MCN: A New Semantics towards Effective XML Keyword Search

Junfeng Zhou[1,2], Zhifeng Bao[3], Tok Wang Ling[3], and Xiaofeng Meng[1]

[1]School of Information, Renmin University of China
[2]Department of Computer Science and Technology, Yanshan University
{zhoujf2006,xfmeng}@ruc.edu.cn
[3]School of Computing, National University of Singapore
{baozhife,lingtw}@comp.nus.edu.sg

Abstract. We investigate the expressiveness of existing XML keyword search semantics and propose *Meaningful Connected Network (MCN)* as a new semantics to capture meaningful relationships of the given keywords from XML documents. Our evaluation method adopts a two-step strategy to compute all MCNs. In the first step, we identify a set of query patterns from a new schema summary; in the second step, all query patterns are processed based on two efficient indices, partial path index and entity path index. The experimental results show that our method is both effective and efficient.

1 Introduction

As an effective search method to retrieve useful information, keyword search has gotten a great success in IR field. However, the inherently hierarchical structure of XML data makes the task of retrieving the desired information from XML data more challenging than that from flat documents. An XML document can be modeled as either a rooted tree or a directed graph (if IDRefs are considered). For example, Fig. 1 shows an XML document D, where solid arrows denote the containment edge, dashed arrows denote the reference edge, and the number beside each node denotes the node id.

The critical issue of XML keyword search is how to find meaningful query results. In both tree model and graph model, the main idea of existing approaches is to find a set of *Connected Network*s (CNs) where each CN is an acyclic subgraph T of D, T contains all the given keywords while any proper subgraph of T does not. In particular, in tree data model, Lowest Common Ancestor (LCA) semantics [1] is first proposed, followed by SLCA (smallest LCA) [1] and MLCA [5] which apply additional constraints on LCA. In graph data model, methods proposed in [6–10] focused on finding matched CNs where IDRefs are considered.

In practice, however, most existing approaches [1–10] only take into account the structure information among the nodes in XML data, but neglect the *node categories*; thus they suffer from the *limited expressiveness*, which makes them fail to provide an effective mechanism to describe how each part in the returned data fragments are connected in a meaningful way, as shown in Example 1.

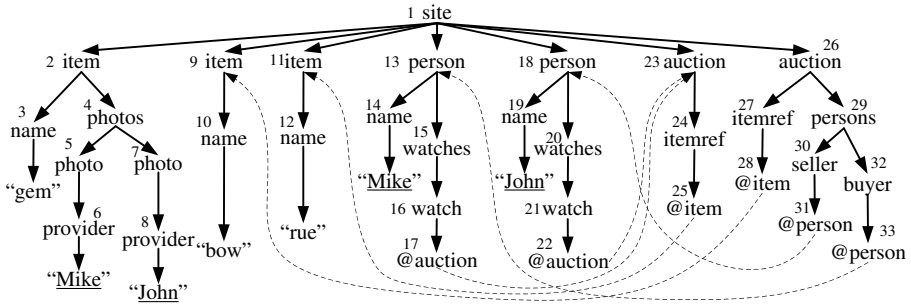

Fig. 1. An example XML document D

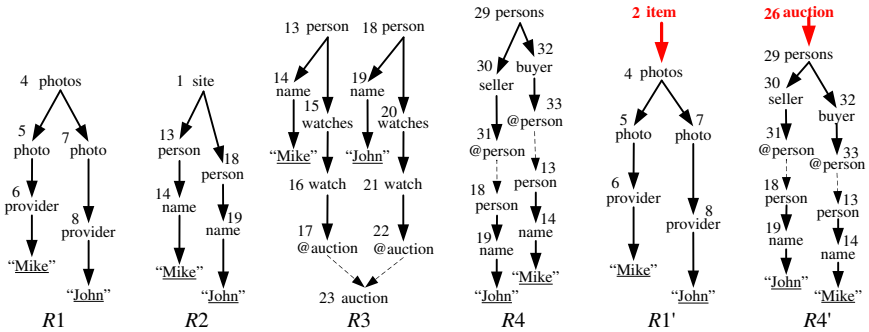

Fig. 2. $R1$ to $R4$ are four possible answers of existing methods for query Q={Mike, John} against D in Fig. 1. $R3$, $R1'$ and $R4'$ are three meaningful answers of our method.

Example 1. Consider a keyword query Q={Mike, John} issued on D in Fig. 1. If IDRefs in D are not considered, then $R1$ and $R2$ in Fig. 2 are two (not all) matched results according to the LCA semantics [1] where the LCA node are *photos* and *site*, respectively. If IDRefs in XML data are considered, we may find more results, e.g., $R1$ to $R4$ are four matched results of existing methods. However, we cannot identify any meaningful relationship between 'Mike' and 'John' from $R1$, $R2$ and $R4$, because the nodes (*photos, site, persons*) connecting 'Mike' and 'John' do not convey any useful information to explain the relationship between 'Mike' and 'John'. We observe when talking about relationships between data elements, users just care about relationships of those representative nodes (*photo, person, item* and *auction* in Fig. 1), which we call entities in ER model, and most of the time their query is based on the relationships of entities, rather than those meaningless ones (e.g., *photos*, *site* and *persons*), which are just used to organize data. With this observation, $R3$, $R1'$ and $R4'$ in Fig. 2 should be meaningful results. $R3$ means 'Mike' and 'John' watch the same *auction*; $R1'$ means both 'Mike' and 'John' provided a *photo* to the same *item*; $R4'$ means 'Mike' bought an *item* sold by 'John' in an *auction*. However, according to the semantics of existing works [1–10], $R1'$ and $R4'$ will be removed as answers because of $R1$ and $R4$, respectively.

