Yang D, Shen DR, Yu G *et al.* Query intent disambiguation of keyword-based semantic entity search in dataspaces. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(2): 382–393 Mar. 2013. DOI 10.1007/s11390-013-1338-0

Query Intent Disambiguation of Keyword-Based Semantic Entity Search in Dataspaces

Dan Yang[1,2] (杨 丹), *Student Member*, *CCF*, *Member*, *ACM*
De-Rong Shen[1,*] (申德荣), *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*
Ge Yu[1] (于 戈), *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*, Yue Kou[1] (寇 月), *Member*, *CCF*, *ACM*
and Tie-Zheng Nie[1] (聂铁铮), *Member*, *CCF*, *ACM*

[1] *College of Information Science and Engineering, Northeastern University, Shenyang 110004, China*

[2] *Software College, University of Science and Technology LiaoNing, Anshan 114044, China*

E-mail: yangdan@research.neu.edu.cn; {shenderong, yuge, kouyue, nietiezheng}@ise.neu.edu.cn

Received February 16, 2012; revised September 25, 2012.

Abstract Keyword query has attracted much research attention due to its simplicity and wide applications. The inherent ambiguity of keyword query is prone to unsatisfied query results. Moreover some existing techniques on Web query, keyword query in relational databases and XML databases cannot be completely applied to keyword query in dataspaces. So we propose KeymanticES, a novel keyword-based semantic entity search mechanism in dataspaces which combines both keyword query and semantic query features. And we focus on query intent disambiguation problem and propose a novel three-step approach to resolve it. Extensive experimental results show the effectiveness and correctness of our proposed approach.

Keywords query intent disambiguation, semantic entity search, dataspace

1 Introduction

Keyword query becomes the most popular search model due to its simplicity and wide applications. Meanwhile as the dataspaces have evolved into a data rich repository, it is very common that users search for various entities (e.g., phone numbers, papers, projects, conferences, persons) in dataspaces. So we propose KeymanticES, a novel keyword-based semantic entity search mechanism in dataspaces to satisfy user's information needs. And in this paper we focus on solving keyword query intent disambiguation problem. First let us consider the following motivating scenarios to illustrate the main problems need to be solved to effectively do keyword-based semantic entity search in dataspaces.

Scenario 1. The user submits keyword query "Halevy, dataspaces, SIGMOD 2011" to find all papers written by Halevy published in SIGMOD 2011 about dataspaces. If considering each keyword independently without semantic associations will produce many unrelated searching results. So it needs to infer keywords connections by considering the implicit semantic associations among these keywords.

Scenario 2. The user inputs keyword query "paper, dataspaces, SIGMOD 2011" to find all papers about dataspaces published in SIGMOD 2011. Here "paper" is the object type the user intends to find. Take another query "company, Zhongguancun, founders, Tsinghua graduates" as an example, the user wants to search the list of companies and their founders with searching conditions that the company is located in Zhongguancun and its founders are Tsinghua graduates. Here there are two object types: company and founder that the user wants to find. So it needs to infer the target entity class(es) the user wants to find out, e.g., paper, conference, or person whatever.

Scenario 3. The user who intends to find some IT experts whose research area is dataspace will probably submit query "IT expert, research area, dataspace" while not "Person, research area, dataspace". A user may not know which keywords to use for his/her specified search needs. Here assume the underlying entity class in the dataspace is *Person* but not *IT Expert*. So it needs to fill the concept gap or mismatch between the user and search mechanism to resolve such kind of queries.

Regular Paper

This research was supported by the National Basic Research 973 Program of China under Grant No. 2012CB316201, the National Natural Science Foundation of China under Grant Nos. 60973021, 61033007, 61003060, and the Fundamental Research Funds for the Central Universities of China under Grant No. N100704001.

*Corresponding Author

Although many research efforts have been conducted in keyword query over Web, relational databases (RDB) and XML databases, keyword query in dataspaces still faces many new challenges as follows: 1) Dataspace is a heterogeneous environment which involves different types of entities. Moreover there exist many different and rich associations among these entities which are more complex than primary-foreign-key relationships in RDB. It is of great challenge to leverage these associations effectively to grasp users' query intent. 2) Dataspace is short of unified or complete schema information. So some existing approaches of keyword query over relational databases leveraging schema graph, such as approaches based on Candidate Network (CN)[1-3], template-based[4-5] approach, are not fully applicable in dataspace. 3) With respect to query intent disambiguation, Web query tends to leverage Web log or search log to mine users' query pattern (e.g., statistics of users' click-through, analyzing URL, co-occurrence of keywords in the Web page). And XML keyword search can adopt LCA (lowest common ancestor)[6] or smallest LCA[7-9] semantic in tree data model. However these techniques are not fully applicable in dataspaces due to the lack of reliable query log and different data models.

The main contributions in this paper can be summarized as follows. 1) We propose KeymanticES, a novel keyword-based semantic entity search mechanism in dataspaces which combines the flexibility of keyword-based retrieval of entities with the ability to query on metadata/knowledge base which is typical of semantic search system. KeymanticES can find out user-desired entities in the underlying entity repository on-the-fly by issuing a keyword query. To the best of our knowledge, currently there is no keyword-based semantic entity search related work in dataspaces environment. 2) We propose a novel three-step query intent disambiguation approach which mainly consists of keyword semantic item mapping, goal entity class recognizing, and candidate query set generating leveraging rich associations of entity classes. 3) We propose an effective keyword semantic item mapping algorithm and an effective goal entity class recognizing algorithm. And we also propose a candidate query ranking function. 4) We conduct extensive experiments on real-world data to evaluate the effectiveness and correctness of our query intent disambiguation approach.

The rest of the paper is organized as follows. Section 2 presents our proposal, KeymanticES. In Section 3 query intent disambiguation approach and related algorithms are introduced in detail. Experimental results and analysis are provided in Section 4. Section 5 discusses related work and comparisons with our work. Section 6 summarizes the main contributions of the paper and our future work.

2 Our Proposal: KeymanticES

In this section, we first present the data model used in KeymanticES, then the definition of the keyword-based semantic entity search problem, and finally the overview of KeymanticES.

2.1 Data Model

In our previous work we proposed Layered Graph Data Model (lgDM)[10] in dataspaces which consists of entity data graph G_D and entity schema graph G_S. G_D is a directed labeled graph with labels corresponding to entities and associations of entities. G_S is a directed labeled graph to describe metadata information and associations of entity classes. They are both stored in the entity repository and at the same time treated as a semantic knowledge base (KB). And in order to solve the problem illustrated in Scenario 3, we propose a primitive concept graph G_{PC} (see Definition 1) which is also treated as part of KB. G_{PC} is domain dependent and can be built by domain expert or system developer.

