness of W_3 and W_4 is not equal to each other if the semantic strongness is measured based on the number of entity nodes. W_3 denotes the direct relationship between person and auction, while W_4 manifests that photo has related relationship with video through another entity (*item*). If $C = 4$, W_4 is returned, which conveys very weak semantics; otherwise, if $C = 3$, W_3 will not be returned even if it conveys a direct relationship. To avoid these problems, in our method, *the C imposed on MEW is not the maximum number of edges, but that of entity nodes.* This C is a user-specified variable with a default value of 3.

Algorithm 1 is used to compute the set of QPs of the given ATP query. There are three input parameters, Q is the ATP query, G the entity graph and C the constraint value which specifies the maximal number of entity nodes in an MEW. isEmpty(Q_X) is used to check whether queue Q_X is empty. removeHead(Q_X) is used to remove from Q_X the first element. getTail(W) is used to get the last node of W. addTail(Q_X, X) is used to put X at the end of Q_X. isMEW(W) is used to check whether W is an MEW. selfE(u) denotes the set of self entity nodes of u. In lines 1~2, if Q does not contain related edge, Q is returned. Otherwise, Q is put into a queue Q_Q of queries in line 3. If Q_Q is not empty, we get an ATP query Q in line 5. In line 6, we get a related edge $n_1 \sim> n_2$ of Q. For each pair of entity nodes (u, v) where $u \in selfE(n_1)$ and $v \in selfE(n_2)$, we get the set of MEWs of u and v in lines 9~23. For each MEW $W \in \mathcal{W}_{MEW}$, we use it to replace the corresponding related edge $n_1 \sim> n_2$ in line 27 and merge W with the other parts of Q to generate a set of ATP queries $\mathcal{Q}_W$. In lines 28~32, for each newly produced ATP query $Q' \in \mathcal{Q}_W$, if Q' does not contain any related edge, it is put into the QP set $\mathcal{Q}$; otherwise, it is put



e_1 (-->): auction -> itemref -> @item --> item
e_2 (->): item -> videos -> video
e_3 (->): item -> photos -> photo
e_4 (-->): auction -> buyer -> @person --> person
e_5 (-->): person -> watches --> watch -> @auction --> auction
e_6 (-->): auction -> seller -> @person --> person

Fig.5. Entity graph G of S in Fig.3.

[4]Data bound means the size of a connection sequence may be as large as the number of nodes in an XML document.

into Q_Q for further processing. Finally, all QPs are stored in $\mathcal{Q}$ and returned in line 34.

Example 5. Consider Q_1 in Fig.1, which is expressed as the ATP Q in Fig.6. Assume that the first processed related edge is "$I \sim\!> * \sim$'M'" and $C = 3$. According to D in Fig.1, we have selfE("M") $= \{V, P\}$, selfE(I) $= \{I\}$. According to lines 8∼24 of Algorithm 1, we need to compute the set of MEWs of (I, V) and (I, P). For (I, V), we get one MEW W_1 in Fig.6. For (I, P), we get three MEWs shown as W_2 to W_4 in Fig.6. In lines 26∼33, we get four new ATP queries shown as Q_1 to Q_4. Note that all MEWs are shown with just entity nodes and entity paths for simplicity. All the four queries are put into Q_Q, then for each query in Q_Q, we process the second related edge, i.e., "$I \sim\!> * \sim$'J'". Similarly, selfE("J") $= \{V, P\}$ and we get the same set of MEWs for (I, V) and (I, P), i.e., W_1 to W_4. We illustrate the following operations using W_4 as an example. According to lines 26∼33, W_4 needs to replace the related edge of Q_1 to Q_4 and then generate all possible QPs. By applying W_4 to Q_1, we get Q_{41}. The combination of W_4 and Q_2 generates Q_{42} and Q_{43}. Similarly, the combination of W_4 and Q_3 generates Q_{44} and Q_{45}, and the combination of W_4 and Q_4 generates Q_{46} to Q_{48}. Further, in line 27, when generating all possible QPs, our method will consider other constraints to filter out meaningless QPs. For example, according to the schema in Fig.3, each auction instance must correspond to just one buyer, one seller and one item, then we can safely discard Q_{42}, Q_{44}, Q_{46} and Q_{47}. Note that we omit the partial paths of each QP for simplicity.

Theorem 1 (Completeness). *For a given ATP query, Algorithm* 1 *produces all QPs that satisfy each MEW used has at most C entity nodes.*

The correctness of Theorem 1 is obvious according to Algorithm 1 and Example 5. In lines 4∼25, we generate all MEWs, then in lines 26∼33, we replace each related

```
Algorithm 1. queryRewriting($Q$, $G$, $C$)
1:  if ($Q$ does not contain related edge) then
2:      return $\{Q\}$
3:  addTail($Q_Q, Q$)                 /*queue of queries*/
4:  while ($\neg$ isEmpty($Q_Q$)) do
5:      $Q \leftarrow$ removeHead($Q_Q$)
6:      pick a related edge $n_1 \sim\!> n_2$ of $Q$
7:      $\mathcal{W}_{MEW} \leftarrow \varnothing$
8:      foreach ($u \in selfE(n_1)$, $v \in selfE(n_2)$) do
9:          $W \leftarrow u$
10:         addTail($Q_W, W$)         /*queue of walks*/
11:         if ($u = v$) then $\mathcal{W}_{MEW} \leftarrow \mathcal{W}_{MEW} \cup \{W\}$
12:         while ($\neg$ isEmpty($Q_W$)) do
13:             $W \leftarrow$ removeHead($Q_W$)
14:             $v' \leftarrow$ getTailNode($W$)
15:             foreach (adjacent edge $e$ of $v'$ in $G$) do
16:                 find a neighbor node $w$ of $v'$ joined by $e$
17:                 $W' \leftarrow W + ew$
18:                 if ($w = v \wedge$ isMEW($W'$)) then
19:                     $\mathcal{W}_{MEW} \leftarrow \mathcal{W}_{MEW} \cup \{W'\}$
20:                 if ($L(W') < C - 1$) then
21:                     addTail($Q_W, W'$)
22:             endfor
23:         endwhile
24:     endfor
25: endwhile
26: foreach ($W \in \mathcal{W}_{MEW}$) do
27:     $\mathcal{Q}_W \leftarrow$ generate all ATP queries by merging
            $W$ with the other parts of $Q$
28:     foreach ($Q' \in \mathcal{Q}_W$) do
29:         if ($Q'$ does not contain related edge) then
30:             $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q'\}$
31:         else addTail($Q_Q, Q'$)
32:     endfor
33: endfor
34: return $\mathcal{Q}$
```