Motivated by the above problem, we propose a new semantics called *Meaningful Connected Network (MCN)* to capture the *meaningful relationships* of the

given keywords from *XML document graph* by considering reference relationship. A MCN is an acyclic subgraph T of the given XML document D and contains all the keywords at least once, which is defined based on the relationships among entities (or entity instances), rather than simply on data elements without considering their categories. For the above query, we first check which entity instances 'Mike' ('John') belongs to, then find the meaningful relationships of the two entity instances of 'Mike' and 'John'. Finally, $R3$, $R1'$ and $R4'$ (not $R1$, $R2$ and $R4$) as MCNs are considered as meaningful results and returned.

However, finding even the first CN is reducible to the classical group Steiner tree problem, which is known to be NP-complete [12]. As a MCN may contain nodes that are redundant to a CN (e.g., item in $R1'$ is redundant to $R1$), finding all MCNs from the given XML document is more difficult than finding all CNs. Note all MCNs can be classified into different groups according to their structure, we call the structure of a MCN together with all query keywords a *query pattern* (QP) (e.g., $R3$, $R1'$ and $R4'$ are three QPs after removing all node id), thus finding all occurrences of MCNs equals to solving the following two problems.

- ($P1$) Efficiently identify all QPs from the underlying schema.
- ($P2$) Efficiently evaluate all QPs against the given XML data.

For problem $P1$, we propose to use *entity graph* as a schema summary to capture the meaningful relationships of entity nodes from the original schema. Entity graph is generated from the original schema by removing the noisy information while preserving the connection relationships of entity nodes. Then we propose an algorithm based on entity graph to efficiently compute all QPs.

For problem $P2$, we design two efficient indices to improve the query performance. The first is *partial path index*. For each keyword k, partial path index stores a set of paths, each of which starts with an entity instance and ends at one of its attribute node that directly contains k. The second is *entity path index*. Entity path index stores all the matching instances of entity pairs, where each pair is joined by one edge of the entity graph.

In summary, our contributions are listed as follows:

- We propose an effective semantics, i.e., Meaningful Connected Network (MCN), to enhance the expressiveness of keyword search query.
- We propose to use an entity graph to reduce the cost of computing QPs.
- We propose an algorithm based on partial path index and entity path index to improve the performance of query evaluation. We proved the costly structural join operations[1] can be avoided from the evaluation of *all* QPs.
- Extensive experiments are conducted on datasets of various characteristics, and the results show our method is both effective and efficient.

2 Background and Related Work

Schema: We assume the schema is always available, as we can use the methods proposed in [13, 14] to infer the schema (if unavailable). We use a node labeled directed graph to model a schema. Formally, $S = (V_S, E_S)$, where V_S denotes a

[1] Join operations based on ancestor-descendant or parent-child relationship.

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 to its child element. The second is the *reference edge*, which is drawn as a dashed arrow from the attribute of referrer element to referee element.

Node Categories: In the following discussion, whenever we mention *entity* and *attribute*, they refer to the notions defined in ER-model, rather than that defined in XML specification[2]. Generally speaking, two kinds of methods can be adopted to specify the category of each schema element, which are (1) automatic methods using heuristic inference rules [4, 6] and (2) manual method done by users, DBA or domain expert. The inference rules used in [4, 6] are as follows:

1. A node represents an entity if it corresponds to a $*$-node in the 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.

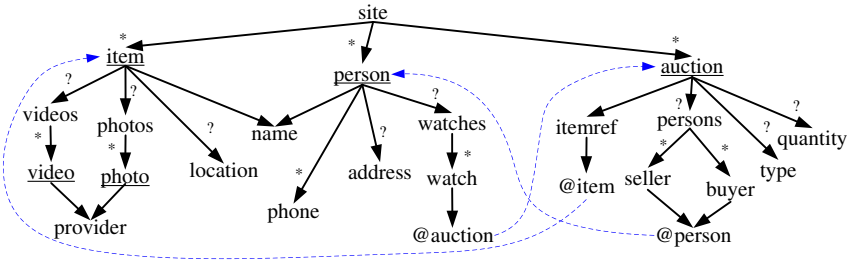

Fig. 3. An example schema $\boldsymbol{S}$ of XMark

The automatic method can avoid the cost of manual intervention, but it may not be quite correct. E.g., for the schema in Fig. 3, *person* is a $*$-node, thus by rule 1, *person* represents entity, instead of the attribute of *site*. *name* is considered as an attribute of *person* and *item* according to rule 2. *site* is not a $*$-node and has no value child, so it is a connection node by rule 3. However, according to the above rules, *phone* will be considered as entity, which is unreasonable, since *phone* is usually considered as a multi-value attribute of *person*. By using manual method from scratch, we can get accurate category of each schema node. However, it may impose great burden to users, DBA or domain experts.

Therefore, to achieve as accurate as possible node categories while paying minimum manual intervention, we first employ the above three inference rules to get an approximately accurate categorization, followed by a minor manual adjustment from users, DBA or domain expert. Using this method, we can consider *phone*, *watch*, *buyer* and *seller* as multi-value attributes of their entities, respectively. We take the underlined nodes as entities in Fig. 3.