Definition 1 (Primitive Concept Graph). *A primitive concept graph G_{PC} is a directed graph which consists of primitive concept nodes, and their parent-child (hierarchy) and similar relationship edges. The primitive concept is an entity class or type that entities belong to (e.g., Scientist, IT expert).*

Fig.1(a) shows an example of G_D which contains entities of six entity classes (*Paper*, *Person*, *Conference*, *Project*, *Publisher*, and *Journal*). And an example of corresponding G_S is shown in Fig.1(b). Here we call associations from an entity class to itself *self associations* (e.g., Co-author: *Paper*, Co_project: *Project*). Each self association has a target entity class related with it (e.g., *Paper*, *Project*). Fig.2 shows an example of G_{PC}. The one-way arrow directed edges indicate *parent-child* relationships of two concepts; the two-way arrow edges represent *similar* relationships. Throughout this paper, to provide context and specificity to our presentation, we will use such an academic publication example in our discussion.

2.2 Problem Statement

We formulate the problem of keyword-based semantic entity search in dataspaces as below.

Definition 2 (Keyword-Based Semantic Entity Search). *Given a keyword query $Q = \{k_1, k_2, \ldots, k_{|Q|}\}$ submitted by a user, keyword-based semantic entity search is to translate (query intent disambiguation) query Q into some semantic meaningful queries by leveraging associations (semantics) of keywords, then search them in the entity repository of dataspace, and finally return the ranked list of entities to the user.*

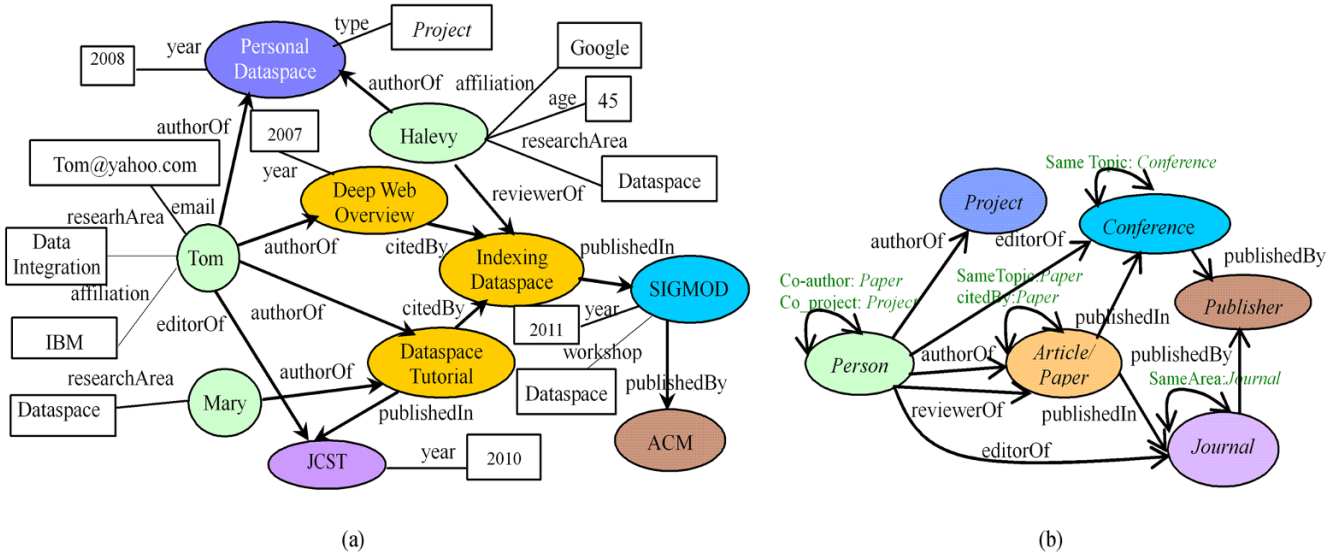


Fig.1. Example of data graph G_D and schema graph G_S. (a) Data graph G_D. (b) Schema graph G_S.

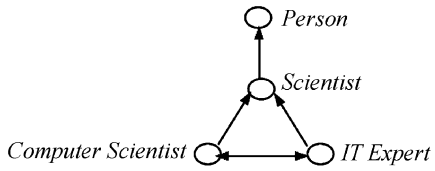


Fig.2. Example of primitive concept graph G_{PC}.

2.3 Overview of KeymanticES

Fig.3 shows the architecture of KeymanticES which is composed of the offline part and the online part. The offline part mainly deals with the preprocessing which consists of entity extraction (i.e., entity repository building) module and indexing module. In this work we only consider entities that are recognized, extracted, and stored in the entity repository, and indexed offline for efficient query processing. The online part mainly deals with semantic entity search processing which consists of three phases.

Phase 1: keyword query intent disambiguation to generate candidate query set.

Phase 2: semantic entity search according to each candidate query in the candidate query set.

Phase 3: search results ranking.

Issuing a keyword query by a user, KeymanticES tries to understand the user's query intent and on-the-fly finds out user-desired entities in the entity repository by analyzing semantics of keywords.

3 Query Intent Disambiguation Approach

The goal of query intent disambiguation is to automatically interpret the user's all possible search intentions and to generate candidate query set $CQ = \{cq_1, cq_2, \ldots, cq_n\}$, $n \geqslant 1$, of keyword query Q. The

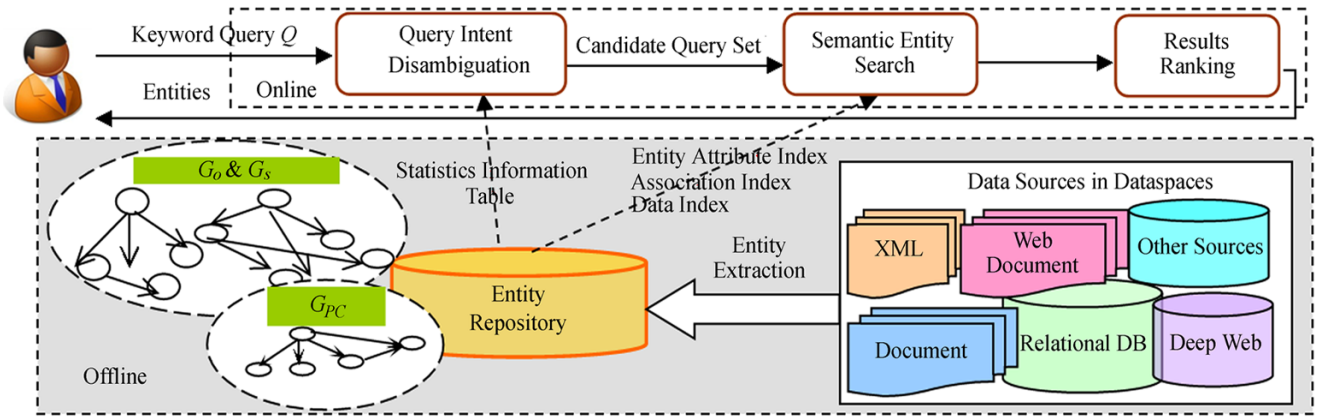


Fig.3. Architecture of KeymanticES.

format of each candidate query cq_i is "*goal entity class(es)* $\langle$(*entity class*$_1$ *attribute*$_1$ *value*$_1$ $\wedge \cdots \wedge$ *attribute*$_n$ *value*$_n$) [*association* (*entity class*$_2$ *attribute*$_1$ *value*$_1$ $\wedge \cdots \wedge$ *attribute*$_n$ *value*$_n$)]$^+\rangle$", where "$^+$" indicates zero or more, and contents inside "[]" are optional. Different *cq*s, correspond to different interpretations. Given a keyword query it is reasonable to infer different candidate queries because search intention may not be unique according to semantics of the query and the underlying entities. All possible combinations for the different semantic meanings of keywords are generated. The candidate queries may be cast as either SQL queries to be processed by a typical database system or keyword queries with additional predicates to be processed by a typical IR engine. In this paper we propose a novel three-step query intent disambiguation approach (shown in Fig.4).