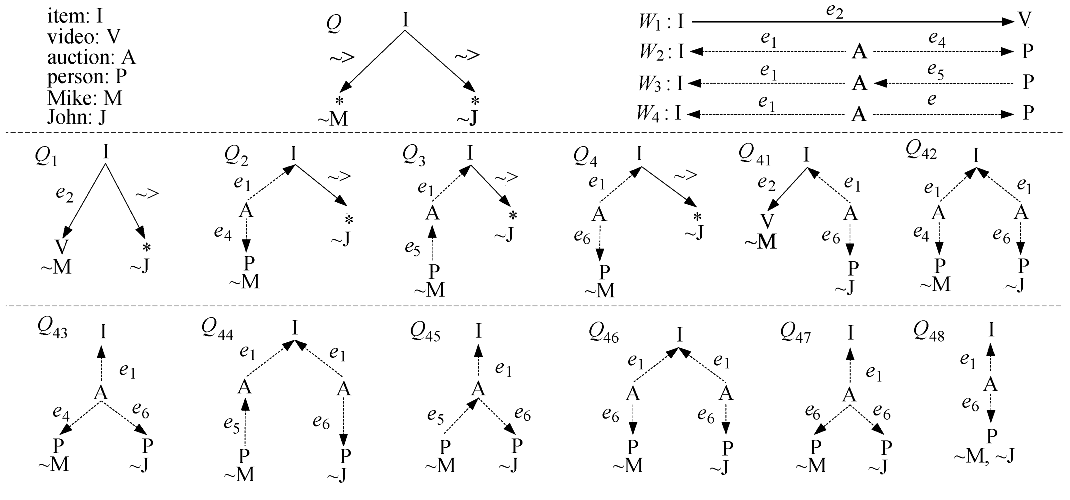


Fig.6. Illustration of the query rewriting, where Q is the original ATP query, $W_1 \sim W_4$ are four MEWs, $Q_1 \sim Q_4$ are ATP queries after processing the first related edge "I$\sim\!> * \sim$'M'", $Q_{41} \sim Q_{48}$ are all QPs by applying W_4 on $Q_1 \sim Q_4$.

edge by one of its MEW to generate all possible QPs, more detailed discussion is omitted for limited space. Let F be the average fan-out of G, C the maximal number of entity nodes of each MEW, N the maximal number of elements in selfE(n). The worst case time complexity of Algorithm 1 is $O(|Q|N^2F^C)$.

Compared with the methods proposed in [10], our method made improvements from two aspects: 1) Algorithm 1 avoids the costly I/O operation of accessing inverted list; 2) Algorithm 1 is based on an entity graph, which is much smaller than the original schema graph. Compared with [12, 16], our method avoids losing meaningful QPs.

4.2 Query Evaluation

Let $\mathcal{Q}$ be the set of QPs of Q, for a QP $Q' \in \mathcal{Q}$, a naive method is first evaluating each TP query of Q', then combining the results of all TP queries together to get the result set $R_{Q'}$ of Q'. Then the result set $\mathcal{R}$ of Q can be written as $\mathcal{R} = \bigcup_{Q' \in \mathcal{Q}} R_{Q'}$. Let L be the average number of data elements of each tag index[2], N_a the average number of query nodes in a QP, obviously, the worst space and time complexity of such method is $O(|\mathcal{Q}|N_aL)$.

To accelerate the evaluation of QPs, we firstly introduce *partial path index* (*PPI*). For each keyword k, we store in *PPI* a list of tuples of the form $\langle EN, ID_{EN}, TN, ID_N, P\rangle$ where EN denotes the entity name, i.e., $EN \in selfE(k)$, ID_{EN} is the id of database instance of EN, TN is the tag name of the data element that directly contains k and denoted by its id ID_N, P is the partial path from EN to TN in schema S. The *PPI* of D in Fig.1 is shown in Table 3, where only partial content is presented. Note that we can reduce the size of *PPI* by storing an integer for each EN, TN and P in *PPI* through constructing the mapping relationship between each EN, TN, P and an integer.

Table 3. Partial Path Index of D in Fig.1

Keyword	Tuple Sets
name	⟨*item*, 2, *name*, 3, *item/name*⟩
	⟨*item*, 10, *name*, 11, *item/name*⟩
	⟨*item*, 17, *name*, 18, *item/name*⟩
	⟨*person*, 20, *name*, 21, *person/name*⟩
	⟨*person*, 23, *name*, 24, *person/name*⟩
Mike	⟨*video*, 8, *provider*, 9, *video/provider*⟩
	⟨*video*, 15, *provider*, 16, *video/provider*⟩
	⟨*person*, 23, *name*, 24, *person/name*⟩
John	⟨*video*, 5, *provider*, 7, *video/provider*⟩
	⟨*video*, 13, *provider*, 14, *video/provider*⟩
	⟨*person*, 20, *name*, 21, *person/name*⟩

The second index is *entity path index* (*EPI*). For each entity path p corresponding to an edge of the entity graph, we maintain a set of tuples of the form $\langle ID_1, ID_2\rangle$, where each tuple denotes the ID pair of a pair entity instances that are connected by p in the given XML document. The entity path index of D in Fig.1 is shown in Table 4.

Table 4. Entity Path Index of D in Fig.1

Path	Instances	Path	Instances
e_1	$\{\langle 26, 17\rangle\}$	e_4	$\{\langle 26, 23\rangle\}$
e_2	$\{\langle 2, 5\rangle, \langle 2, 8\rangle, \langle 10, 13\rangle, \langle 10, 15\rangle\}$	e_5	$\varnothing$
e_3	$\varnothing$	e_6	$\{\langle 26, 20\rangle\}$

Compared with [29], each path in "*path_value index*" of [29] starts from the root of an XML document, while the path of *PPI* starts from an entity node, which is more flexible. Moreover, *EPI* materializes the *reference* relationship, for the "*path index*" of [29], each key must be a path that contains *only* containment edges.

Theorem 2. *Let Q be the given ATP query, S the schema graph. Using PPI and EPI, structural join operation can be avoided from the evaluation of any "/" and "$\sim>$" edge of Q. For any "//" edge $e = (u, v)$ of Q (u and v are schema nodes), if no cycle of containment edge exists between u and v in S and neither of u and v lies in a cycle of containment edge, e can be evaluated without structural join operation.*

Proof. [Sketch] There are three kinds of edges in Q, i.e., "/", "//" and "$\sim>$". Each "$\sim>$" edge will be replaced by MEWs after executing Algorithm 1, where each MEW consists of a set of entity paths. Since the results of each entity path can be got from *EPI*, the results of each MEW can be got by joining the results of entity paths. Thus all "$\sim>$" edges can be processed without structural join operation.

All "P-C" edges form different *maximal* "P-C" *paths*[5]. For each maximal "P-C" path p of Q, it may appear to be one of following forms. 1) $F1$: p contains no entity node. We can get the matched results of p using the leaf node of p as a keyword to search *PPI*, then return all instances of partial paths that contain p. 2) $F2$: p consists of a set of partial paths (recall that the first node of a partial path is an entity node). Obviously, we can get the results of each partial path from *PPI*, then get the results of $F2$ by joining the results of all partial paths. 3) p equals $F1/F2$. We can get the matched results by joining the results of $F1$ and $F2$.

For each "ancestor descendant" ("A-D") edge "$A//B$" of Q, assume A and B are entity nodes. If there exists a "P-C" path p from A to B in S and p contains some nodes that are in a circle of containment edge, then we may produce infinite "P-C" paths from A to B; otherwise, if there exists only one "P-C" path p from A to B in S, according to Definition 2, p is an

[5]Path p of Q is a "P-C" path if it consists of only "P-C" edges, p is a maximal "P-C" path if no other "P-C" path contains p.

MEW and can be processed using *EPI*, i.e., structural join operation can be avoided. If A or B is not an entity node, we firstly get selfE(A) or selfE(B), of which each element is an entity node, the following operations are the same as just stated. □

Therefore we can transform the evaluation of related edges, "P-C" edges and some "A-D" edges into simple selection and value join operations.