To facilitate our discussion, whenever we use *entity*, it refers to the entity-type which corresponds to an entity node of a schema; the term *entity instance* is

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

used to denote the instance of *entity* in XML document. In this paper, we mainly focus on how to provide effective and efficient mechanism to extract meaningful results based on the results of existing classification methods. The notations used in our discussion are shown in Table 1.

Table 1. Summary of notations

Notation	Description	Notation	Description
MEW	meaningful entity walk	QP	query pattern
(M)CN	(meaningful) connected network	TP	tree (or twig) pattern
PPI	partial path index	EPI	entity path index

Discussion and Related Work: Among existing XML keyword search methods [1–10], The basic semantics [1–5] is based on the *tree* model, thus cannot capture the meaningful relationship conveyed by IDRefs.

For *graph* model (IDRefs considered) based methods [6–10], the first group [9, 10] directly compute all *CN*s from the given XML document. However, finding even the minimal connected network is reducible to the classical NP-complete group Steiner tree problem [12]. Thus these methods [9, 10] apply special constraints to CN and find only a subset of all CNs.

The second group [6–8] adopt a two-step strategy: (1) compute the set of QPs that are isomorphic to the set of CNs, (2) evaluate all QPs to get the matching results. However, when the schema graph becomes complex and when evaluating large amount of QPs, both the two steps are no longer a trivial task.

For step 1: the methods proposed in [6, 8] focus on finding all QPs of schema elements, text values are attached to different schema elements, thus they cannot process queries involving text values attached to two schema elements of same name, e.g., {person:Mike, person:John}. [7] proposed 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 to find all QPs from a new *expanded* graph G' that is generated from G and each QP is a subgraph of G'.

For a given keyword query $Q = \{k_1, k_2, ..., k_m\}$, [7] 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 $S_{k_1}, S_{k_2}, ..., S_{k_m}$, produce data element set S_K for all subsets K of Q, where $S_K = \{d | d \in \cup_{k \in K} S_k \wedge \forall k \in K, d \textit{ contains } k \wedge \forall k \in Q - K, d \textit{ does not contain } k\}$ [11]. (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', q 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, and (3) the size of the new expanded graph G'.

For step 2: as a keyword query may correspond to multiple QPs, and each QP consists of several tree (twig) patterns (*TPs*) connected together by reference edges, which imposes great challenges for subsequent query processing. Existing methods made improvements from the following four orthogonal aspects: (1)

designing efficient algorithms [15, 16], (2) reducing the size of the given QP [17], (3) designing efficient indices to reduce the size of input streams [15, 16], and (4) reusing the intermediate results to improve the whole performance. However, all these methods suffer from costly structural join operation, which greatly affects the whole performance.

3 Semantics of Keyword Search Queries

From D in Fig. 1, we know person "Mike" bought the item sold by person "John", which can be expressed as answer $R4'$ in Fig. 2. $R4'$ manifests the meaningful relationship between entity instances may contain edges of different directions (*mixed directions* problem) and cannot be got by simply traversing the document. Even if we omit the direction of each edge, it is infeasible to XML document because of its large size. An alternative way is finding such relationship from schema graph, which has much smaller size. However, some relationships produced in such case may be meaningless (*meaningfulness* problem). For example, R' : "item→name←person" is a possible relationship produced by traversing 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 of same element type name in XML documents, which contradicts R'. Thus, we need an effective way to capture both "*mixed directions*" and "*meaningfulness*" so as to avoid losing meaningful relationships (e.g., $R4'$) by traversing directed graph and producing meaningless relationships (e.g., R') 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 $W : v_0, v_1, \ldots, v_k$. Any $v_i - v_j$ walk $W' : v_i, e_{i+1}, v_{i+1}, \ldots, v_{j-1}, e_j, v_j (0 \leq i \leq j \leq k)$ 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 Definition 1 doesn't 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*

- *$L(W) \leq 1$, or,*
- *W doesn't 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 2. According to Definition 2, W_1*:*"person" is a MEW since person is an entity and $L(W_1) = 0 \leq 1$. W_2*:*"item→name←person" is not a MEW according to Definition 2, the reason is stated in the first paragraph of this section, where W_2 is denoted as R'. W_3*:*"person→watches→watch→@auction- -→auction" is a MEW according to Definition 2, which means a person is watching an auction. W_4*:*"person←- -@person←buyer←persons←auction→persons→seller →@person - -→person" is a MEW which means a person bought an item sold by another person. W_5*:*"item←site→person" is not a MEW according to Definition 2 as W_5 has the form "u←v→w" and site is not an entity, which means the relationship of item and person cannot be interpreted by a non-entity node, i.e., site.