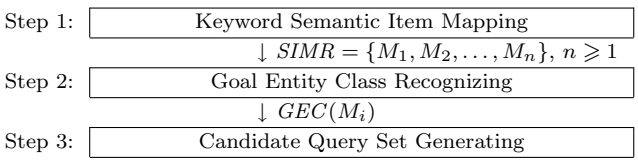


Fig.4. Steps of query intent disambiguation.

3.1 Semantic Item Mapping

When inferring the user's query intent, first we should find out the role of each keyword in the query. In this subsection we first give some related definitions, then describe the core challenges and our solution, at last give out our keyword semantic item mapping algorithm.

Definition 3 (Semantic Item). *The semantic item (SI) indicates the semantic role of a keyword. Semantic items in KeymanticES include entity class (EC), entity attribute (EA), value of attribute (V), association(A), primitive concept (PC), i.e.,* $SI = \{EC, EA, V, A, PC\}$.

Definition 4 (Semantic Item Mapping). *Semantic item mapping (SIM) is the process of mapping each keyword* k_i *in Q into one of the five elements of SI (i.e., EC, EA, V, A, PC). The SIM result is a set composed of all mapping results denoted as* $SIMR = \{M_1, M_2, \ldots, M_n\}$, $n \geqslant 1$, *where each* M_i *is composed of three parts, i.e.,* $M_i = \langle EC_i, A_i, V_i\rangle$, $EC_i \in EC$, $A_i \in A$, $V_i \in V$.

Here we assume each keyword in a query denotes an element of interest to the user and is not a stop-word or unrelated word. In this paper we do not address keyword query cleaning issues. So a keyword in the query may refer to either data instances (i.e., value of attribute), the meta-data (i.e., entity classes, entity attributes, associations), or primitive concept.

3.1.1 Core Challenges of SIM

One challenge of SIM is that when a keyword is mapped to an attribute's value, the result is not unique. We call it SIM ambiguity 1. For example, in query "paper, dataspaces", the mapping result of keyword "dataspaces" may be value of *Paper.topic*, or be value of *Paper.title*.

The other is that keywords in a query are semantically inter-dependent, which is called SIM ambiguity 2. For example, keyword "dataspaces" in query "paper, dataspaces" is probably the *title* attribute value of the *Paper* because of the co-occurence with entity class *Paper*; in query "person, dataspaces" it probably means the research area of *Person* because of co-occurence with *Person*; in query "project, dataspaces" it probably means *name* of *Project* due to the co-occurence with *Project*; while in query "conference, dataspaces" it probably indicates the *name* of *Conference* because of the co-occurence with *Conference*.

3.1.2 Statistic Information Table

Aiming at the core challenges mentioned above, we give a threshold-based mapping results filtering strategy. First we build a keyword statistic information table (denoted as SIT) based on frequency of each keyword in the underlying data which provides an objective way of solving SIM ambiguity 1 and ambiguity 2. Table 1 shows an example of SIT.

Table 1. Example of Statistic Information Table

Keyword k_i	Entity *class.attribute* [Frequency f]	Sum of f
dataspaces	*Paper.topic*[250], *Paper.title*[200], *Person.researchArea*[50], *Project.name*[15], *Conference.workshop*[10], *Person.email*[1]	526
⋮	⋮	⋮
Google	*Paper.title*[150], *Person.affiliation*[100]	250
Halevy	*Person.name*[3], *Person.email*[4]	7

Take the first row in Table 1 as an example. It indicates that keyword "dataspaces" appears in the value of *Paper.topic* attribute 250 times, value of *Paper.title* attribute 200 times, value of *Person.researchArea* attribute 50 times, value of *Project.name* attribute 15 times, value of *Conference.workshop* attribute 10 times and *Person.email* attribute once. Then we can calculate the relative frequency *rf* for each "*entity class.attribute*" of k_i in the SIT as follows.

$$rf(T[k_i]_j) = \frac{T[k_i]_j \times f}{sf}, \tag{1}$$

where $T[k_i]_j$ is the j-th "*entity class.attribute*" in the second column of the same row with k_i, *sf*

is the sum of frequencies of k_i in SIT. For example, for keyword "dataspace", *rf*(*Paper.topic*) = 250/526 = 0.475, *rf*(*Paper.title*) = 200/526 = 0.38, *rf*(*Person.researchArea*) = 50/526 = 0.095, *rf*(*Project.name*) = 15/526 = 0.029, *rf*(*Conference.workshop*) = 10/526 = 0.019, *rf*(*Person.email*) = 1/526 = 0.002.

Intuitively keywords referring to the same or related entities are adjacent in a query. For example, when a keyword is mapped to a meta-data, it becomes more likely that an adjacent keyword should be mapped to a value in the domain of that meta-data. So we introduce the concept of reward score to this kind of mappings.

Reward Score (*rs*). If any nearest neighbour of k_i (k_{i-1} or k_{i+1}) is matched with any part of $T[k_i]_j$ (entity class or attribute), then $rs(T[k_i]_j)$ of k_i is set to 1, else 0. For example, for keyword "dataspace", in query "person, dataspace, email", the neighbour keywords of it are "person" and "email". So *rs*(*Person.researchArea*) and *rs*(*Person.email*) are both set to 1, reward scores of other items (e.g., *Paper.topic*, *Paper.title, Project.name*, *Conference.workshop*) are 0.

Final Score (*fs*). *fs* of k_i is the sum of its *rs* and *rf*.

$$fs(T[k_i]_j) = rs(T[k_i]_j) + rf(T[k_i]_j). \quad (2)$$

In order to reduce some unrelated mapping results or noisy items, we adopt a filtering strategy based on final score *fs*. Set a threshold θ ($\theta \in [0..1]$) to *fs*, items of k_i with *fs* more than θ in SIT are selected as valid mapping results of k_i. Suppose $\theta = 0.01$, *fs*(*Person.email*) = 0.002. Then *Person.email* will be filtered and not treated as a reasonable mapping result of keyword.

