Lecture Notes in Artificial Intelligence 7121

Subseries of Lecture Notes in Computer Science

Jie Tang Irwin King Ling Chen
Jianyong Wang (Eds.)

Advanced Data Mining and Applications

7th International Conference, ADMA 2011
Beijing, China, December 17-19, 2011
Proceedings, Part II

Springer

Volume Editors

Jie Tang
Jianyong Wang
Tsinghua University
Department of Computer Science and Technology
Beijing, 100084, China
E-mail: {jietang, jianyong}@tsinghua.edu.cn

Irwin King
The Chinese University of Hong Kong
Department of Computer Science and Engineering
Hong Kong, SAR, China
E-mail: king@cse.cuhk.edu.hk

Ling Chen
University of Technology
Faculty of Engineering and Information Technology
Sydney, NSW 2007, Australia
E-mail: ling.chen@uts.edu.au

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25855-8 e-ISBN 978-3-642-25856-5
DOI 10.1007/978-3-642-25856-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

CR Subject Classification (1998): I.2, H.3, H.4, H.2.8, J.1, F.1, I.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Interactive Predicate Suggestion for Keyword Search on RDF Graphs

Mengxia Jiang[1,2], Yueguo Chen[2], Jinchuan Chen[2], and Xiaoyong Du[1,2]

[1] School of Information, Renmin University of China, Beijing, China
[2] Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
{jmx,chenyueguo,jcchen,duyong}@ruc.edu.cn

Abstract. With the rapid growth of RDF data set, searching RDF data has recently received much attention. So far, structural languages such as SPARQL support to search RDF data efficiently and accurately. Unfortunately it requires users to have the prior knowledge of the underlying schema and query syntax of RDF data set. On the other hand, keyword search over graphs outperforms SPARQL queries in terms of usability. However, the predicate information is ignored in keyword queries, which results in the huge searching space and generates ambiguous interpretation of queries. In this paper, we design an interactive process of keyword search, which allows users to reduce the ambiguity of keywords by selecting some predicates to constrain the semantics of query keywords. We propose an efficient and effective algorithm to online discover a small number of predicates for users to choose. Experiments on the YAGO data set demonstrate the effectiveness and efficiency of our method.

1 Introduction

There is an emerging trend to apply keyword search on the graph-shaped data extracted from the Web. As a W3C-endorsed data model that builds on graphs, RDF is gaining prevalence for resource description and large-scale RDF collections such as DBpedia [4] or YAGO [19], which arouses intensive studies for the efficient storage and querying of RDF data [1,17]. Although SPARQL is a powerful language that search RDF data like SQL on relational database, it only suits for professional users because: firstly, the complex syntax structure of SPARQL [17] prevents ordinary users to learn and handle it easily; secondly, users must know at least part of the RDF schema information (like table name or column name in SQL query) as a prior knowledge of writing SPARQL query, which is usually nontrivial.

From another perspective, an RDF collection can be modelled as a graph, where nodes are subjects/objects, and edge labels are predicates. Keyword search on graph data has attracted much attention for its easy portal to assist users to access the desired information. As introduced in [5], given a directed graph G in which each node is associated with a plain text label, keyword search on graphs concerns with querying G by a set of keywords. The expected answer is a ranked list of substructures which contain the query keywords in their leaf nodes.

J. Tang et al. (Eds.): ADMA 2011, Part II, LNAI 7121, pp. 96–109, 2011.

However, RDF is not merely a directed labeled-graph, which makes keyword search on RDF graphs somehow different. In particular, there are two main differences between RDF graphs and common labeled-graphs:

- The edges in RDF graphs are labeled. An edge label (i.e., a predicate) usually describes the relationship between the corresponding two nodes. Generally, predicates are predefined and specified in certain format. They typically do not support keyword search directly.
- The literal nodes in traditional labeled graphs are typically short labels. However, many RDF labels are long texts that may contain dozens of words. As a result, a large number of subgraphs potentially contain query keywords, which leads to the ambiguity of keyword search.