Definition 3 (Meaningful Connected Network (MCN)). *Let* $Q = \{k_1, k_2, ..., k_m\}$ *be the given keyword query. A meaningful connected network of* Q *on the XML document* D *is a subgraph* T *of* D*, which holds all the following properties:*

1. T *contains* $k_i (1 \leq i \leq m)$ *at least once,*
2. *for any node* u *of* T*, if* u *is not an entity instance and joined by just one edge with other nodes of* T*,* u *contains at least one keyword,*
3. *for any two nodes* u *and* v *of* T*, if* u *and* v *are entity instances, there exists at least one* $u - v$ *MEW instance* W *on* T*,*
4. *no proper subgraph of* T *can hold for the above three properties.*

Example 3. Consider the six subgraphs in Fig. 2, where there are four entities, i.e., *person*, *photo*, *item* and *auction*. For $R1$, we cannot explain intuitively the relationships of the two *photo* nodes as they are connected together through a connection node, i.e., *photos*. Similarly, we cannot explain intuitively the relationships of the two *person* nodes in $R2$ and $R4$ as they are connected together through two connection nodes, i.e., *site* and *persons*. Since the walk "photo←photos→photo" in $R1$ is not a MEW (*photos* is not an entity node), according to Definition 3, $R1$ is not a MCN. $R2$ and $R4$ are not MCNs for the same reason. Thus they will be considered as meaningless answers. The intuitive meanings of $R3$, $R1'$ and $R4'$ are explained in Example 1, where each pair entity nodes in $R3$, $R1'$ and $R4'$ are connected through MEWs. According to Definition 3, $R3$, $R1'$ and $R4'$ are MCNs and considered as meaningful answers.

As shown in [11], the size of the joining sequence of two data elements in a given XML document is data bound (data bound means the size of a result may be as large as the number of nodes in an XML document), so is the MCN. Thus users are usually required to specify the maximum size C for all MCNs, which equals to the number of edges. However, such method may return results of very weak semantics or lose meaningful results. For example, each one of $R3$, $R1'$ and $R4'$ contains 3 entity instances, thus the semantic strongness of the three MCNs should be equal to each other, if all edges have the same weight. By specifying $C = 10$, $R3$ and $R4'$ will not be returned as matching results. On the other hand, a MCN may contain no connection node but an overwhelming number of entity instances, if its size equals to C, it may convey very weak semantics. Therefore in our method, the constraints imposed on MCN is not the maximum number of edges, but that of entity instances. This C is a user-specified variable with a default value of 3. As a result, a formal definition of keyword search is as below.

Definition 4 (Keyword Search Problem). *For a given keyword query Q, find all matched MCNs from the given XML document D, where each MCN contains at most C entity instances.*

4 Computation of Query Patterns

Definition 5 (Entity Path). *Path $p : v_1, v_2, ..., v_k (2 \leq k)$ of schema graph S is called an entity path if only v_1 and v_k are entity nodes, and for any $v_i (2 \leq i \leq k-1), v_i \neq v_j (1 \leq j \leq k \wedge i \neq j)$.*

Definition 6 (Partial Path). *A partial path p is a path of the given QP Q, which starts with an entity node n 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. As the definition of MCN is based on relationship of entity nodes, an important operation is for a given keyword k, finding the set of entity nodes that have entity instances containing k as their attribute or attribute values, which is denoted as $selfE(k)$. We maintain an *auxiliary index* H that stores the set of partial paths (*not their instances*) for each keyword k, where each partial path has database instances appearing in the given XML document D, which can be got after parsing D. For example, H stores "photo/provider, person/name" for 'Mike'. According to H, selfE('Mike')={photo,person}.

Definition 7 (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 connect u and v in S, there is an edge in G that joins u and v.*

As shown in Fig. 4 (A), 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. According to Definition 7, we have the following lemma.

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.*

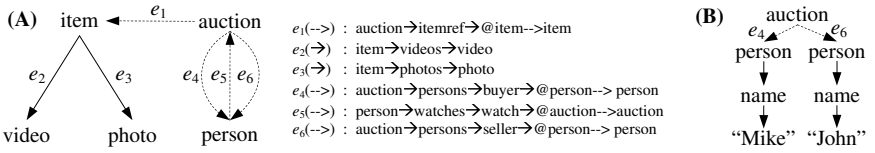

Fig. 4. The entity graph G (A) of S in Fig. 3, (B) is the QP of $R4'$ in Fig. 2.

Moreover, we observe that a QP consists of two kinds of relationships, (1) the relationship between entity nodes, i.e., *entity path*; (2) the relationship between

Algorithm 1: getQP(Q, G, C) /*$Q = \{k_1, k_2, ..., k_m\}$*/

```
1  Q ← ∅                                   /*queue of QPs*/
2  if (|Q| = 1) then {S_QP ← selfE(k_1); return S_QP}
3  foreach (combination E = (E'_1, E'_2, ..., E'_m), E'_i ∈ selfE(k_i)) do
4      get the partition P = {S_E1, S_E2, ..., S_Eq} of E according to entity names
5      get the keyword set K_Ei = {k | E ∈ S_Ei ∧ E ∈ selfE(k) ∧ k ∈ Q} of S_Ei
6      get the entity set S_K_Ei = {E_i^K | K ⊆ K_Ei} of E_i (1 ≤ i ≤ q)
7      put nodes of S_K_Ei into S_QP if they are QPs, otherwise put them into Q
8      while (¬ empty(Q)) do
9          J ← RemoveHead(Q)
10         foreach (edge e = (E, u) in G_u that is incident with a node u of J) do
11             if (E ∈ {E_1, E_2, ..., E_q}) then
12                 foreach (node E' ∈ S_K_E) do
13                     J' ← Add E' and (E', u) to J;
14                     if (isQP(J') ∧ nEntity(J') ≤ C) then S_QP ← S_QP ∪ {J'}
15                     else if (nEntity(J') < C) then Add J' to Q
16             else
17                 J' ← Add E and (E, u) to J;
18                 if (nEntity(J') < C) then Add J' to Q
19 return S_QP
```

entity nodes and attribute or attribute values, i.e., *partial path*. For example, after removing the node id, $R4'$ as a QP is shown in Fig. 4 (B), which contains two entity paths, i.e., e_4 and e_6, and a partial path, i.e., "person/name". According to Lemma 1, for a QP, the relationships of entity nodes can be got from entity graph, and the partial paths can be got from the auxiliary index H stated in the paragraph after Definition 6. For simplicity, we use the relationships of entity nodes of a QP to denote the QP in the following.