Definition 12 (Related Entity Node, REN). *Let Q be the given ATP query, Q' is a QP of Q. Entity node E_K of Q' is a related entity node if there exists a related edge $e = (n_1, n_2)$ in Q, such that E is on one of the set of MEWs of e, where E is the name of the entity node, K is the keyword set covered by the subtree rooted at E, after excluding all other subtrees that are rooted at E's descendant entity nodes. For two related entity nodes $E^1_{K_1}$ and $E^2_{K_2}$, we say $E^1_{K_1} = E^2_{K_2}$ if $E^1 = E^2 \wedge K_1 = K_2$.*

Intuitively, an REN is a node on one MEW of the processed QP, it may have associated keywords that are the value predicates of the ATP query Q. For Q_{43} in Fig.6, there are four RENs, i.e., $I_\varnothing$, $A_\varnothing$, $P_{\{M\}}$ and $P_{\{J\}}$.

Algorithm 2 shows the detailed steps of query processing. In line 1, we get a set of QPs $\mathcal{Q}$. In lines 2∼13, we check for each REN E_K of a QP Q', whether E_K has entity instances that contain all keywords in K. If E_K does not have such entity instances, $R_{E_K} = \varnothing$. For each QP $Q' \in \mathcal{Q}$, if there is an REN E_K and $R_{E_K} = \varnothing$, Q' is removed from $\mathcal{Q}$ in line 16. In line 17, we get the set of selection operations of Q', which is denoted as $S_{Q'}$. If there exists a selection operation which has empty result set, Q' is removed from $\mathcal{Q}$ in line 18. In line 20, we firstly do selection operations, then store the results of value join operations of $S_{Q'}$ in R_σ. In line 21, if there are structural join operations that cannot be equivalently transformed into selection operations, we do structural join operations that take the result set R_σ as input, then store the final matched answers of Q' in $\mathcal{R}_{Q'}$. In line 22, all matched answers are merged into $\mathcal{R}$ and returned in line 24.

```
Algorithm 2. indexMerge($Q, G, C$)
1:  $\mathcal{Q} \leftarrow$ queryRewriting($Q, G, C$)
2:  foreach (REN $E_K \in Q' \wedge Q' \in \mathcal{Q} \wedge E_K \notin \mathcal{E}$) do
3:      if ($K \neq \varnothing$) then
4:          $R_{E_K} \leftarrow$ merge results of selection operations
                of each keyword of $K$ against $PPI$
5:      else continue
6:      foreach ($E'_{K'} \in \mathcal{E} \wedge label(E) = label(E')$) do
7:          $R = R_{E_K} \cap R_{E'_{K'}}$
8:          if ($K \subset K'$) then $R_{E_K} \leftarrow R_{E_K} - R$
9:          else if ($K \supset K'$) then $R_{E'_{K'}} \leftarrow R_{E'_{K'}} - R$
10:         else {$R_{E_K} \leftarrow R_{E_K} - R$; $R_{E'_{K'}} \leftarrow R_{E'_{K'}} - R$}
11:     endfor
12:     $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_K\}$
13: endfor
14: foreach ($Q' \in \mathcal{Q}$) do
15:     if ($\exists E_K \in \mathcal{E} \wedge E_K \in Q' \wedge R_{E_K} = \varnothing$) then
16:         {$\mathcal{Q} \leftarrow \mathcal{Q} - \{Q'\}$; continue}
17:     $S_{Q'} \leftarrow$ getSelOperation($Q'$)
                        /*$S_{Q'} = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$*/
18:     if ($\exists \sigma \in S_{Q'} \wedge R_\sigma = \varnothing$) then $\mathcal{Q} \leftarrow \mathcal{Q} - \{Q'\}$
19:     else
20:         $R_\sigma \leftarrow$ results of selection operations of $Q'$
21:         $\mathcal{R}_{Q'} \leftarrow$ results of structural join operations
                based on $R_\sigma$
22:         $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_{Q'}$
23: endfor
24: return $\mathcal{R}$
```

Example 6. Consider the query Q in Fig.6 again. Q_{41}, Q_{43}, Q_{45} and Q_{48} are four QPs of Q after query rewriting (as stated in Example 5, Q_{42}, Q_{44}, Q_{46} and Q_{47} are not considered as matched QPs). In lines 2∼11, we compute the query result of each REN in these QPs. According to *PPI* in Table 3, we have $R_{V_{\{M\}}} = \{8, 15\}$, $R_{P_{\{J\}}} = \{20\}$, $R_{P_{\{M\}}} = \{23\}$ and $R_{P_{\{M,J\}}} = \varnothing$. In lines 15∼16, Q_{48} is removed from $\mathcal{Q}$ since $R_{P_{\{M,J\}}} = \varnothing$. For the remainder QPs, the selection operations are shown in Table 5. From Table 4 we know that $R_{\sigma_{e_3}} = \varnothing$ and $R_{\sigma_{e_5}} = \varnothing$, then in line 18, Q_{45} is removed from $\mathcal{Q}$. Therefore the QPs processed in lines 20∼22 are Q_{41} and Q_{43}. Assume that the set of selection conditions for a QP Q_i is $S_{Q_i} = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, then the matched results of Q_i is $\mathcal{R}_{Q_i} = \pi_I(R_{\sigma_1} \bowtie R_{\sigma_2} \bowtie \cdots \bowtie R_{\sigma_k})$. Take Q_{43} as an example, $R_{\sigma_{(P,M)}} = \{(23)\}$, $R_{\sigma_{(P,J)}} = \{(20)\}$, $R_{\sigma_{e_1}} = \{(26, 17)\}$, $R_{\sigma_{e_4}} = \{(26, 23)\}$, $R_{\sigma_{e_6}} = \{(26, 20)\}$, $\mathcal{R}_{Q_{43}} = \pi_I\ (R_{\sigma_{(P,M)}} \bowtie R_{\sigma_{(P,J)}} \bowtie R_{\sigma_{e_1}} \bowtie R_{\sigma_{e_4}} \bowtie R_{\sigma_{e_6}}) = \{17\}$. Similarly, $\mathcal{R}_{Q_{41}} = \varnothing$. Finally, $\mathcal{R}_Q = \mathcal{R}_{Q_{41}} \cup \mathcal{R}_{Q_{43}} \cup \cdots = \{2, 10, 17\}$. Note that not all QPs are shown in Fig.6. For simplicity, we omit in Algorithm 2 that we can make further optimization by sharing the results of some selection operations between different QPs, which is illustrated in Fig.7, where the results of selection and value join are shown in each "⟨ ⟩".

Table 5. Selection Operations for Different QPs

Query	Set of selection conditions
Q_{41}	$(V, M), (P, J), e_1, e_2, e_6$
Q_{43}	$(P, M), (P, J), e_1, e_4, e_6$
Q_{45}	$(P, M), (P, J), e_1, e_5, e_6$

Note that Algorithm 1 may produce redundant QPs. For instance, consider the schema graph in Fig.8(a), where photo and item are entities, Fig.8(b) is the XML document conforming to Fig.8(a). Assume $C = 3$, for the ATP query $Q : *[\sim$"Mike"$][\sim\!> * \sim$"John"$]$ searching for entity instances that contain "Mike" as their attribute value and have the related relationship with