Predicates can help us to reduce the disambiguation of keywords. For example, "Apple" with the predicate "hasCEO" obviously means a company. The authors of [20] and [6] have observed this and they proposed that predicates should be taken into accounts for keyword search over RDF graphs. However, it is not clear how the predicates are involved in keyword search process. To the best of our knowledge, there are few studies on predicate suggestion for keyword search over RDF graphs. Traditional methods of keyword search on graphs [12,9,13] mainly focus on the labeling literals associated with nodes, without edge information (predicates). The motivation of this paper is, in real scenarios, users who present the keyword query generally have an expectation on the semantics of the keywords. Thus, it will be helpful if a "smart" search engine can suggest a short ranked list of predicates to interpret the user's intention, so that even the user does not know any knowledge about the underlying schema of an RDF data set, he can find the desired information easily by inputting some keywords and associating some predicates to constrain the semantics of keywords.

In this paper, we propose an interactive predicate suggestion for keyword search on RDF graphs. We use an example to illustrate the main idea. Suppose a user inputs a keyword query "Ulm Nobel_Prize", our system will suggest a ranked list of predicates for each keyword to capture the user's intention at moment m_0, e.g., "isCityOf" and "bornIn" for "Ulm", "isCalled" and "hasWon" for "Nobel_Prize". If the user selects "bornIn" for "Ulm", the system will be aware of user's intention by suggesting "hasWon" for "Nobel_Prize" to help user find a person who was born in Ulm and has won Nobel prize. Once the predicate selection is completed, our system will rank all the possible results that contain the issued keywords and the associated predicates.

As a new problem, searching RDF data with predicate suggestion is a challenging work. As an online process, predicate suggestion should be able to efficiently capture users' intention as accurate as possible, even facing RDF graphs with billions of triples. In this paper, we exploit both the semantics and structure of RDF graphs for predicate suggestion. We summarize the main contributions: (1) we shed light on easing keyword search on RDF graphs by interactive predicate suggestion; (2) we propose the prioritized propagation and aggregation (PPA) algorithm to capture user's intention with enhanced performance; (3) we test our method on 17.6 millions YAGO triples to demonstrate the effectiveness and efficiency.

In the rest, Sections 2 and Section 3 give formal problem statement and the framework of our solution. In Section 4, we elaborate the MPA and PPA algorithms for effective and efficient predicate suggestion. Section 5 provides experimental study, followed by the related work in Section 6 and the conclusion in Section 7.

2 Problem Statement

Interactive predicate suggestion for keyword search over RDF data is based on the following two intuitions:

Intuition 1. *Generally, users who input the keywords know the exact semantics of these keywords. However, they do not know how the semantics is presented in RDF graphs because they often do not know the predicates used in RDF data sets. Simply providing more keywords for clarifying the query intention may cause false negatives of matching subgraphs.*

Intuition 2. *The desired results are substructures with a very small radius. In other words, all the querying keywords must appear in literal nodes close to each other in terms of the graph distance of RDF data sets. A small radius guarantees the strong semantic correlations among the mentioned nodes of querying keywords.*

Intuition 1 provides a supporting evidence for interactive predicate suggestion. The query intention of users can be interpreted by keywords associated with predicates, which are obtained from user feedbacks. Intuition 2 reveals the substructures with small radius are preferred, which forms the theoretical basis of the prioritized propagation and aggregation (PPA) approach we propose in the predicate suggestion process.

RDF Graph Formulation. An RDF statement is a subject-predicate-object triple and represented by (S, P, O), which corresponds to an edge from a node S to a node O with edge type P in the RDF graph. An RDF dataset can be abstracted as a graph G. As a general discrimination, vertices in G are partitioned into two disjoint sets: V_L (representing literal nodes) and V_E (entities, typically represented as URIs). A literal node must link to an entity node. For the RDF graph example shown in Fig. 3(a), $p1$ is an entity node, "2010" and "*Paper*" are literal nodes. Note that keywords can only appeared in literal nodes because entity nodes are only URIs, which typically are not meaningful in semantics. We formalize an RDF data graph as follows.