3.1.3 Keyword SIM Algorithm

We observe that in a domain the number of attributes and their values are usually much larger than that of primitive concepts, entity classes and associations. So our SIM algorithm (shown in Algorithm 1) employs a greedy procedure which first determines the less ambiguous and simple mapping of keywords to primitive concepts, entity classes and associations, then determines the more complex mapping of keywords to other semantic items. Algorithm 1 assumes that each keyword can only be mapped to one kind of semantic items in each M_i. For example, "paper" in keyword query "Paper, Halevy" cannot be mapped to entity class and attribute at the same time in M_i.

Algorithm 1. Keyword Semantic Item Mapping

Input: Keyword query $Q = \{k_1, k_2, \ldots, k_{|Q|}\}$,
G_{PC}, G_S, SIT,
fs: final score, θ: threshold of *fs*

Output: $SIMR = \{M_1, M_2, \ldots, M_n\}$, $n \geqslant 1$

```
1:  if (Q is not NULL) then
2:    for i = 1 to |Q| do
3:      Scan PC set;
4:      if (matched (k_i) == TRUE) then
5:        F_ec(k_i).addTo(EC_i);
6:        Q.remove(k_i); //entity class of k_i is denoted
          as F_ec(k_i)
7:      end if
8:      scan EC set;
9:      if (matched(k_i) == TRUE) then
10:       k_i.addTo (EC_i);
11:       Q.remove(k_i);
12:     end if
13:     scan A set;
14:     if (matched(k_i) == TRUE) then
15:       k_i.addTo (A_i);
16:       Q.remove(k_i);
17:     end if
18:   end for
19:   if (Q is not NULL) then
20:     for j = 1 to |Q| do
21:       Scan SIT;
22:       for each fs(T[k_m]_n) > θ do
23:         k_j.addTo(V_i);
24:         if (F_ec(k_j) is not in EC_i) then
25:           F_ec(k_j).addTo (EC_i);
26:           Q.remove(k_m);
27:         end if
28:       end foreach
29:     end for
30:   end if
31: end if
32: return SIMR
```

3.2 Goal Entity Class Recognizing

When inferring a user's query intent it is a key problem to determine which object (entity class) the user wants. In this subsection, after giving some related definitions, we describe the core challenges and our heuristic rules, and then describe our goal entity class recognizing algorithm.

Definition 5 (Goal Entity Class). *Given a keyword query Q and its semantic item mapping result set $SIMR = \{M_1, M_2, \ldots, M_n\}$, $n \geqslant 1$, for each $M_i \in SIMR$, goal entity class (GEC) of it is denoted as $GEC(M_i)$ which is composed of all the possible searching target entity class(es) sets of M_i, i.e., $GEC(M_i) = \{GEC(M_i)_1, GEC(M_i)_2, \ldots, GEC(M_i)_j\}$, $j \geqslant 1$, where $GEC(M_i)_j = \{ec_1, ec_2, \ldots, ec_n\}$, $n \geqslant 1$, $ec \in EC$.*

Definition 6 (Goal Entity Class Recognizing). *Goal entity class recognizing is the process of inferring each M_i's possible implicit target entity class(es) involved in the searching entity results.*

For each $M_i \in SIMR$, we do GEC recognizing and try to infer goal entity class(es) the user desires. And from scenario 2 we notice that sometimes users want to find more than one target entity classes in the query. Consequently the size of each $GEC(M_i)_j$ is equal or greater than one, i.e., $|GEC(M_i)_j| \geqslant 1$.

3.2.1 Core Challenges of GEC Recognizing

One challenge of GEC recognizing is that keyword-based entity search in dataspaces involves many different entity classes, not simply one entity classes in some other systems (e.g., product search system, paper search system). And usually users do not explicitly input GEC as a keyword in a query. For instance, a user submits query "Mary, SIGMOD 2011" to find all papers Mary published in SIGMOD 2011. Here *Paper* is the goal entity class but it does not appear in the query as a keyword.

Another is that usually an entity class has many self associations and associations related to it. That is to say there are many semantic combinations for these keywords. For example, a user submits query "Mary, JCST". There are many direct or indirect associations between *Person* and *Journal* (e.g., "$Person \xrightarrow{\text{editorOf}} Journal$", "$Person \xrightarrow{\text{authorOf}} Paper \xrightarrow{\text{publishedIn}} Journal$"). So the user maybe intends to find all Mary's papers published in JCST or some person whose name is Mary and who is the editor of JCST, etc.

3.2.2 GEC Recognizing Heuristic Rules

To address the challenges above we propose a GEC recognizing strategy which makes semantic associations of entity classes first-class citizen. When association subset AS_i of M_i is not null, namely there is(are) explicit association(s) appearing in the query, we infer GEC by the association of the known entity class directly. For example, association "authorOf" appears directly in query "authorOf, dataspaces tutorial" or query "authorOf, paper, dataspaces tutorial", and assuming "dataspaces tutorial" is mapped to entity class *Paper*, according to G_S there exists association "$Person \xrightarrow{\text{authorOf}} Paper$", we assume that the user intends to find person entities who are authors of the paper "dataspaces tutorial", so here $GEC\,(M_i)_1 = \{Person\}$. Otherwise we give some GEC recognizing heuristic rules according to entity class subset EC_i of M_i. Firstly two related definitions used in heuristic rules are given.

Definition 7 (Association Tree). *An association tree (AT) is a tree composed of at least three different entity classes(nodes) and their associations. The related degree of an entity class ec (denoted as $R(ec)$) in each AT is represented by the sum of indegree and outdegree of each node. For example, AT "project ← person → paper → conference", the related degree of each entity class is as follows: $R(Project) = 1$, $R(Person) = 2$, $R(Paper) = 2$, $R(Conference) = 1$.*

Definition 8 (Association Chain) *An association chain (AC) is a chain composed of at least three different entity classes and their associations which is a special AT. The length of an AC is the number of entity classes in the chain. Each path (if the path length $\geqslant 3$) from the root to leaf in AT can be looked as an AC, e.g., "Person → Paper → Conference".*

Rule 1. If M_i includes only one entity class *ec* (i.e., $|EC_i| = 1$) and the name of entity class *ec* appears in the query Q directly, then generally it is $GEC(M_i)$ by default, namely $GEC(M_i) = \{ec\}$. For instance, keyword "paper" appears in query "paper, dataspaces" directly, so here $GEC(M_i)_1 = \{Paper\}$.

Rule 2. If M_i includes only one entity class *ec* (i.e., $|EC_i| = 1$), then if the number of entities involved are more than one, then each $GEC(M_i)_j$ is composed of the target entity class of each self association of *ec* according to G_S, namely $GEC(M_i)_j$ = {target entity class of the j-th self association of *ec*}; else $GEC(M_i)_1 = \{ec\}$. For example, M_i of query "Halevy, Xin Dong" involves two entities, entity class *Person* has two self associations, e.g., *co_author*, *co_project*. So $GEC(M_i)_1 = \{Paper\}$, $GEC(M_i)_2 = \{Project\}$. For another example, assuming M_i of keyword query "Halevy" includes only one entity class *Person* and involves only one entity, so here $GEC(M_i)_1 = \{Person\}$.