Algorithm 1 shows how to compute all QPs, which has three parameters, a keyword query Q, an entity graph G and the maximum number C of entity nodes. If Q contains only one keyword k, the set of QPs S_{QP} equals to selfE(k) (line 2); otherwise, for each combination $\mathcal{E}$=$(E'_1, E'_2, ..., E'_m)$ where $E'_i \in selfE(k_i)$, it computes the set of QPs of $\mathcal{E}$ (line 3-18). In particular, it first gets a partition P=$\{S_{E_1}, S_{E_2}, ..., S_{E_q}\}$ of $\mathcal{E}$ according to entity names, where $S_{E_i} = \{E | E \in \mathcal{E} \wedge label(E) = E_i\}$ (line 4). Then it gets the set of keywords $\mathcal{K}_{E_i}$ of S_{E_i} and gets all combinations of keywords contained by an entity node, i.e., multiple keywords may be contained by an entity node(line 5-6). Then adds all nodes of an entity set $S_{\mathcal{K}_{E_i}}$ to $\mathcal{Q}$ if they do not contain all keywords; otherwise, put them into S_{QP} (line 7). In line 8-18, while $\mathcal{Q}$ is not empty, a graph J is removed from $\mathcal{Q}$ in line 9 for further computing. isQP(J')=TRUE means that J' is a QP and nEntity(J') denotes the number of entity nodes in J'. Finally, S_{QP} is returned (line 19). Note a MCN is an instance of a QP, the checking of QP is similar to that of MCN except that QP is defined on schema graph.

Example 4. Assume C=3, G is the entity graph in Fig.4(A). As shown in Fig.5, according to D in Fig. 1, for Q=$\{M, J\}$,we have selfE('M')=selfE('J')=$\{P, PH\}$. There are three combinations of entities for Q, i.e., (PH, PH), (P, PH) and (P, P). For (PH, PH), according to line 4 of Algorithm 1, P=$\{S_{PH}\}$, where S_{PH}= $\{PH_1, PH_2\}$ (PH_1 and PH_2 denote they are two entity nodes of same name). In line 5, we know that a PH node may contain two keywords, i.e., $\mathcal{K}_{PH}$=$\{M, J\}$. According to line 6, there are three possible cases where a photo node contains these keywords, that is, $S_{\mathcal{K}_{PH}}$=$\{PH^{[M]}$, $PH^{[J]}$,$PH^{[M,J]}\}$. In line 7, $PH^{[M]}$ and $PH^{[J]}$ are put into $\mathcal{Q}$ and $PH^{[M,J]}$ is put into S_{QP} since it is already a QP. In line 9, $PH^{[M]}$ is first removed from $\mathcal{Q}$, since there is only one node, i.e., item, adjacent to photo in G and nEntity($PH^{[M]} \leftarrow^{e_4} I$)=2, it is put into $\mathcal{Q}$ in line 18. There are six newly generated graph for $PH^{[M]} \leftarrow^{e_4} I$, and only $PH^{[M]} \leftarrow^{e_4} I \rightarrow^{e_4} PH^{[J]}$ is a QP. We omit the computation for (P, PH) and (P, P) and just show them in Fig. 5.

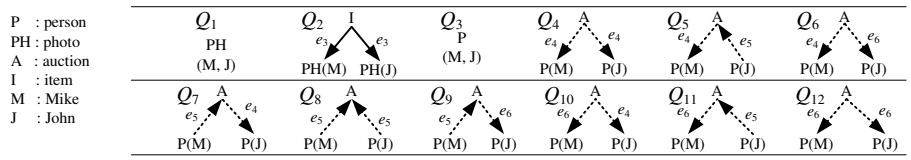

Fig. 5. The set of QPs of Q=$\{Mike, John\}$, partial paths are omitted, for Q_1 and Q_2, the partial path is 'photo/provider', for Q_3 to Q_{12}, partial path is 'person/name'.

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

The correctness of Theorem 1 is obvious according to Algorithm 1 and Example 4 and we omit the proof for limited space. Compared with the method of [7], our method made improvements from three aspects: (1) by postponing checking the real dataset until evaluating QPs, Algorithm 1 avoids the costly I/O operation, (2) Algorithm 1 is based on entity graph, which is much smaller than the expanded schema graph, and (3) the value of C in Algorithm 1 is much smaller than that of [7], where C equals to the number of edges in a QP.

5 Query Processing

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 $< P_{ID}, Path >$, where P_{ID} is the ID of a partial path, and $Path$ is a database instance of P_{ID} in XML documents. All $Path$s are clustered together according to P_{ID} and sorted in document order. The PPI of D in Fig. 1 is shown in Table 2, where only partial content is presented. The second index is *entity path index EPI*, for each edge e of the given entity graph, we maintain in EPI a set of database instances of e. The EPI of D in Fig. 1 is shown in Table 3.

Theorem 2 *Let Q be a given keyword query. Using PPI and EPI, the structural join operations can be avoided from the evaluation of Q.*

Table 2. The partial path index of the XML document D in Fig. 1.