Definition 1 (RDF data graph). *An RDF graph G can be represented as a tuple $(V, L, \phi_V, E, P, \phi_E)$, where*

- *V is a set of vertices and $V = V_L \cup V_E$;*
- *L is a set of node labels;*
- *ϕ_V is a mapping function which assigns each node in V_L by a label in L;*
- *E is a set of edges with form $e(v_1, v_2)$, where $v_1, v_2 \in V$;*
- *P is a set of predicates;*
- *ϕ_E is a mapping function which assigns each edge in E by a predicate in P.*

Input/Output. The interactive predicate suggestion process is partitioned into a number of moments. The initial input at moment m_0 is a set of keywords denoted as $Q_0 = \{K_1, K_2, \cdots, K_n\}$ if the keyword size $|Q| = n$. At each moment, the user will select a predicate from the suggestion list for an unpaired keyword. As time goes by, more and more keywords will be paired with predicates due to user's selection. For example, suppose Q_0={mining,ICDM}. At moment m_1 the user selects "confName" for "ICDM" to identity "ICDM" is a conference name, we get Q_1={mining,(ICDM,confName)} as the input of moment m_1. We formalize the input at any moment m_t as follows:

Definition 2. *Supposing $Q_0 = \{K_1, \cdots, K_n\}$, the input of interactive predicate suggestion at moment m_t is formalized as $Q_t = (Q_0, P_t, \phi_t)$, where*

- *P_t is a list of associated predicates and the size $|P_t| \leq n$;*
- *ϕ_t is a mapping function to associate keywords with corresponding predicates, and*
  - *$\phi_t(K_i) = p \in P_t$ means K_i is associated with a predicate p;*
  - *$\phi_t(K_i) = *$ means K_i is not associated with any predicates.*

The output at moment m_t is a suggestion list of predicates for each un-associated keyword, and we use $S_t(K_i)$ to denote the output for the keyword K_i. For example, *author* and *title* are suggested to keyword *mining* at moment m_1. S_t will be presented to users for predicate selection. It is important to note that at each moment, predicates are finely evaluated and quickly updated to capture the user's intention for un-associated keywords. This interactive suggestion process is the main focus of this paper. It is up to the user to abort the interactive selection phase at any moment, so after the completion of predicate suggestion, it is not necessary for every keyword to be associated with a predicate. In the extreme case, users may reject any predicate suggestion and submit Q_0 as the final input. Once the users finish the process predicate selection at some moment m_t, Q_t will be sent to query execution engine and a list of ranked substructures that fit Q_t will be ranked and returned.

Problem. In interactive predicate suggestion, supposing the input is Q_t at moment m_t, we are concerned with the problem of efficiently updating top-ranked predicates for each keyword that is un-associated with predicates in Q_t to capture user's intention. Let's present this problem in a formal way.

Problem 1. At moment m_t, given the input Q_t, suggest the top-k ranked predicate list $S_t(K_i)$ for any keyword K_i if $\phi_t(K_i) = *$.

Finally, major notations used in this paper are shown in Fig. 1.

3 The Framework

3.1 The Structures of Search Results

In traditional approaches ([5,9,12,10,14]) of keyword search on graphs, the answers are typically tree-shaped substructures (in particular, Steiner trees [5])

$\phi_E(e)$	the predicate on edge $e(v_1, v_2)$
$\phi_V(v)$	the label on node v
$V(K_i)$	keyword nodes that matches K_i
T_{K_i}	keyword node that matches K_i in subtree T
$M(T_{K_i}, K_i)$	matching degree between T_{K_i} and K_i
$\phi_t(K_i)$	The predicate associated with K_i at m_t, or * if not associated

Fig. 1. Notations

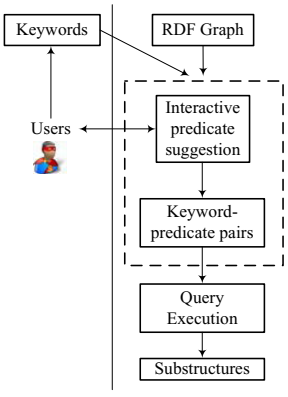


Fig. 2. The framework

where the keyword nodes correspond to leaves. With directed paths from the root to leaves, subtree patterns are meaningful in domains such as XML and relational DB. However, in RDF graphs, we claim the restriction for "directed path" is not necessary. The main reason is, edge directions are generally invertible in RDF statements, e.g., (Alice, eat, Apple) can also be expressed as (Apple,isEatenBy,Alice), making the edge direction in RDF graphs less meaningful. Thus, we consider the resulting substructures as undirected subtrees. In particular, two kinds of nodes are distinguished in an undirected subtrees:

1) **Keyword nodes**, which contain query keywords;
2) **Spanning nodes**, which connect all keyword nodes together.