Rule 3. If M_i includes three or more entity classes (i.e., $|EC_i| \geqslant 3$), and these entity classes can form ACs or ATs directly or by another entity class indirectly, then $GEC(M_i)_j$ is composed of entity class *ec* with maximum related degree $R(ec)$ of each AC or AT, namely $GEC(M_i)_j = \{\arg\max_{ec} R(ec)_j\}$, where $ec \in EC_i$. Otherwise refer to rules 4∼6. For instance, assuming there exists an AC for the three entity classes of M_i "*Person → Paper → Conference*", so $GEC(M_i) = \{Paper\}$. Assuming there exists an AT for the following four entity classes of M_i: "*Project ← Person → Paper → Conference*", then $GEC(M_i)_1 = \{Paper\}$ and $GEC(M_i)_2 = \{Person\}$. For another example, assuming M_i includes three entity classes *Journal*, *Person* and *Conference*, according to Fig.2 they form an AT "*Conference ← Person → Journal*" directly, and an AT "*Person → Paper → Conference/Journal*" through *Paper* indirectly, so $GEC(M_i)_1 = \{Person\}$ and $GEC(M_i)_2 = \{Paper\}$ respectively.

Rule 4. If M_i includes two entity classes (i.e., $|EC_i| = 2$), and they can form AC or AT indirectly through the third entity class, then refer to rule 3. For example, M_i includes two entity classes *person* and *conference* which can form an AC "*Person* → *Paper* → *Conference*" indirectly by *Paper*, so according to rule 3, $GEC(M_i)_1 = \{Paper\}$.

Rule 5. If M_i includes two entity classes (i.e., $|EC_i| = 2$), and there exists no indirect AC or AT through the third entity class, but there is(are) direct association(s) between them, then $GEC(M_i)$ is composed of these two entity classes. For example, M_i includes two entity classes *Person* and *Project* which have a direct association "*Person* → *Project*", so $GEC(M_i)_1 = \{Person, Project\}$.

Rule 6. If M_i includes two entity classes (i.e., $|EC_i| = 2$), and there exists no indirect AC or AT and no direct associations between them, then $GEC(M_i)$ is composed of these two entity classes, namely $GEC(M_i)_1 = \{ec_1\}$ and $GEC(M_i)_2 = \{ec_2\}$.

3.2.3 GEC Recognizing Algorithm

Algorithm 2 gives the pseudo-code of GEC recognizing algorithm based on the above heuristic rules.

Algorithm 2. Goal Entity Class Recognizing
Input: $M_i \in SIMR$; keyword query Q;
Output: $GEC(M_i)$

```
1:  if (EC_i is not NULL) then
2:    Initialization: C ← ∅;
3:    switch (size of EC_i)
4:    case 1: if (size of involved entities) > 1) then
5:              GEC(M_i)_j = {target entity class of the
                j-th self association of ec};
6:            else GEC(M_i)_1 = {ec};
7:            end if
8:            break;
9:    case 2: if (exists indirect AC or AT between ec_1
                and ec_2) then
10:             goto case 3;
11:           else if (exists direct association(s)
                      between ec_1 and ec_2) then
12:             GEC(M_i) = {ec_1, ec_2};
13:           else GEC(M_i)_1 = {ec_1};
                   GEC (M_i)_2 = {ec_2}
14:           end if
15:           end if
16:           Break;
17:   case ⩾ 3: if (exists AC or AT between
                  entity classes) then
18:             GEC(M_i) = arg max_ec (R(ec))
19:           end if
20:           Break;
21: return GEC(M_i);
```

3.3 Candidate Query Set Generating

In the step for each M_i we supplement possible associations for mapped entity classes in EC_i according to G_S, and generate candidate query set *CQ*. In order to provide users a ranked interpretation list of candidate queries according to the relevance of each candidate query *cq* to query Q, we define a ranking function *Score* (*cq*, Q) which also gives preference to semantic associations of entity classes. *Score*(*cq*, Q) contains three components (shown in (3)) which reflect our four principles in designing the function. The first component considers whether exists AC or AT and length of AC (denoted as *Length*(AC)), namely the query with highly correlated keywords is with high score. The second component considers average importance degree I of all entity class(es) in GEC. Since entity classes in a dataspace do not have the same importance, users may be more interested in *cq* with more important goal entity class(es). The importance degree of an entity class I can be got by calculating the score of each node (entity class) in G_S using PageRank algorithm[11], which is a classic algorithm to measure comparative importance of nodes in a graph structure. The third component considers the sum of final score *fs* of each keyword mapped to an entity's attribute value. The fourth component is keywords translation completeness. Ideally we prefer a complete candidate query including all the keywords of query to an incomplete mapping with only a subset of keywords.

$$Score(cq, Q) = \frac{Length(\text{AC } of \ cq)}{MaxLength} \oplus AVG(I(ec \in \text{GEC})) \oplus \sum_{i=1}^{p} fs(k_i) \oplus \frac{|k_i \in cq \cap Q|}{|Q|}, \quad (3)$$

where $\oplus$ is a binary aggregation function, and each component of the formula can have a weight set by the user or system and the sum of weights is 1; *MaxLength* is the possible maximum length of ACs (be varied according to different domains), e.g., $MaxLength = 10$; $I(ec \in \text{GEC})$ indicates the importance degree of *cq*'s GEC, and *AVG* is its average score; p is the number of keywords mapped to an entity's *attribute.value*; $|k_i \cap cq \cap Q|$ is the number of common keywords of *cq* with Q; $|Q|$ is the number of keywords of query Q. Each component in the formula is normalized to [0..1].

4 Experiments

In this section, we first introduce datasets and query sets used in our experiments and then show and discuss the experimental results on query intent disambiguation performance study and KeymanticES performance study respectively.

4.1 Experimental Setup

We implement our method and conduct the experiments on a 3.16 GHz Pentium 4 machine with 4 GB memory and 500 GB of disk. We employ a CN-based keyword search approach DISCOVER-II[2] as the baseline. Though its underlying data is different from ours, the goal is similar to our approach, because both methods infer query structures from keywords and do semantic search.

Datasets. Our experiments are based on two real datasets. 1) DBLP. It is a global person register for researchers of computer science and neighboring sciences. We extract entities and associations from the four research areas of DBLP dataset: DB (database), DM (data mining), IR (information retrieval), and AI (artificial intelligence). In addition we modify and extend it manually for our experimental requirements. For example, we introduce a new entity class *Project*, and add attributes and values (e.g., affiliation, research area) to some entities (e.g., *Person* entities). Accordingly we add some associations among these entity classes (e.g., $Person \xrightarrow{\text{authorOf}} Project$, $Paper \xrightarrow{\text{supportedBy}} Project$). Finally the dataset contains entities of six entity classes (i.e., *Paper*, *Person*, *Conference*, *Project*, *Journal*, *Publisher*) and their associations. 2) DBLife[12]. It manages information for the database research community. And the dataset comprises five entity tables (*person*, *publication*, *topic*, *organization*, *conference*), and nine relationship tables (*relatedPeople*, *relatedTopic*, *relatedOrg*, *serveConf*, *giveConfTalk*, *giveOrgTalk*, *coAuthor*, *writePub*, *giveTutorial*). We extract five entity classes and their associations from these tables.