Keyword	Tuple sets	Keyword	Tuple sets
Mike	$< photo/provider, 5/6 >$ $< person/name, 13/14 >$	John	$< photo/provider, 7/8 >$ $< person/name, 18/19 >$

Table 3. The entity path index of the XML document D in Fig. 1.

Edge	Entity Paths	Edge	Entity Paths
e_1	$\{23/24/25/11, 26/27/28/9\}$	e_4	$\{26/29/32/33/13\}$
e_2	$\emptyset$	e_5	$\{13/15/16/17/23, 18/20/21/22/23\}$
e_3	$\{2/4/5, 2/4/7\}$	e_6	$\{26/29/30/31/18\}$

Proof. [**Sketch**] According to Algorithm 1, any keyword query Q has a set of QPs S_{QP}. Each $Q' \in S_{QP}$ consists of two kinds of relationships, (1) the relationship between entity nodes and (2) the relationship between entity nodes and attribute or attribute values. The former can be computed by probing EPI and the latter can be computed by probing PPI, the final results can be got from the results of all selection operations on PPI and EPI.

As shown in Algorithm 2, we find the set of QPs S_{QP} in line 1. In line 2-3, all QPs that contain at least one selection operation that produces empty set is removed from S_{QP}. In line 4-12, we check for each node E_K of a QP Q', whether E_K has database instances that contain all keywords in K_E (K_E is the set of keywords attached to E_K). If E_K does not have such database instances, $R_{E_K} = \emptyset$. In line 13-17, for each QP $Q' \in S_{QP}$, if there is an entity node E_K and $R_{E_K} = \emptyset$, Q' is removed from S_{QP} in line 14; otherwise, Q' is evaluated in line 16. In line 17, the results $\mathcal{R}_{Q'}$ of Q' are put into $\mathcal{R}$. In line 18, $\mathcal{R}$ is returned.

Example 5. According to Example 4, $Q=\{Mike, John\}$ corresponds to 12 QPs shown in Fig. 5, i.e., $S_{QP}=\{Q_i | 1 \leq i \leq 12\}$. According to Table 3 and Table 2, we can get Table 4, which shows all selection operations of S_{QP}. Obviously, there are 8 selection operations involved in Q_1 to Q_{12}. According to line 4-12 and the PPI in Table 2, we have $\mathcal{E}=\{PH(M,J), PH(M), PH(J), P(M,J), P(M), P(J)\}$ for Q_1 to Q_{12} in Fig. 5. As $R_{PH(M,J)}=R_{P(M,J)}=\emptyset$, we can delete Q_1 and Q_3 from S_{QP} in line 14. In line 16, we evaluate the remainder QPs, among which three QPs have non-empty result sets, i.e., Q_2, Q_6, Q_8. $\mathcal{R}_{Q_2} = \sigma_{photo/provider \sim `Mike'} \bowtie \sigma_{e_3} \bowtie \sigma_{e_3} \bowtie \sigma_{photo/provider \sim `John'} = \{R1'\}$. Similarly, $\mathcal{R}_{Q_6}=\{R4'\}$, $\mathcal{R}_{Q_8}=\{R3\}$, where $R3, R1', R4'$ are the three MCNs in Fig. 2. Other QPs in S_{QP} have empty result sets. Therefore the final result set $\mathcal{R}_Q=\mathcal{R}_{Q_2} \cup \mathcal{R}_{Q_6} \cup \mathcal{R}_{Q_8} = \{R3, R1', R4'\}$.

Note Algorithm 1 may produce redundant QPs. For instance, consider the schema graph in Fig. 6 (A), where photo and item are entities, (B) is the XML document conforming to (A). Assume $C = 3$, i.e., each QP contains at most three entity nodes, Fig. 6 (C) and (D) are two QPs according to Algorithm 1. Obviously, Fig. 6 (E) and (F) are two matches of Fig. 6 (C) and (D), respectively. In fact, the QP in Fig. 6 (D) is redundant since in Fig. 6(E), both the two photo nodes and the two provider nodes represent same data element in Fig. 6 (B). Therefore before evaluating each QP Q' in $\mathcal{Q}$, we need to check in line 2-14 for each E_K, whether there exists entity instances of label E such that each entity instance d contains all keywords of K and no other $E'_{K'}$ exists such that $k \in (K' - K)$ and d contains k. Thus we have Theorem 3.

Theorem 3 (Non Redundancy.) *If a QP evaluated in line 16 of Algorithm 2 produces a MCN d, then no other QPs can produce d as their instance.*

Table 4. The selection operations of the 12 QPs of Q in Example 5.

Selection Op.	Result	Queries	Selection Op.	Result	Queries
$photo/provider{\sim}`Mike'$	$\neq \emptyset$	Q_1, Q_2	e_3	$\neq \emptyset$	Q_2
$photo/provider{\sim}`John'$	$\neq \emptyset$	Q_1, Q_2	e_4	$\neq \emptyset$	Q_4 to Q_7, Q_{10}
$person/name{\sim}`Mike'$	$\neq \emptyset$	Q_3 to Q_{12}	e_5	$\neq \emptyset$	Q_5, Q_7 to Q_9, Q_{11}
$person/name{\sim}`John'$	$\neq \emptyset$	Q_3 to Q_{12}	e_6	$\neq \emptyset$	Q_6, Q_9 to Q_{12}