Recalling Definition 1, keyword nodes are literal nodes, and spanning nodes can only be entities (represented using URIs). The root node is a special spanning node and generally considered as the answer node for the keyword query. According to Intuition 2, the resulting subtree should be compact with small radius. We therefore define the paths from the root to leaves as the shortest paths of the nodes. This guarantees that the resulting subtree to be a minimum spanning tree (MST). If a spanning node is removed, the subtree will be disconnected. Two examples of answering subtrees for query "ICDM mining" are shown in Fig. 3(c).

3.2 Framework of Interactive Predicate Suggestion

Next we explain the framework process of interactive predicate suggestion for keyword search on RDF graphs which is showed in Fig. 2. The initial input of users at moment m_0 is a set of keywords. Immediately after that, the predicate suggestion module will be invoked. For each keyword, a ranked list of predicates will be suggested to interpret the semantics of this keyword. Then at moment m_1, users can select a predicate for a keyword that mostly captures his intention for that keyword. This selection will be reflected in the updating of predicate lists

for the remaining un-associated keywords by the predicate suggestion module. Basically, the predicate suggestion module will be invoked for each moment. This interactive process goes on until all keywords are associated with predicates, or users decide to finish this process at any moment. Finally, keywords and selected predicates will be submitted to the query execution engine. The output of the execution engine is a ranked list of connected subtree which contains all query terms in its node labels.

3.3 Relevance Evaluation

Now we discuss the relevance heuristics for the resulting substructure. In recent works, Many heuristics [8,5,13,10] have been proposed to evaluate the relevance scores of matching subtrees for keyword search over graphs. Based on them, we design three general components to evaluate the relevance of answers in RDF graphs: (1) the similarity between keywords and the literal of their matched keyword nodes; (2) the importance of root node T_r and (3) the distance from the root to keyword nodes. To integrate these components together, we use the idea of match propagation and aggregation (will be explained in Section 4) and consider the score of a subtree as the sum of matching degree propagating from leaf nodes to the root, decayed by a factor $0 < C < 1$ and weighted by the importance of root node. That is,

$$Score(T) = I_{T_r} \cdot \sum_{K_i \in K} M(T_{K_i}, K_i) C^{T_{p_i}} \tag{1}$$

where T_{K_i} is the keyword node that matches K_i in T and $M(T_{K_i}, K_i)$ is the similarity of keyword match, quantified by popular similarity measures such as Jaccard Coefficient [15]. T_{p_i} is the distance from T_r to T_{K_i}. I_{T_r} reflects the importance of root T_r in the whole graph, which can be evaluated by popular authority ranking methods like PageRank [16].

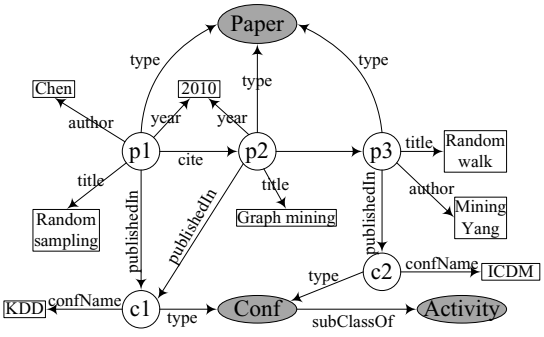
4 Prioritized Propagation and Aggregation

In this section, we first propose the *match propagation and aggregation* (MPA) algorithm, which provides a basic methodology for predicate suggestion. To make MPA more practical in real applications, we further propose an alternative of MPA, called the *prioritized propagation and aggregation* (PPA) algorithm, to leverage the performance of MPA.

4.1 MPA

To illustrate the main idea, we start from an example.

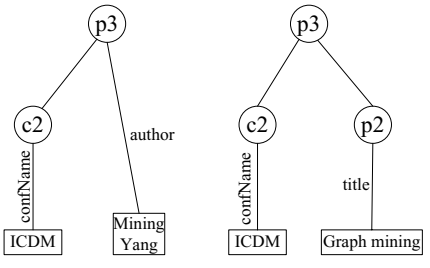
Example 1. In Fig. 3(a), suppose the query is "mining random" and at moment m_1, 'random" is associated with predicate "title". How can we suggest predicates for keyword "mining"? Notice that "mining" may be a part of an author name or title name.