Query Sets. We manually select keywords from above two datasets respectively to form queries with length from 1 to 6 (denoted as $N1 \sim N6$). Each query type ($N1 \sim N6$) contains five queries on each dataset. The query sets include different types of queries such as flat queries (i.e., search for entities with only attributes constrains but no associations), chain queries (i.e., search for entities with nesting of associations or conjunction of associations). All the queries contain ambiguities: SIM ambiguity 1 or SIM ambiguity 2 or both. Most of the queries do not contain explicit goal entity class(es). And the number of goal entity class of some queries is more than 1. For each query of $N1 \sim N6$, we ask for six students in our lab to give their desired query intents (e.g., goal entity class, candidate queries, ranking of them) for each query.

4.2 Query Intent Disambiguation Performance Study

Effectiveness. The experiments include: 1) the quality of the GEC recognizing; 2) the effectiveness of *CQ* generating.

GEC Recognizing. The average precision comparison of goal entity class recognizing on the two datasets is shown in Table 2. We can see that our approach is able to infer a desired goal entity class with high precision in most cases and even high when the number of keywords is more than 4.

Table 2. Average Precision of GEC Recognizing

Dataset	Number of Keywords					
	1	2	3	4	5	6
DBLP	0.97	0.95	0.93	0.93	0.92	0.92
DBLife	0.95	0.94	0.92	0.91	0.91	0.89

CQ Generating. First we evaluate the effectiveness of candidate query set ranking strategy. We use three widely adopted measures: 1) number of top-1 answers that are relevant; 2) reciprocal rank (R-rank); 3) mean average precision (MAP). The experimental results on the two datasets are shown in Table 3. We find that our approach can return top-1 *cq* in most cases and has an average R-rank greater than 0.8 and even over 0.9 on DBLP.

Table 3. Ranking Performance

Dataset	Top-1 Number/Total Number	R-Rank	MAP
DBLP	27/30	0.95	0.925
DBLife	25/30	0.89	0.853

Then we measure the effect of associations to the size of candidate query set *CQ* on the two datasets. We experiment when the number of associations (#as) is 5, 8, and 12 respectively (shown in Fig.5). From the figure we can see the overall trend is that the size of *CQ* increases with the number of keywords. Moreover, the larger the number of associations is, the larger the size of *CQ* is. It shows that associations are very important to the effectiveness of query intent disambiguation.

Efficiency. We examine the average execution time of each step of our query intent disambiguation approach. The experimental results are shown in Fig.6. From the figure we can see that the full execution time of query intent disambiguation is incremental with the increasing of the number of keywords gradually. Additionally, as the number of keywords increases, the

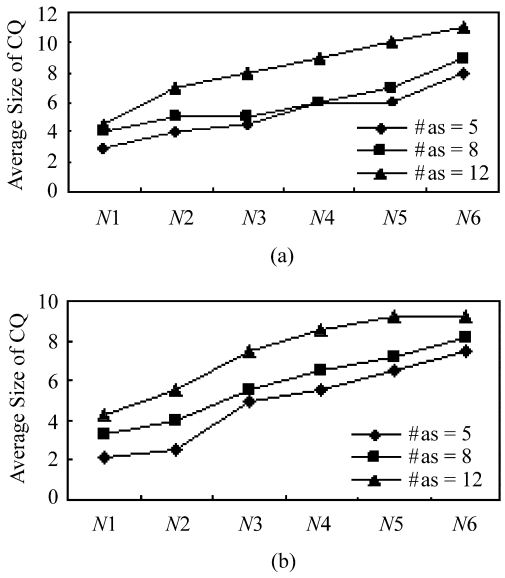


Fig.5. Effect of associations on the size of *CQ*. (a) DBLP. (b) DBLife.

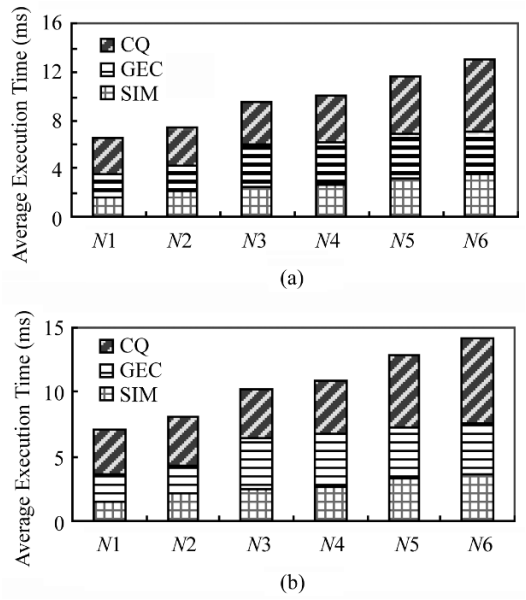


Fig.6. Time performance of query intent disambiguating. (a) DBLP. (b) DBLife.

execution time of SIM increases; the time spending on GEC recognizing increases very little because the number of entity classes in the mapping results has a small change; the execution time of *CQ* generating increases due to the size of candidate query set increasing. In general the *CQ* generating accounts for large fraction of the total execution time.

Scalability. Due to the evolution feature of dataspaces, it is not uncommon that the number of underlying entity classes and their associations change with time. So we experiment on DBLife dataset to evaluate the flexibility and scalability of our approach. Fig.7 shows the average execution time of our query intent disambiguation approach when the number of entity classes (#ecs) is 3, 4 and 5 respectively. We observe that the runtime increases with the increment of the number of entity classes. And the runtime is stable between 6 ms and 16 ms.

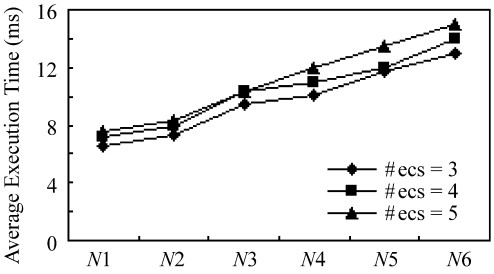


Fig.7. Scalability of query intent disambiguation on DBLife.