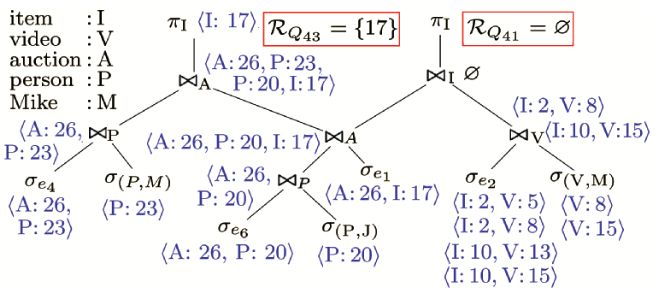

item : I
video : V
auction : A
person : P
Mike : M
π_I ⟨I: 17⟩
$\mathcal{R}_{Q_{43}} = \{17\}$
π_I
$\mathcal{R}_{Q_{41}} = \varnothing$
⟨A: 26, P: 23, P: 20, I: 17⟩
$\bowtie_A$
$\bowtie_I$ ∅
⟨I: 2, V: 8⟩
$\bowtie_V$ ⟨I: 10, V: 15⟩
⟨A: 26, P: 23⟩ $\bowtie_P$
⟨A: 26, P: 20, I: 17⟩ $\bowtie_A$
σ_{e_4} ⟨A: 26, P: 23⟩
$\sigma_{(P,M)}$ ⟨P: 23⟩
⟨A: 26, P: 20⟩ $\bowtie_P$
σ_{e_1} ⟨A: 26, I: 17⟩
σ_{e_6} ⟨A: 26, P: 20⟩
$\sigma_{(P,J)}$ ⟨P: 20⟩
σ_{e_2} ⟨I: 2, V: 5⟩ ⟨I: 2, V: 8⟩ ⟨I: 10, V: 13⟩ ⟨I: 10, V: 15⟩
$\sigma_{(V,M)}$ ⟨V: 8⟩ ⟨V: 15⟩

Fig.7. Executing of Q_{41} and Q_{43} using Algorithm 2.

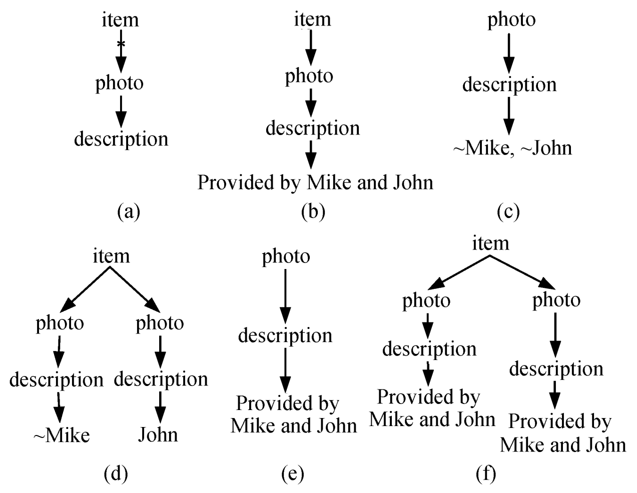


Fig.8. Illustration of redundant QPs.

entity instances that contain "John". Figs. 8(c)~8(d) are two QPs according to Algorithm 1. Obviously, Figs. 8(e)~8(f) are two matches of Figs. 8(c)~8(d), respectively. In fact, the QP in Fig.8(d) is redundant since for the corresponding result shown in Fig.8(e), each description node contains both "Mike" and "John", and both the two description nodes represent the same data element in the XML document in Fig.8(b). Therefore before evaluating each QP Q' in $\mathcal{Q}$, we need to check in lines 2~16 of Algorithm 2 for each REN E_K in all QPs of $\mathcal{Q}$, whether there exists entity instances of label E such that each entity instance d contains all keywords of K and no other REN $E'_{K'}$ exists such that $k \in (K' - K)$ and d contains k. Thus we have Theorem 3.

Theorem 3 (No Redundancy). *For a QP Q' evaluated in lines* 20~22 *of Algorithm* 2*, no other QP can produce the same matches as produced by Q'.*

Theorem 3 guarantees that Algorithm 2 needs no operations to evaluate redundant QPs of the given ATP query Q. Let $\mathcal{Q}$ be the set of QPs of Q, S_σ the set of distinct selection operations of $\mathcal{Q}$, L the average number of elements in an input stream which may be either the result set of a distinct selection operation or a distinct tag index, n the number of query nodes that must be processed using structural join operation, rather than *EPI* and *PPI*. Since $N^2F^C|Q| \ll L$ (the cost of Algorithm 1) in practice, the worst space and time complexity of Algorithm 2 is $O((|S_\sigma| + n)L)$.

5 Experimental Evaluation

5.1 Experimental Setup

Our experiments were implemented on a PC with Pentium4 2.8 GHz CPU, 2 G memory, 160 GB IDE hard disk, and Windows XP professional as the operating system.

The algorithms used for comparison include TwigStack, IM (i.e., indexMerge), MLCA[15] and XSEarch[9]. All algorithms were implemented using Microsoft VC++ 6.0. We select eXist[6], X-Hive[7] and MonetDB[8] to make comparison with our method, the detailed information of these systems can be found from their web sites, respectively.

5.2 Datasets, Indices and Queries

We use XMark[9], DBLP[10] and SIDMOD[11] (short for SIGMOD Record) datasets for our experiments. The main characteristics of the three datasets can be found in Table 6.

Table 6. Statistics of Datasets

Dataset	Size (MB)	Nodes (MB)	Max L	Avg. L	Index/Doc.
DBLP	130	3.3	6	2.9	4.9
XMark	115	1.7	12	5.5	5.3
SIGMOD	0.5	0.01	6	5.1	4.8

Note: L denotes length.

The last column of Table 6 is *the ratio of index size to document size*, where index consists of three parts, 1) PPI, 2) EPI and 3) assistant index used to get self entity nodes, of which PPI has much larger size than the other two.

The node category is specified using the two-step method discussed in Section 2. In the second step, node category is adjusted so as to make the results more accurate.

We select three kinds of queries for comparison: 1) nine ATP queries with related edge (Table 7), 2) six

[6]http://exist.sourceforge.net
[7]http://www.x-hive.com
[8]http://monetdb.cwi.nl/projects/monetdb/XQuery/index.html
[9]http://monetdb.cwi.nl/xml
[10]http://www.cs.washington.edu/research/xmldatasets/www/repository.html
[11]http://www.sigmod.org/record/xml/

TPs from XMark and DBLP datasets (Table 8), 3) 80 keyword queries, which are classified into four groups containing two, three four and five keywords, respectively. The keyword queries are omitted for limited space.

Table 7. ATP Aueries Used in Our Experiment

Query	Dataset	Query Expression	No. QPs
RX1	XMark	//person[name="Cong"]∼> item/name	8
RX2	XMark	//person[name="Cong"]∼> open_auction	4
RX3	XMark	//person[name="Cong"]∼> closed_auction	3
RX4	XMark	//open_auction∼>item/name	1
RX5	XMark	//item[name="great"]∼>category	1
RD1	DBLP	//author[name="Alberto"]∼> book/title	1
RD2	DBLP	//editor[name="Ronny"]∼> proceedings/ year	1
RS1	SIGMOD	//author[name="Richard"]∼> book/title	1
RS2	SIGMOD	//article[title="Quest"]∼> author/	1

Table 8. Tree Pattern Queries Used in Our Experiment

Query	Dataset	Query Expression
QX1	XMark	/site/regions/africa/item[/name="condemn"]/description/parlist/listitem/text/keyword
QX2	XMark	/site/closed_auctions/closed_auction [/type = "Featured"] [annotation/description[parlist/listitem/text[keyword [bold]]]]/price
QX3	XMark	/site/closed_auctions//emph
QX4	XMark	/site/people/person[/city="Birmingham"] /name
QD1	DBLP	//inproceedings[/author][/year]
QD2	DBLP	//www[/editor]/url

In our experiment, we assume that each MEW has at most three entity nodes, i.e., $C = 3$ for Algorithm 2. Note that $C = 3$ means that the maximal length of MEW for XMark is 12, which is large enough to find most meaningful relationships. As the demo and optimization techniques of [10] are not publicly available, we do not make comparison with it.