Algorithm 2: indexMerge(Q, G, C)

```
1  S_QP ← getQP(Q, G, C)
2  foreach (edge e ∈ Q', where Q' ∈ S_QP) do
3      if (R_σe = ∅) then S_QP ← S_QP − {Q'}
4  foreach (node E_K ∈ Q' ∧ Q' ∈ S_QP ∧ E_K ∉ E) do
5      if (E_K is attached with keywords set K_E) then
6          R_EK ←merge the results of selection operations of each keyword of K_E
7      foreach (E'_K' ∈ E ∧ label(E) = label(E')) do
8          R = R_EK ∩ R_E'K'
9          if (K_E ⊂ K'_E') then R_EK ← R_EK − R
10         else if (K_E ⊃ K'_E') then R_E'K' ← R_E'K' − R
11         else {R_EK ← R_EK − R; R_E'K' ← R_E'K' − R}
12     E ← E ∪ {E_K}
13 foreach (Q' ∈ S_QP) do
14     if (∃E_K ∈ E ∧ E_K ∈ Q' ∧ R_EK = ∅) then S_QP ← S_QP − {Q'}
15     else
16         R_Q' ← merge the results of all selection operations of Q'
17         R ← R ∪ R_Q'
18 return R
```

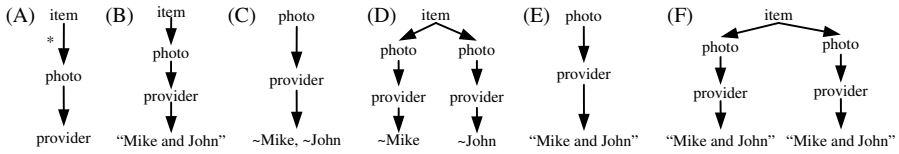

Fig. 6. Illustrating of redundant QP.

6 Experimental Evaluation

6.1 Experimental Setup

We used a PC with Pentium4 2.8 GHz CPU, 2G memory, 160 GB IDE hard disk, and Windows XP professional as the operating system. We implemented *IM* (short for indexMerge) algorithm using Microsoft Visual C++ 6.0. The compared methods include SLCA [1], XSEarch [3]. Further, we select two query engines, X-Hive[3] and MonetDB[4], to compare the performance of evaluating QPs.

[3] http://www.x-hive.com

[4] http://monetdb.cwi.nl/projects/monetdb/XQuery/index.html

6.2 Datasets, Indices and Queries

We use XMark[5] and SIDMOD[6](short for SIGMOD Record) datasets for our experiments. The main characteristics of the two datasets can be found from Table 5. The last column of Table 5 is *the ratio of index size to document size*, where index consists of (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.

Table 5. Statistics of datasets, L denotes Length.

Dataset	Size(M)	Nodes(M)	Max L	Avg L	Index/Doc.
XMark	115	1.7	12	5.5	5.3
SIGMOD	0.5	0.01	6	5.1	4.8

We select 40 keyword queries (32 from XMark and 8 from SIGMOD), which are omitted for limited space and classified into 4 groups containing 2, 3, 4 and 5 keywords, respectively. Table 6 shows the statistics of our keyword queries.

Table 6. Statistics of keyword queries. The 2^{nd} column is the average number of keywords of a keyword query, the 3^{rd} column is the average number of QPs of a keyword query, the 4^{th} column is the average number of entities in a QP, the 5^{th} column is the average number of distinct selection operations for the set of QPs of a keyword query.

Keyword queries	# of Keywords	Avg. # of QPs	Avg. # of entities	Avg. # of Sel. Op.
1^{st} group	2	5.7	1.77	11.8
2^{nd} group	3	10.3	2.13	14.1
3^{rd} group	4	16.5	2.42	19.4
4^{th} group	5	19	2.61	21.2

In our experiment, the node category is assigned using the method discussed in Section 2, we assume each MCN has at most 3 entity nodes, i.e., C=3 for Algorithm 2. Note C=3 means there may have 17 edges in a MCN, which is large enough to find most meaningful relationships.

6.3 Evaluation Metrics

We consider the following performance metrics to compare the performance of different methods: (1) Running time, (2) Precision and (3) Recall.

We define the Precision and Recall using the following steps: (1) users submit their keyword queries, (2) by asking users' search intension, we formulate the XQuery expressions corresponding to their keyword queries, then let them select the XQuery expressions that meet their search intension. 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=$(R_Q \cap R)/R$.

6.4 Performance Comparison and Analysis

Figure 7 (a) to (d) compare the Precision and Recall of different methods for the four group of keyword queries in Table 6, from which we know for XMark dataset, the Recall of our method is 100%, this is because all results that meet users'

[5] http://monetdb.cwi.nl/xml
[6] http://www.sigmod.org/record/xml/

search intension are returned by our method. However, the Precision is a little worse than SLCA and XSEarch, this is because our method may return results involving IDREF. If the users' search intension involves IDREF, obviously, our method will be more effective; otherwise, SLCA has the highest Precision. The average figures in Figure 7 (a) and (b) shows that for XMark dataset, though the Precision of our method is not better than SLCA and XSEarch, it has the highest Recall. For SIGMOD dataset, as shown in Figure 7 (c) and (d), both Precision and Recall of our method are better than SLCA and XSEarch, since in such a case IDREF is not considered, thus the number of QP is very small, usually equals to 1, and our method always return results that meet the users' search intension.