(a)

2010	1
Activity	2
Chen	3
Conf	4
Graph mining	5
ICDM	6
KDD	7
Paper	8
Random walk	9
Stream mining	10
Mining Yang	11
c1	12
c2	13
p1	14
p2	15
p3	16

(b)



(c)

	...	12	13	14	15	16
1	...	2		1	1	
2	...	2	2			
3	...	2		1	2	
4	...	1	1	2	2	2
5	...	2		2	1	2
6	...		1			2
7	...	1		2	2	
8	...	2	2	1	1	1
9	...		2		2	1
10	...	2		1	2	
11	...		2		2	1
12	...		2	1	1	
13	...	2			2	1
14	...	1			1	2
15	...	1	2	1		1
16	...	2	1	2	1	

(d)

Fig. 3. (a) An example of RDF graph. (b) Sorted node numbering. (c) Two answering subtrees for query "ICDM mining". (d) A fragment of shortest distance matrix D.

Intuitively, the keyword matching similarity and the distance of nodes should be both considered in the predicate ranking. However, how to combine them adequately is a problem. In this paper, we rank predicates in a uniform framework called match propagation and aggregation. In simple words, we consider each keyword node has an authority since it matches the keyword, and this matching degree will propagate to other nodes via the selected predicate edges. A predicate of an un-associated keyword will be ranked by the matching traffic that goes through this predicate. In particular, during the predicate suggestion process, keyword nodes have two kinds of roles, as explained below.

Propagator: At moment m_t, if a keyword K_i is associated with predicate p, keyword nodes that match K_i are propagators and the matching degree can be propagated to other nodes via edge e with $\phi_E(e) = p$.

Aggregator: At moment m_t, suppose keyword K_j is not associated with any predicate, i.e., $\phi_t(K_j) = *$, keyword nodes that match K_j are aggregators and will receive the matching degrees from their neighbors.

Basically, at the node level, matching degree will flow from propagators to aggreagators. At the predicate level, the aggregated matching degree on aggregators

via predicate p will be taken as the basis to rank p. In Example 1, nodes matching "random" are propagators and nodes matching "mining" will receive the matching degree from propagators via edge type "title".

If K_j is un-associated and p is one of its predicate, now we compute the matching degree traffic of p and let $M_p(K_j)$ to denote it. Recalling that T_{p_j} is the shortest distance from the root to leaf node that matches K_j, we have

$$M_p(K_j) = \sum_{T'} \sum_{\phi_t(K_i) \neq *} M(T'_{K_i}, K_i) \cdot M(T'_{K_j}, K_j) \cdot C^{T'_{p_i} + T'_{p_j}} \tag{2}$$

where T' is the subtree T that satisfies T_{K_j} is connected by edge with predicate p. $\phi_t(K_i) \neq *$ actually indicates the propagators that are associated in Q_t.

Why $M_p(K_j)$ works to rank predicate p of keyword K_j? The reason is, $M_p(K_j)$ captures the matching degree traffic from all propagators to T_{K_j} through edges with predicate p. Higher $M_p(K_j)$ means that the matching degree of keywords is higher and the paths from the root to leaf nodes are shorter, which results in a subtree ranked higher by scoring function shown in Equation 1.

4.2 PPA

MPA is a kind of random walk algorithms [3], where a random surfer starts from the propagators and visits their neighboring entities recursively. If an entity is visited, it will become active and can influence its neighbors at the next step. Further, we use "frontier" to describe the set of entities that are activated at current step. For example, {p2,c2} is the frontier of keyword node "random walk" at 2 steps in Fig. 3(a). Recalling that the output subtrees are MSTs, the frontier $F(i)$ at step i is actually the nodes reached from propagators by shortest paths with exact i steps, and we denote it as $F(i) = \{x | D(v, x) = i\}$, where v is a propagator and D is the shortest distance matrix. A fragment of D with maximal distance $d_{\max} = 2$ is shown in Fig. 3(d).