The last column of Table 7 is the number of QPs of the ATP query in the first column according to C and the DTD schema of the three datasets. As DBLP and SIGMOD datasets do not contain edges of ID, IDREF or IDREFS type, we take them as the ones that do not contain referential edges, thus each ATP query of the two datasets corresponds to one QP. For XMark dataset, however, although all ATP queries in Table 7 contain only *one* related edge, they may correspond to multiple QPs. As shown in Fig.9, even if for ATP queries with just one related edge, existing methods cannot work well, let alone for ATP queries containing more than one related edge. We show the running time of ATP queries with more than one related edge using keyword queries, where keywords in each keyword query are connected using related edges.

5.3 Evaluation Metrics

We consider the following performance metrics to compare the performance of different methods: 1) running time, 2) precision, 3) recall and 4) scalability.

As an ATP query may correspond to multiple QPs, for example, query RX2 has four QPs, the running time of RX2 equals the time of executing RX2 using the following XQuery expression, instead of the sum of running time of all its QPs. In such a way, existing system can make full use of their optimization methods to achieve better performance.

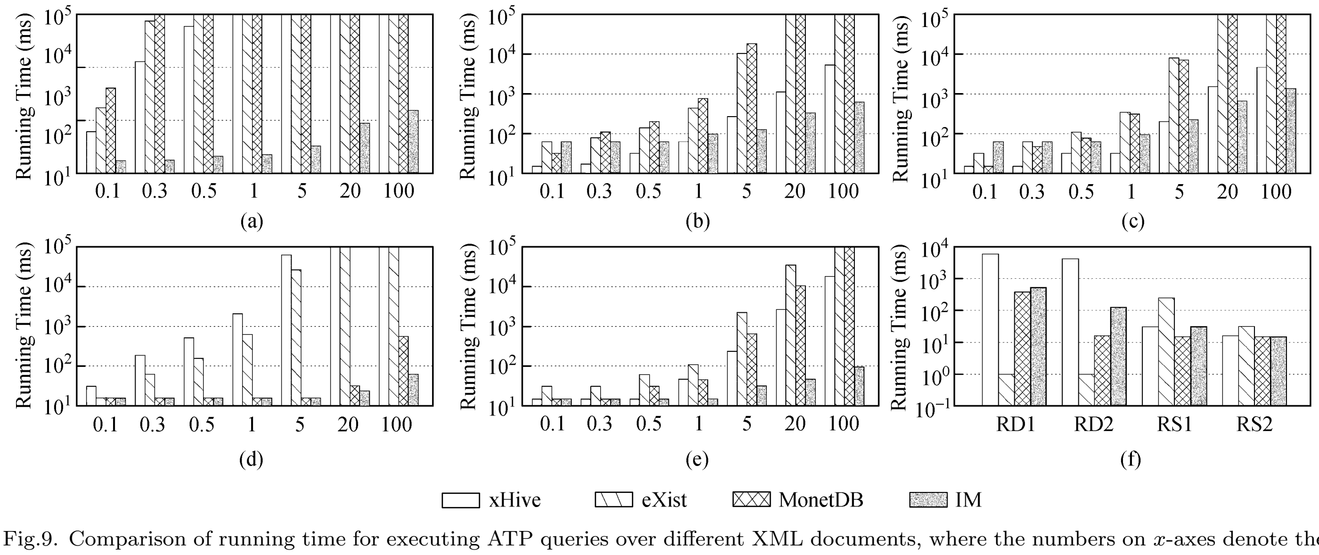


Fig.9. Comparison of running time for executing ATP queries over different XML documents, where the numbers on x-axes denote the size (MB) of XML documents. (a) RX1. (b) RX2. (c) RX3. (d) RX4. (e) RX5. (f) Other ATP queries.

```
for    $p in doc()//person, $o in doc()//open_auction
where  $p/name="Cong" and ($p/@id=$o/seller/@person
       or $p/@id=$o/annotation/author/@person
       or $p/@id=$o/bidder/personref/@person
       or $o/@id=$p/watches/watch/@open_auction)
return $o/@id
```

If all query nodes are connected together by related edges in an ATP query Q, Q is in essence a keyword query, therefore we compare the precision and recall of our method with those of MLCA[15] and XSEarch[9]. We define the two metrics using the following steps: 1) users submit their keyword queries; 2) by asking users' search intention, we formulate the XQuery expressions corresponding to their keyword queries, then let them select the XQuery expressions that meet their search intention. For a given keyword query Q, the result of the selected XQuery expressions is denoted as R; 3) evaluate all keyword queries using different methods, the result of a specific method on Q is denoted as R_Q. Then the precision and recall of this method are defined as: Precision $= (R_Q \cap R)/R_Q$, Recall$= (R_Q \cap R)/R$.

5.4 Performance Comparison and Analysis

5.4.1 ATP Queries

For all ATP queries of XMark dataset, since the performances of existing methods vary largely according to the given query, in Fig.9, we show the running time of these queries separately. Fig.9(a) shows the running time of RX1, from this figure we know existing methods cannot work well for this query, since RX1 has eight QPs and each QP involves three value join operations using XQuery expression. xHive, eXist and MonetDB even cannot work well for XMark dataset of size more than 300 k. Both RX4 and RX5 correspond to one QP which involves only one value join operation, as shown in Figs. 9(d)$\sim$9(e). MonetDB works very well, even though, our method beats MonetDB for large XML document, e.g., RX5. All other ATP queries correspond to one QP for DBLP and SIGMOD datasets, the result is shown in Fig.9(f), which shows that our method can also work well for the remaining queries.

Note that in Fig.9(f), there is not running time for eXist since in our experiment, DBLP dataset of size 130 M cannot be imported into eXist system. Moreover, MonetDB and eXist will break down when executing RX1 over XMark dataset with size of only 500 k in our experiment. In all these figures, the running time of 100 000 ms means that it is more than 100 000 ms or the method cannot process the query.

From Fig.9 we know our method outperforms existing methods in most cases, especially for queries with complex conditions, e.g., RX1 to RX3. For the other queries of Table 7, our method also works very well, because our method transforms all structural join operations into a set of simple selection operations.

5.4.2 Tree Pattern Queries

Fig.10 shows the comparison of running time for tree pattern queries. We omit the running time of eXist in Fig.10 since in our experiment, XMark dataset (115 MB) and DBLP dataset (130 MB) cannot be imported into eXist system.

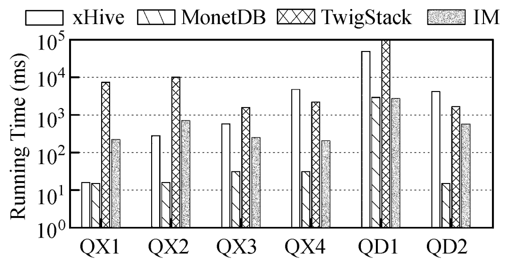


Fig.10. Comparison of running time for executing tree pattern queries on XMark (115 MB).

From this figure we know for tree pattern queries, our method only works a little better than MonetDB for QD1. This is because both QX1 and QX2 contain several entity nodes, thus need to fetch large amount of elements and consume CPU cost to do value join operations, QX3 contains "//" edge and cannot be equivalently transformed into selection operations, thus QX3 cannot avoid structural join operations. For QX4, even our method can avoid structural join operations, it still needs more time than MonetDB. Thus for tree pattern queries, MonetDB has the best performance for most queries.