Figure 7 (e) shows the running time of different methods, where xHive and MonetDB process all QPs as our method. From this figure we know existing query engines cannot work well for these queries as they need to process large number of complex QPs (we try to merge as many as possible query patterns to one XQuery expression so as to make full use of their optimization methods to achieve better performance). Further, except SLCA (which is based on tree model and thus has lower Recall), our method achieves best query performance.

As the demo and optimization techniques of [7] are not publicly available, we do not make comparison with it. Even though, the improvement of our method is obvious and predictable. In our experiment, (1) C=3, (2) our method is based on an entity graph that is much smaller than the original schema graph, (3) our method does not need to scan real data. However, C=3 in our method means C=12 in [7] for XMark dataset, such a value is unnegligible because [7] is based on an expanded schema graph and the cost of computing QPs grows exponentially, let alone scanning the real data to construct an expanded schema graph for each query.

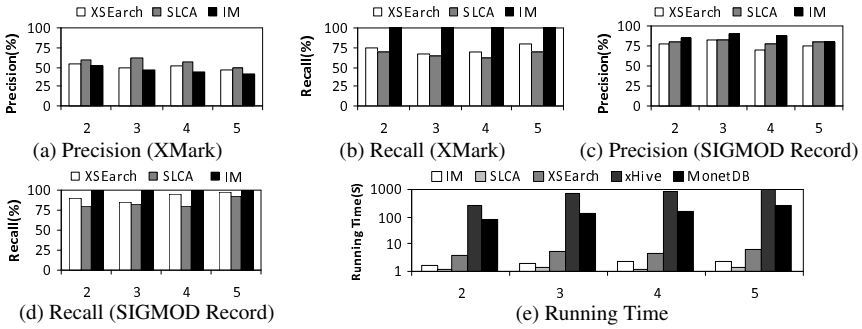

Fig. 7. The comparison of the average Precision (a and c), Recall (b and d) and running time (e). 2,3,4 and 5 denote the number of keywords in a query and the corresponding four query groups in Table 6, respectively.

7 Conclusion

In this paper, *Meaningful Connected Network (MCN)* was proposed to enhance the expressiveness of XML keyword search. Entity graph and two indices were

then introduced to improve the performance of query evaluation. We proved our method 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). The experimental results verify the effectiveness and efficiency of our method in terms of various evaluation metrics. We will focus on designing effective automatic node classification method and ranking mechanism considering node categories in the future work to provide higher reliability to the effectiveness of our keyword search method.

Acknowledgements. This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60833005, 60573091; China 863 High-Tech Program (No:2007AA01Z155); China National Basic Research and Development Program's Semantic Grid Project (No. 2003CB317000).

References

1. Yu, X., Yannis P.: Efficient Keyword Search for Smallest LCAs in XML Databases. In: SIGMOD Conference, pp. 537-538. (2005)
2. Lin, G., Feng, S., Chavdar, B., Jayavel, S.: XRANK: Ranked Keyword Search over XML Documents. In: SIGMOD Conference, pp. 16-27. (2003)
3. Sara, C., Jonathan, M., Yaron, K., Yehoshua, S.: XSEarch: A Semantic Search Engine for XML. In: VLDB Conference, pp. 45-56. (2003)
4. Ziyang, L., Yi, C.: Identifying meaningful return information for XML keyword search. In: SIGMOD Conference, pp. 329-340. (2007)
5. Yunyao, L., Cong, Y., Jagadish, H. V.: Schema-Free XQuery. In: VLDB Conference, pp. 72-83. (2004)
6. Cong, Y., Jagadish, H. V.: Querying Complex Structured Databases. In: VLDB Conference, pp. 1010-1021. (2007)
7. Vagelis, H., Yannis, P., Andrey, B.: Keyword Proximity Search on XML Graphs. In: ICDE Conference, pp. 367-378. (2003)
8. Sara, C., Yaron, K., Benny, K., Yehoshua, S.: Interconnection semantics for keyword search in XML. In: CIKM Conference, pp. 389-396. (2005)
9. Konstantin, G., Benny, K., Yehoshua, S.: Keyword proximity search in complex data graphs. In: SIGMOD Conference, pp. 927-940. (2008)
10. Hao, H., Haixun, W., Jun, Y., Philip, S., Y.: BLINKS: ranked keyword searches on graphs. In: SIGMOD Conference, pp. 305-316. (2007)
11. Vagelis, H., Yannis, P.: DISCOVER: Keyword Search in Relational Databases. In: VLDB Conference, pp. 670-681. (2002)
12. Reich, G., Widmayer, P.: Beyond Steiner's problem: a VLSI oriented generalization. In: WG Workship. (1990)
13. Geert, J., B., Frank, N., Stijn, V.: Inferring XML Schema Definitions from XML Data. In: VLDB Conference, pp. 998-1009. (2007)
14. Geert, J., B., Frank, N., Thomas, S., Karl, T.: Inference of Concise DTDs from XML Data. In: VLDB Conference, pp. 115-126. (2006)
15. Nicolas, B., Nick, K., Divesh, S.: Holistic twig joins: optimal XML pattern matching. In: SIGMOD Conference, pp. 310-321. (2002)
16. Haifeng, J., Wei, W., Hongjun, L., Jeffrey, X., Y.: Holistic Twig Joins on Indexed XML Documents. In: VLDB Conference, pp. 273-284. (2003)
17. Sihem, A., Y., SungRan, C., Laks, V., S., L.: Minimization of Tree Pattern Queries. In: SIGMOD Conference, pp. 497-508. (2001)