However, MPA may face performance issues in real RDF graphs, since some nodes are of very high degree and the number of visited nodes increases exponentially during the flooding process. Also, if the amount of propagators are huge, the time and space cost will become unaffordable for the predicate suggestion process. $|F(i)|$ may easily go up to hundreds of thousands. Thus, it becomes necessary to prioritize the nodes in the same frontier and only propagate the nodes with higher authority. In detail, for a node v in $F(i)$, the propagators that reach v on the i-th step can be denoted as a set $S(v) = \{s | D(s, v) = i\}$. Thus, all nodes in the frontier $F(i)$ like v can be ranked by

$$PPA(v) = \sum_{s \in S(v)} M(T_{K_s}, K_s) \cdot C^i \tag{3}$$

After that, only nodes of top-k highest scores will be further propagated in $F(i)$. An important problem is how to decide the parameter k. The most simple way is setting k by a threshold, which is either a fixed number or a fixed ratio. However, the fixed number (e.g., top 100) cannot reflect the size of the frontier. The fixed ratio (e.g., top 30%) may exceeds the total time budget if the size

of frontier is extremely huge. Thus, we propose a more flexible but efficient strategy by combining them together using a piecewise linear function with factor (*min*,*max*,*ratio*), where *min* and *max* are the lower and upper bound of k and *ratio* decides the portion of k out of all candidates. Formally, let $|F(i)|$ be the size of the frontier, and we decide k by

$$k = \begin{cases} |F(i)|, \text{ if } |F(i)| \leq min \\ min, \text{ if } |F(i)| > min \text{ and } |F(i)| \cdot ratio < min \\ |F(i)| \cdot ratio, \text{ if } min \leq |F(i)| \cdot ratio < max \\ max, \text{ if } |F(i)| \cdot ratio > max \end{cases} \tag{4}$$

4.3 Interactive Predicate Suggestion

The steps of interactive predicate suggestion is summarized in Algorithm 1. Line 1-12 are the process of initialization, which involves mapping the keywords to nodes in RDF graph and finding entities within $d_{\max}$ steps based on prioritized propagation and aggregation. Line 13-18 are the first predicate suggestion to users at moment m_0. Since no keywords are associated with predicates at this moment, the suggestion of predicate p is basically a statistics of p occurring in subtrees. Line 19-27 are steps for predicate suggestion at moment m_t with $t \geq 1$. Here, some of keywords are associated with predicates and serve as propagators, whose matching degree will be propagated to other keywords that are un-associated. The predicates associated with aggregators will be ranked by the matching degree traffic flowing through them, as computed in Line 25.

For time and space complexity, supposing the keyword node set size is N and the average degree is $\frac{2|E|}{|V|}$ (for undirected graphs), the visited node size is $O(nN(\frac{2|E|}{|V|})^{d_{\max}})$ for MPA. MPA faces performance problems when N or edge degree are unusually huge. Thus, it is our recommendation to replace MPA by PPA, which limits the upper bound of the visited node size as $O(n \cdot \max)$.

5 Experimental Study

In this section, we instantiate the interactive predicate suggestion on YAGO dataset to evaluate the effectiveness and performance. Without loss of justice, our method is compared with other existing methods like traditional keyword search on graphs. All experiments are implemented by Java, and running on a computer with 1.86GHz CPU and 4GB main memory.

5.1 Data Set and Query Set

As a huge collection of RDF facts, YAGO(1) dataset[1] contains 17,631,348 statements, 8,979,923 unique subjects and 4,654,016 unique objects. We map all RDF statements into edges and finally builds an RDF graph with 10,796,073 nodes and 92 different kinds of edge types (predicates). To index these data and respond