Fig.11 shows the scalability of our method for RX1 and QX4. As XMark dataset of size more than 1GB cannot be imported into MonetDB, xHive and eXist, we do not show their running time in this figure. From Fig.11 we know that our method achieves better scalability.

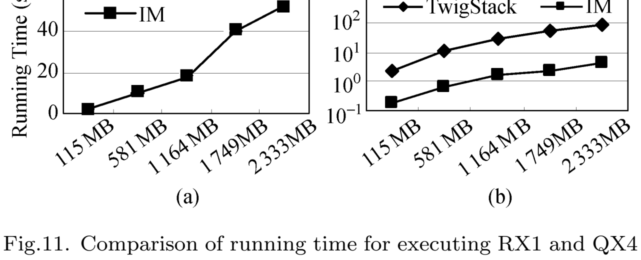


Fig.11. Comparison of running time for executing RX1 and QX4 over XML documents of different sizes. (a) RX1. (b) QX4.

5.4.3 Keyword Queries

Fig.12 shows the precision and recall of different methods for keyword query, from which we know for

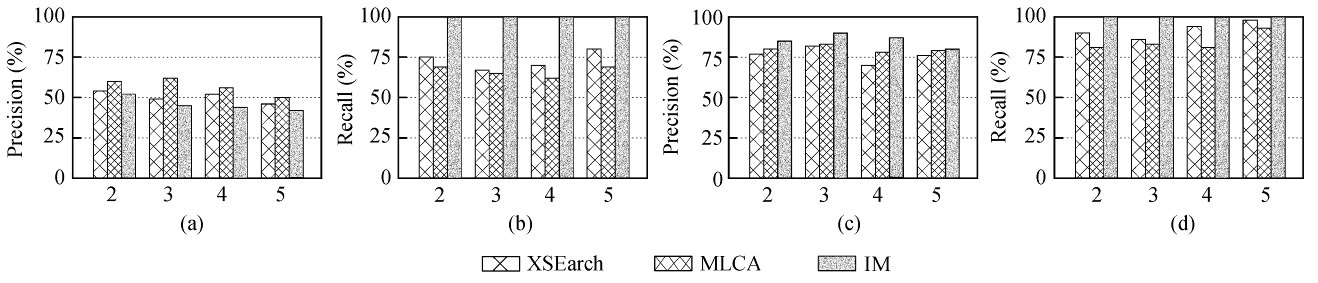


Fig.12. Result quality: average precision and recall for keyword queries, where numbers on x-axes denote the number of keywords for each group of queries. (a) Precision (XMark). (b) Recall (XMark). (c) Precision (SIGMOD record). (d) Recall (SIGMOD record).

for XMark dataset, the Recall of our method is 100%, this is because all results that meet users' search intention are returned by our method. However, the Precision is a little worse than MLCA and XSEarch, this is because our method may return results involving IDREF. If the users' search intention involves IDREF, obviously, our method will be more effective; otherwise, MLCA has the highest Precision. The average figures in Figs. 12(a)∼12(b) show that for XMark dataset, though the Precision of our method is not better than those of MLCA and XSEarch, it has the highest Recall. For SIGMOD dataset, as shown in Fig.12(c), both Precision and Recall of our method are better than those of MLCA and XSEarch, since in such a case IDREF is not considered, thus the number of QP is very small, usually equals 1, and our method always returns results that meet the users' search intention. Moreover, as shown in Fig.12(d), the Recall of our method is better than MLCA and XSEarch. The Precision and Recall on DBLP dataset is similar to that of SIGMOD dataset, we omit it for limited space.

We further employed F-measure as a metric, where $F = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$, to compare the effectiveness of different algorithms. As shown in Fig.13, we can see that our method beats the other algorithms and achieves the best F-measure. For example, on SIGMOD Record, F-measure of IM reaches 94.7% for queries of three keywords, while those of the other ones are about 83%. On XMark, although recall of IM is much better than those of XSEarch and MLCA, precision of IM is a little worse than those of XSEarch and MLCA, which results in that F-measure of IM is just a little better than those of XSEarch and MLCA. Even though, it is still the best one according to Fig.13.

As our method can use structural constraints in keyword queries to prune irrelevant answers, obviously, the Precision of our method can be improved if structural constraints are considered. However, this cannot be easily justified as we do not know in which case users will use structural constraints in their query expressions. Therefore, the precision and recall about the ATP queries in Table 7 are not presented.

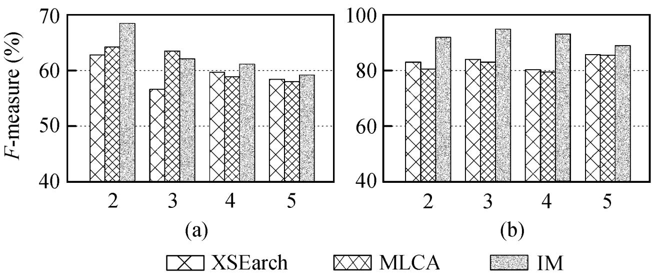


Fig.13. Comparison of the average of F-measure, where numbers on x-axis denote the number of keywords for each group of queries.

Fig.14 shows the comparison of average running time, from which we know that our method is not efficient as MLCA, because MLCA is based on tree model and simply returns results based on structural information, which will cause lower Recall, as shown in Fig.12. By affording additional time, though slower than MLCA, our method can achieve both high Recall and Precision.

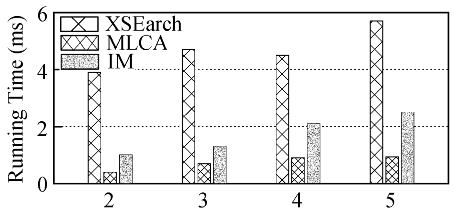


Fig.14. Average running time on XMark (115 MB), where numbers on x-axis denote the number of keywords for each group of queries.

In summary, our method is effective when compared with existing keyword search methods since it considers IDREF and returns all meaningful results. Moreover, our method is efficient when compared with existing query engines, especially when the given queries contain related edges.

6 Discussion and Related Work

Structured Query Methods. As the core operation of processing an XQuery expression is finding the matched results of TPs embedded in it, thus many algorithms[2-4,6] focused on efficiently processing a given TP query. Among them, TwigStack[2] was the first one that guarantees that the CPU time and I/O complexity are independent of the size of partial matches to any root-to-leaf query path when considering only A-D edges. [3-4, 6] made improvements against TwigStack. All these methods require that users understand the schema.

Keyword Search Methods. Among existing keyword search methods, the basic semantics of [7-9, 11, 13-15, 17-23] are based on *tree* model and cannot capture the meaningful relationships conveyed by IDREF. On the contrary, the keyword search semantics of [10, 12, 16] are based on *graph* model (IDREF considered), thus can find more answers, where each answer T is an acyclic subgraph of the given XML document D, T contains all keywords of the given keyword query Q, while any proper subgraph of T does not.

According to the query processing mechanism, *graph* model based methods[10,12,16] can be classified into two groups. The first kind of method[34] directly computes all answers from the given XML document D. However, finding even the minimal answer (acyclic subgraph of D) is reducible to the classical NP-complete group Steiner tree problem[35]. Thus this method[34] applies special constraints and finds only a subset of all answers.

The second group of methods[10,12,16] use a two-step strategy: 1) compute all QPs of the given keyword query, 2) evaluate all QPs to get the final results. When the schema becomes complex and when evaluating large amount of QPs, both the two steps are no longer a trivial task.