4.3 KeymanticES Performance Study

Effectiveness. We use the following metrics: precision, recall and F-score defined in (4) to compare the effectiveness of KeymanticES and DISCOVER-II on the two datasets. The experimental results are shown in Fig.8 and Fig.9 respectively. From the figures it can be seen that KeymanticES performs better than DISCOVER-II. Because DISCOVER-II cannot return results for queries containing keywords such as "paper", "person", "authorOf", "Scientist". The improvement is due to the reason that unlike DISCOVER-II, instead of mapping keyword to the value of attribute in the table's text column, KeymanticES maps the keyword to either

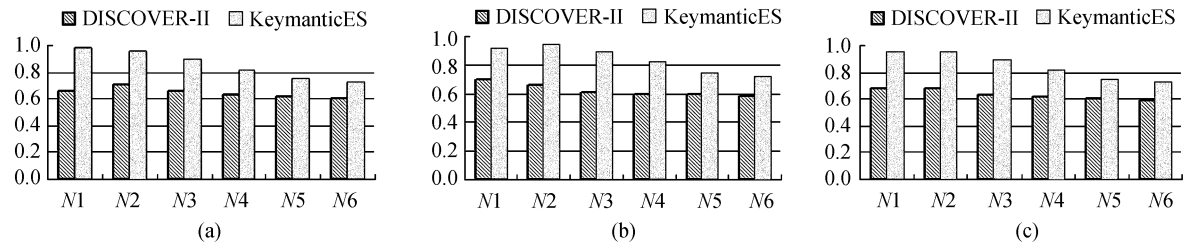


Fig.8. Effectiveness comparisons on DBLP database. (a) Precision. (b) Recall. (c) F-score.

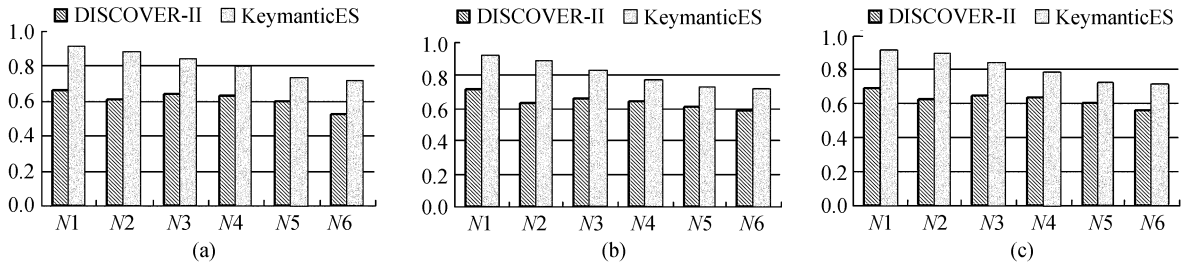


Fig.9. Effectiveness comparisons on DBLife database. (a) Precision. (b) Recall. (c) F-score.

meta-data, data instance, or primitive concept.

$$\text{precision} = \frac{searchedRelatedEntities}{allSearchedEntities},$$

$$\text{recall} = \frac{searchedRelatedEntities}{allRelatedEntities},$$

$$F\_\text{score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (4)$$

Scalability. We examine the scalability of KeymanticES on DBLife dataset when the number of keywords (#keywords) is 3, 4, 5, and 6 respectively. The experimental result is shown in Fig.10. From the figure we can see that the query time increases in a sub-linear fashion with the increase of size of the dataset, and the query time is stable between 7 ms and 26 ms.

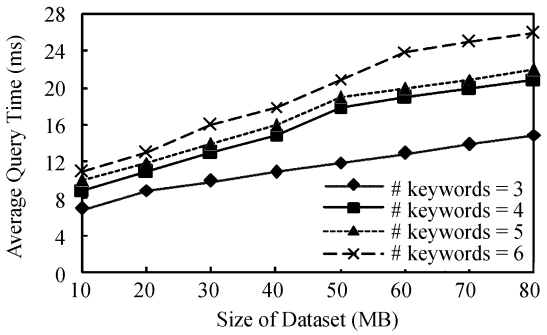


Fig.10. Scalability of keymanticES DBLife.

5 Related Work

Keyword search is becoming a widely accepted query mechanism for information searching due to its simplicity. But keyword query is by nature ambiguous, and one of its challenges is to infer users' query intent from limited keywords. Query intent disambiguation query translation, or query transformation is to make keyword query have semantic information or structure information to better represent the user's underlying intent. The problem has been studied mainly in the fields of IR, Web query, the context of keyword search over relational databases or semi-structure XML data. In IR, the state-of-the-art is to use statistics models like conditional random fields (CRF), hidden Markov model (HMM), Markov random field (MRF) or statistical language models[13] like n-gram language models to find the dependency between keywords in a query. Some related work proposes to answer keyword query by query forms[14-15] which infer structures from existing structured query templates. Related work about web query leverages the query log or click log to do keywords translation, such as [16-19]. Some related work does query disambiguation based on structure information of underlying relational database such as [4-5, 20-23]. Keymantic[22] translates the ambiguous keyword queries into full specified SQL expressions. It treats the keyword search as a bipartite graph assignment problem and uses an extended version of the Hungarian algorithm. Related work in [23] has a goal similar to that of [22], but it follows a different approach. It proposes a probabilistic approach based on a hidden Markov model. SQLS_{UGG}[5] suggests SQL queries based on queryable templates as users type in keywords. Its goal is similar to ours, but our method can match keywords to primitive concepts and associations, which is not supported by SQLS_{UGG}. Moreover it transforms the query template to SQL query which is the combination of all matched attributes of keywords in SELECT clause. Instead of records, our approach can further identity goal entity class of the query and return entities to users. In the context of semantic XML keyword search, XML-oriented query engine SphereSearch[24] supports concept-aware, context-aware and abstraction-aware search on XML data by leveraging the implicit structure and context of Web pages. However it does not take into account specific labeled relationships between entities. XSeek[8] adopts smallest LCA semantic and considers a node type as an concept leveraging DTD. It infers *return* nodes by analyzing patterns of query keyword matches and data semantics. However it does not consider ranking problem and keyword ambiguity problem. XReal[25] exploits the statistics of underlying XML database to address search intention identification (i.e., identifying *search for* nodes, *search via* nodes), result retrieval and relevance oriented ranking without using any schema know-

ledge. The strategy is more suitable for tree models and falls short in our context. Though the goal of our query intent disambiguation approach is similar to XReal and both of them are to identify the user search intention, we use different method due to different data models. In terms of keyword search over heterogeneous data sources, EASE[26] extends Steiner tree with r-radius Steiner graph as the query response unit. It models unstructured, semi-structured and structured data as graphs with nodes being documents, elements and tuples respectively. By contrast KeymanticES models entities as nodes in our layered graph data model aiming at entity search. Besides, though the research background of EASE is similar to dataspace, it focuses on the issues of indexing and ranking.

Our proposal KeymanticES is different from the above related work and has the following advantages: 1) KeymanticES is an entity search mechanism aiming at all kinds of heterogonous data sources in dataspaces, not limited to documents, structured relational databases, or XML databases. 2) KeymanticES provides semantic search (but not OR and AND semantic of keywords) using simple keyword-based search model by leveraging rich associations of entity classes. 3) KeymanticES supports searching for entities of multiple entity classes instead of just one entity class. 4) Our three-step query intent disambiguation approach can effectively translate a keyword query to ranked candidate queries step by step.