Step 1. The methods proposed in [12, 16] focus on finding all QPs of schema elements rather than keywords and each QP is a subgraph of the given schema graph G. Thus they cannot identify QPs of a keyword query where several keywords correspond to the same schema element, e.g., {person:Mike, person:John}. [10] proposes a method to compute from the schema graph all QPs of keyword queries that allow both text values and schema elements. The main idea is as follows.

For a given keyword query $Q = \{k_1, k_2, \ldots, k_m\}$, [10] maintains for each keyword k an inverted list which stores the data elements directly containing k and works through the following steps. 1) For each keyword $k_i \in Q$, produce the element set S_{k_i} which consists of all data elements containing k_i. 2) Based on the m sets $S_{k_1}, S_{k_2}, \ldots, S_{k_m}$, produce data element set S_K for all subsets K of Q, where $S_K = \{d|d \in \cup_{k\in K} S_k \wedge \forall k \in K, d\ \mathit{contains}\ k \wedge \forall k \in Q - K, d\ \mathit{does\ not\ contain}\ k\}$[28]. 3) For all elements of S_K, find the set of corresponding schema elements $S_{label(S_K)} = \{l|\exists d \in S_K,\ l = label(d)\}$. For each $l \in S_{label(S_K)}$, add a node named l to the original schema graph by attaching K to l to produce a new schema graph G'. 4) Find from G' all the QPs, where each QP q is a subgraph of G' and contains all keywords at least once, while any proper subgraph of q does not. Thus the performance of the first step is affected by three factors: 1) I/O cost of accessing all data instances to construct G' in the first two steps, 2) the maximum size of QP, 3) the size of the new expanded graph G'.

Step 2. As discussed in the first paragraph of this section, to evaluate all QPs of the given ATP query, existing methods[2-4,6] suffer from costly structural join operation, which greatly affects the whole performance.

There are still many other related work in this area, interested readers can find them from [36-37].

Relaxed-Structure Methods. By combining keyword search semantics and structured query together, the methods proposed in [15-16, 24-26] provide users with a relaxed-structure query mechanism and return approximate answers. However, keyword search semantics of the these methods may result in losing meaningful results and, as discussed in Section 1, all these methods cannot really free users from the great burden of understanding the schema.

Other Related Work. The relaxation method of [38] requires that users generate an initial TP, which is then converted into less restrictive TPs based on a set of relaxation rules. If the initial TP is wildly inaccurate, many results will likely be inaccurate.

[39] extended XPath by a new notion, ClosestAxis, thus users can get closest, rather than definitely related data nodes. [40] proposes to extend XQuery with window functions over an input sequence to support continuous query.

For keyword search methods or relaxation methods, ranking schemes can be used to rank answers. XSEarch[9] ranks answers by considering distance, term frequency, document frequency, etc. XRank[8] extends the Page-rank hyperlink metric to XML. [10] ranks query results according to the size of each answer. All these ranking schemes are orthogonal to retrieval and can be combined with our methods to provide a more effective search mechanism.

7 Conclusions

Considering that existing query mechanisms cannot work well when confronting complex and heterogeneous XML documents, we proposed a new semantics, related relationship, to capture the meaningful relationships of data elements, then made an extension to XPath by incorporating the related relationship into existing XPath grammar, so as to provide users with a query mechanism that can be used to query desired information from complex and heterogeneous XML documents in an *effective* and *flexible* way. We proposed a new schema summary, i.e., entity graph, and two indices to improve the performance of query processing. Further, we proved that our algorithm is not only effective (completeness and no redundancy), but also efficient (the costly structural join operations can be equivalently transformed into just a few selection and value join operations in most cases). The experimental results verify the effectiveness and efficiency of our method in terms of various evaluation metrics.

References

[1] Christophides V, Cluet S, Simèon S. On wrapping query languages and efficient XML integration. In *Proc. the 2000 ACM SIGMOD International Conference on Management of Data* (*SIGMOD2000*), Dallas, USA, May 14-19, 2000, pp.141-152.

[2] Bruno N, Koudas N, Srivastava D. Holistic twig joins: Optimal XML pattern matching. In *Proc. the 2002 ACM SIGMOD International Conference on Management of Data* (*SIGMOD2002*), Madison, USA, June 3-6, 2002, pp.310-321.

[3] Jiang H, Wang W, Lu H, Yu J X. Holistic twig joins on indexed XML documents. In *Proc. the 29th International Conference on Very Large Data Bases* (*VLDB2003*), Berlin, Germany, Sept. 12-13, 2003, pp.273-284.

[4] Lu J, Ling T W, Chan C Y, Chen T. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In *Proc. the 31st International Conference on Very Large Data Bases* (*VLDB2005*), Trondheim, Norway, Aug. 30-Sept. 2, 2005, pp.193-204.

[5] Chen T, Lu J, Ling T W. On boosting holism in XML twig pattern matching using structural indexing techniques. In *Proc. the ACM SIGMOD International Conference on Management of Data* (*SIGMOD2005*), Baltimore, USA, June 13-16, 2005, pp.455-466.

[6] Chen S, Li H, Tatemura J *et al.* Twig2Stack: Bottom-up processing of generalized-tree-pattern queries over XML documents. In *Proc. the 32nd International Conference on Very Large Data Bases* (*VLDB2006*), Seoul, Korea, Sept. 12-15, 2006, pp.283-294.

[7] Xu Y, Papakonstantinou Y. Efficient keyword search for smallest LCAs in XML databases. In *Proc. the ACM SIGMOD International Conference on Management of Data* (*SIGMOD2005*), Baltimore, USA, June 13-16, 2005, pp.527-538.

[8] Guo L, Shao F, Botev C, Shanmugasunda J. XRANK: Ranked keyword search over XML documents. In *Proc. the 2003 ACM SIGMOD International Conference on Management of Data* (*SIGMOD2003*), San Diego, USA, June 9-12, 2003, pp.16-27.

[9] Cohen S, Mamou J, Kanza Y, Sagiv Y. XSEarch: A semantic search engine for XML. In *Proc. the 29th International Conference on Very Large Data Bases* (*VLDB2003*), Berlin, Germany, Sept. 12-13, 2003, pp.45-56.

[10] Hristidis V, Papakonstantinou Y, Balmin A. Keyword proximity search on XML graphs. In *Proc. the 19th International Conference on Data Engineering* (*ICDE2003*), Bangalor, India, March 5-8, 2003, pp.367-378.

[11] Liu Z, Chen Y. Reasoning and identifying relevant matches for XML keyword search. In *Proc. the VLDB Endowment*, Aug. 2008, 1(1): 921-932.

[12] Cohen S, Kanza Y, Kimelfeld B, Sagir Y. Interconnection semantics for keyword search in XML. In *Proc. the 14th ACM CIKM International Conference on Information and Knowledge Management* (*CIKM2005*), Bremen, Germany, Oct. 31-Nov. 5, 2005, pp.389-396.

[13] Liu Z, Chen Y. Identifying meaningful return information for XML keyword search. In *Proc. International Conference on Management of Data* (*SIGMOD2007*), Beijing, China, June 12-14, 2007, pp.329-340.

[14] Li G, Feng J, Wang J, Zhou L. Effective keyword search for valuable LCAs over XML documents. In *Proc. the 6th ACM Conf. Information and Knowledge Management* (*CIKM2007*), Lisbon, Portugal, Nov. 6-9, 2007, pp.31-40.

[15] Li Y, Yu C, Jagadish H V. Schema-free XQuery. In *Proc. the 30th International Conf. Very Large Data Bases* (*VLDB2004*), Toronto, Canada, Aug. 29-Sept. 3, 2004, pp.72-83.

[16] Yu C, Jagadish H V. Querying complex structured databases. In *Proc. the 33rd International Conf. Very Large Data Bases* (*VLDB2007*), Vienna, Austria, Sept. 23-28, 2007, pp.1010-1021.

[17] Chen L J, Papakonstantinou Y. Supporting top-K keyword search in XML databases. In *Proc. the 26th International Conference on Data Engineering* (*ICDE2010*), Long Beach, USA, March 1-6, 2010, pp.689-700.

[18] Zhou R, Liu C, Li J. Fast ELCA computation for keyword queries on XML data. In *Proc. the 13th International Conference on Extending Database Technology* (*EDBT2010*), Lausanne, Switzerland, March 22-26, 2010, pp.549-560.

[19] Kong L, Gilleron R, Lemay A. Retrieving meaningful relaxed tightest fragments for XML keyword search. In *Proc. the 12th International Conference on Extending Database Technology* (*EDBT2009*), Saint-Petersburg, Russia, Mar. 23-26, 2009, pp.815-826.

[20] Bao Z, Ling T W, Chen B, Lu J. Effective XML keyword search with relevance oriented ranking. In *Proc. the 25th International Conference on Data Engineering* (*ICDE2009*), Shanghai, China, March 29-April 2, 2009, pp.517-528.

[21] Li J, Liu C, Zhou R, Wang W. Suggestion of promising result types for XML keyword search. In *Proc. the 13th International Conference on Extending Database Technology* (*EDBT2010*), Lausanne, Switzerland, Mar. 22-26, 2010, pp.561-572.

[22] Li G, Li C, Feng J, Zhou L. SAIL: Structure-aware indexing for effective and progressive top-k keyword search over XML documents. *Inf. Sci.*, 2009, 179(21): 3745-3762.

[23] Feng J, Li G, Wang J, Zhou L. Finding and ranking compact connected trees for effective keyword proximity search in XML documents. *Inf. Syst.*, 2010, 35(2): 186-203.

[24] Trotman A, Sigurbjörnsson B. NEXI, now and next. In *Proc. the 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval* (*INEX2004*), Dagstuhl Castle, Germany, Dec. 6-8, 2004, pp.41-53.

[25] Fuhr N, Groβjohann K. XIRQL: A query language for information retrieval in XML documents. In *Proc. the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR2001*), New Orleans, USA, Sept. 9-21, 2001, pp.172-180.

[26] Theobald M, Schenkel R, Weikum G. An efficient and versatile query engine for Topx search. In *Proc. the 31st International Conference on Very Large Data Bases* (*VLDB2005*), Trondheim, Norway, Aug. 30-Sept. 2, 2005, pp.625-636.

[27] Yu C, Jagadish H V. Schema summarization. In *Proc. the 32nd International Conference on Very Large Data Bases* (*VLDB2006*), Seoul, Korea, Sept. 12-15, 2006, pp.319-330.

[28] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword search in relational databases. In *Proc. the 28th International Conference on Very Large Data Bases* (*VLDB2002*), Hong Kong, China, Aug. 20-23, 2002, pp.670-681.

[29] Pal S, Cseri I, Seeliger O, Schaller G, Giakoumakis L, Zolotov V. Indexing XML data stored in a relational database. In *Proc. the 30th International Conference on Very Large Data Bases* (*VLDB2004*), Toronto, Canada, Aug. 29-Sept. 3, 2004, pp.1134-1145.

[30] Bex G J, Neven F, Vansummeren S. Inferring XML schema definitions from XML data. In *Proc. the 33rd International Conference on Very Large Data Bases* (*VLDB2007*), Vienna, Austria, Sept. 23-28, 2007, pp.998-1009.

[31] Bex G J, Neven F, Schwentick T, Tuyls K. Inference of concise DTDs from XML data. In *Proc. the 32nd International Conference on Very Large Data Bases* (*VLDB2006*), Seoul, Korea, Sept. 12-15, 2006, pp.115-126.

[32] Bernstein P A, Melnik S, Mork P. Interactive schema translation with instance-level mappings. In *Proc. the 31st International Conference on Very Large Data Bases* (*VLDB2005*), Trondheim, Norway, Aug. 30-Sept. 2, 2005, pp.1283-1286.

[33] Vries A P, Vercoustre A M, Thom J A, Craswell N, Lalmas M. Overview of the INEX 2007 entity ranking track. In *Proc. the 6th International Workshop of the Initiative for the Evaluation of XML Retrieval* (*INEX2007*), Germany: Springer, 2007, pp.245-251.

[34] Golenberg K, Kimelfeld B, Sagir Y. Keyword proximity search in complex data graphs. In *Proc. the ACM SIGMOD International Conference on Management of Data* (*SIGMOD2008*), Vancouver, Canada, June 10-12, 2008, pp.927-940.

[35] Reich G, Widmayer P. Beyond Steiner's problem: A VLSI oriented generalization. In *Proc. the 15th International Workshop on Graph-Theoretic Concepts in Computer Science* (*WG1989*), Castle Rolduc, Netherlands, June 14-16, 1989, pp.196-210.

[36] Yu J X, Qin L, Chang L. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, 2010, 33(1): 67-78.

[37] Chen Y, Wang W, Liu Z, Lin X. Keyword search on structured and semi-structured data. In *Proc. the International Conference on Management of Data* (*SIGMOD2009*), Providence, USA, June 29-July 2, 2009, pp.1005-1010.

[38] Amer-Yahiq S, Kondas N, Marian A, Srivastava D, Toman D. Structure and content scoring for XML. In *Proc. the 31st International Conference on Very Large Data Bases* (*VLDB2005*), Trondheim, Norway, Aug. 30-Sept. 2, 2005, pp.361-372.

[39] Zhang S, Dyreson C. Symmetrically exploiting XML. In *Proc. the 15th International Conference on World Wide Web* (*WWW2006*), Edinburgh, UK, May 22-26, 2006, pp.103-111.

[40] Botan I, Fischer P M, Florescu D, Kossmann D, Kraska T, Tamosevicius R. Extending XQuery with window functions. In *Proc. the 33rd International Conference on Very Large Data Bases* (*VLDB2007*), Vienna, Austria, Sept. 23-28, 2007, pp.75-86.

Jun-Feng Zhou received the Ph.D. degree in computer science from Renmin University of China. He is an associate professor of Yanshan University and a member of the China Computer Federation. His research interests include XML structured query processing and XML keyword search.

Tok Wang Ling received his Ph.D. degree in computer science from the University of Waterloo, Canada. He is a professor of the Department of Computer Science, School of Computing at the National University of Singapore. His research interests include data modeling, ER approach, normalization theory, semi-structured data model, XML twig pattern query processing, and XML keyword query processing. He has published more than 190 papers, co-authored a book, co-edited a book, and co-edited nine conference proceedings. He is an ACM Distinguished Scientist, a senior member of IEEE and Singapore Computer Society, and an ER Fellow.

Zhi-Feng Bao is a research fellow in School of Computing, National University of Singapore. He received his Ph.D. degree from the Department of Computer Science, School of Computing, National University of Singapore. His research interests include XML structured query processing, XML keyword search and provenance data management.

Xiao-Feng Meng received the B.S. degree from Hebei University, M.S. degree from Renmin University of China, and Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, all in computer science. He is currently a professor of School of Information in Renmin University of China. His research interests include web data integration, native XML databases, mobile data management, flash based databases. He is the secretary general of Database Society of the China Computer Federation (CCF DBS). He has published more than 100 technical papers.