6 Conclusions and Future Work

In this paper, we studied the problem of keyword-based semantic entity search in dataspaces and proposed a novel mechanism: KeymanticES. Furthermore, focusing on keyword query intent disambiguation problem we proposed a novel three-step approach which effectively solves the semantic ambiguity of keyword query problem and improves the precision of keyword query in dataspaces. Experimental results show the effectiveness and correctness of the proposed approach. Our future work will consider the effect of sequence or order of keywords, e.g., keyword query "paper, dataspaces" and "dataspaces, paper" may be with different query intents or search focuses.

References

[1] Hristidis V, Papakonstantinou Y. Discover: Keyword search in relational databases. In *Proc. the 28th VLDB*, Aug., 2002, pp.670-681.

[2] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-style keyword search over relational databases. In *Proc. the 29th VLDB*, Sept. 2003, pp.850-861.

[3] Luo Y, Lin X, Wang W *et al.* Spark: Top-k keyword search engine on relational databases. In *Proc. ICDE*, Apr. 2008, pp.1552-1555.

[4] Demidova E, Zhou X, Nejdl W. IQ$^{\text{P}}$: Incremental query construction, a probabilistic approach. In *Proc. the 26th ICDE*, Mar. 2010, pp.349-352.

[5] Fan J, Li G L, Zhou L Z. Interactive SQL query suggestion: Making databases user-friendly. In *Proc. the 27th ICDE*, Apr. 2011, pp.351-362.

[6] Schmidt A, Kersten M L, Windhouwer M. Querying XML documents made easy: Nearest concept queries. In *Proc. the 17th ICDE*, Apr. 2001, pp.321-329.

[7] Xu Y, Papakonstantinou Y. Efficient keyword search for smallest LCAs in XML databases. In *Proc. SIGMOD*, June 2005, pp.537-538.

[8] Liu Z Y, Walker J, Chen Y. XSeek: A semantic XML search engine using keywords. In *Proc. the 33rd VLDB*, Sept. 2007, pp.1330-1333.

[9] Li Y, Yu C, Jagadish H V. Schema-free XQuery. In *Proc. the 30th VLDB*, Aug. 31-Sept. 3, 2004, pp.72-83.

[10] Yang D, Shen D R, Nie T Z *et al.* Layered graph data model for data management of dataspace support platform. In *Proc. the 12th WAIM*, Sept. 2011, pp.353-365.

[11] Brin S, Page L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 1998, 30(1/7): 107-117.

[12] Derose P, Shen W, Chen F *et al.* DBLife: A community information management platform for the database research community. In *Proc. CIDR*, Jan. 2007, pp.169-172.

[13] Zhai C. Statistical language models for information retrieval: A critical review. *Foundations and Trends in Information Retrieval*, 2008, 2(3): 137-213.

[14] Chu E, Baid A, Chai X *et al.* Combining keyword search and forms for ad hoc querying of databases. In *Proc. SIGMOD*, June 29-July 2, 2009, pp.349-360.

[15] Tata S, Lohman G M. SQAK: Doing more with keywords. In *Proc. SIGMOD*, June 2008, pp.889-902.

[16] Venkatesh G, Yeye H, Dong X. Keyword++: A framework to improve keyword search over entity databases. In *Proc. VLDB*, Sept. 2010, pp.711-722.

[17] Nikos S, Stelios P, Panayiotis T. Structured annotations of web queries. In *Proc. SIGMOD*, June 2010, pp.771-782.

[18] Paprizos S, Ntoulas A, Shafer J *et al.* Answering web queries using structured data sources. In *Proc. SIGMOD*, June 29-July 2, 2009, pp.1127-1130.

[19] Cheng T, Lauw H W, Paparizos S. Fuzzy matching of Web queries to structured data. In *Proc. ICDE*, Mar. 2010, pp.713-716.

[20] Demidova E, Zhou X, Zenz G *et al.* SUITS: Faceted user interface for constructing structured queries from keywords. In *Proc. DASFAA*, Apr. 2009, pp.772-775.

[21] Pound J, IIyas I F, Weddell G E. Expressive and flexible access to web-extracted data: A keyword-based structured query language. In *Proc. SIGMOD*, June 2010, pp.423-434.

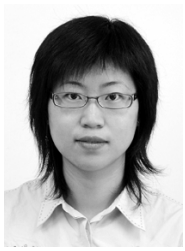
[22] Bergamaschi S, Domnori E, Guerra F. Keyword search over relational databases: A metadata approach. In *Proc. SIGMOD*, June 2011, pp.565-576.

[23] Bergamaschi S, Guerra F, Rota S *et al.* A hidden Markov model approach to keyword-based search over relational databases. In *Proc. ER*, Oct. 31-Nov. 3, 2011, pp.328-331.

[24] Graupmann J, Schenkel R, Weikum G. The sphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *Proc. VLDB*, Aug. 30-Sept. 2, 2005, pp.529-540.

[25] Bao Z F, Ling T W, Chen B *et al.* Effective XML keyword search with relevance oriented ranking. In *Proc. ICDE*, Mar. 29-Apr. 2, 2009, pp.517-528.

[26] Li G L, Ooi B C, Feng J H *et al.* EASE: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proc. SIGMOD*, June 2008, pp.903-914.



Dan Yang received her M.S. degree in computer software and theory from Northeastern University of China, Shenyang, in 2004. She is currently a Ph.D. candidate in computer software and theory, Northeastern University. She is a student member of the CCF and member of the ACM. Her research interests include dataspace and data integration.

De-Rong Shen received her B.S. degree and M.S. degree in computer science from Jilin University of China in 1987 and 1990, respectively, Ph.D. degree in computer science from Northeastern University, China, in 2004. She is a professor of Northeastern University. She is a senior member of the CCF and a member of the ACM, IEEE. Her research interests include Web data processing and distributed database.

Ge Yu received his B.S. degree and M.S. degree in computer science from Northeastern University of China in 1982 and 1986, respectively, Ph.D. degree in computer science from Kyushu University of Japan in 1996. He has been a professor at Northeastern University of China since 1996. He is a senior member of the CCF, and a member of the ACM, IEEE. His research interests include database theory and technology, distributed and parallel systems, embedded software, and network information security.

Yue Kou received her Ph.D. degree in computer software and theory from the College of Information Science and Engineering, Northeastern University of China in 2009. She is an associate professor of Northeastern University of China. She is a member of the CCF and ACM. Her research interests include Web search, Web mining, and dataspace.

Tie-Zheng Nie received his Ph.D. degree in computer software and theory from the College of Information Science and Engineering, Northeastern University of China, Shenyang in 2009. He is an associate professor of Northeastern University of China. He is a member of the CCF and ACM. His research interests include data quality and data integration.