[1] http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html

Algorithm 1. Interactive Predicate Suggestion

Input: RDF graph $G, C, d_{\max}$, query $\{K_1, \cdots, K_n\}$
Output: Suggestion lists S_t and the answer subtrees
`// Offline Part:`
for *each keyword* K_i **do**
  find keyword node set $V(K_i)$ that matches K_i in $V_A \cup V_C$;
  $D(K_i) = \emptyset$;
  for *each keyword node* $v \in V(K_i)$ **do**
    for *step* $i = 1 : d_{\max}$ **do**
      if $D(v, v') = i$ *and* $v' \in V_E$ **then** add v' into $D(K_i)$;
      only expand top k entities in $D(K_i)$ according to Equation 4;
$Root = D(K_1) \cap \cdots \cap D(K_n)$;
$TreeSet = \emptyset$;
for *each entity* $r \in Root$ **do**
  generate subtrees rooted by r and having leaf nodes in $V(K_1), \cdots, V(K_n)$;
  add these subtrees into $TreeSet$;
`// Online Part:`
set $t := 0$;
for *each keyword* $K_i \in \{K_1, \cdots, K_n\}$ **do**
  for *each predicate* p *associated with* K_i **do**
    $S_0(K_i, p) = \sum_T M(T_{K_i}, K_i) \cdot C^{T_{K_i}}$ where $T \in TreeSet$ and T_{K_i} is connected by edge with p;
  $S_0(K_i) =$ the ranked list of $S_0(K_i, p)$;
present S_0 to the user;
while *not interrupted by the user and* $t < n$ **do**
  $t := t + 1$;
  select a predicate for a keyword and build input Q_t;
  remove subtrees not consistent with Q_t from $TreeSet$;
  for *each unpaired keyword* $K_i \in Q_t$ **do**
    for *each predicate* p *associated with* K_i **do**
      $M_p(K_j) = \sum_{T'} \sum_{\phi_t(K_i) \neq *} M(T'_{K_i}, K_i) \cdot M(T'_{K_i}, K_i) \cdot C^{T'_{p_i} + T'_{p_j}}$ where $T' \in TreeSet$ and T'_{K_i} is connected by edge with p;
    $S_t(K_i) =$ the ranked list of $S_t(K_i, p)$;
  present S_t to the user;
rank subtrees that are consistent with Q_t in $TreeSet$ and return;

fast for user queries, we use Compressed Row Storage (CRS, [18]) to store them in a compact way, where a matrix is transformed into three vectors: non-zero values, column indices and row starting indices. Terms are obtained by splitting node labels by "_" or blank, and a total # of 3,430,600 terms are included.

Two distinctly different keyword query sets QS_1 and QS_2 are used: QS_1 contains popular keywords with average keyword node size $|V(K_i)| = 4772$ (example: "movie Italian") while QS_2 contains less popular keywords with average keyword node size $|V(K_i)| = 63$ (example: "Deborah Exterminator"). How do we get these query sets? Firstly, we randomly produce many queries. After getting these queries, we calculate two measures: the number of keyword nodes, and the number of neighbors for keywords in each query. With the information we filter unsatisfied queries and separate the remaining into two groups,that is QS_1 and QS_2. QS_3 is produced similarly. Each query in QS_3 contains 4 keywords. The average keyword node size of QS_3 is 5987, and average root size is 40.

5.2 Quality Evaluation

Because keywords may be matched to a huge number of literal nodes (called keyword nodes) and some nodes may have very high degree, the propagation and aggregation will be extremely expensive. In these cases, we propose to use PPA instead of MPA for reducing the computational cost. The generated subtrees by PPA are actually a subset of subtrees by MPA. Therefore, every resulting subtree in PPA is an correct match to the keyword query. Thus, we only measure the recall of subtrees generated by PPA. To evaluate the quality of predicate lists suggested by our method, we further use intention index, which is defined as follows: for a keyword K_i, supposing the predicate in user's mind is p for K_i, we examine the average ranking position of p in the suggested list. The smaller the intention index of p, the better this suggestion list is.

We fix $ratio = 50\%$ in top-k factor $(min, max, ratio)$. $min - max$ pair is set in different scales to reflect k's influence on the recall. The details are shown in Fig. 4(a). As we can see, with the enlargement of $min - max$ pairs, the recall on both QS_1 and QS_2 increases steadily. But obviously, QS_2 has a significantly higher recall than QS_1, and the reason is: given that in MPA the average root number in QS_1 and QS_2 are 26276 and 21 respectively, the truncation error in PPA by the same top k has a heavier damage to the recall in QS_1. Thus, max tends to be set as a high number (typically $max = 10000$) but a too high max is not necessary. As showed in Fig. 4(a), $max = 10000$, even the recall is only 0.118, the average returned root number in QS_1 is more than 3000. Users only pay attention to a small number of subtrees ranked in the top of results.

We test the intention index of predicate lists suggested by MPA, PPA ($min = 1000, max = 10000$) and frequency-based statistics which rank predicates only based on their appearance. The predicate in user's mind for a keyword is judged by human testers (an average of 5 testers in our experiments). Fig. 4(b) shows PPA has a slightly worse intention index than MPA esp. on QS_1, which accords with our assumption that PPA is an approximation of MPA, but with an enhanced performance.

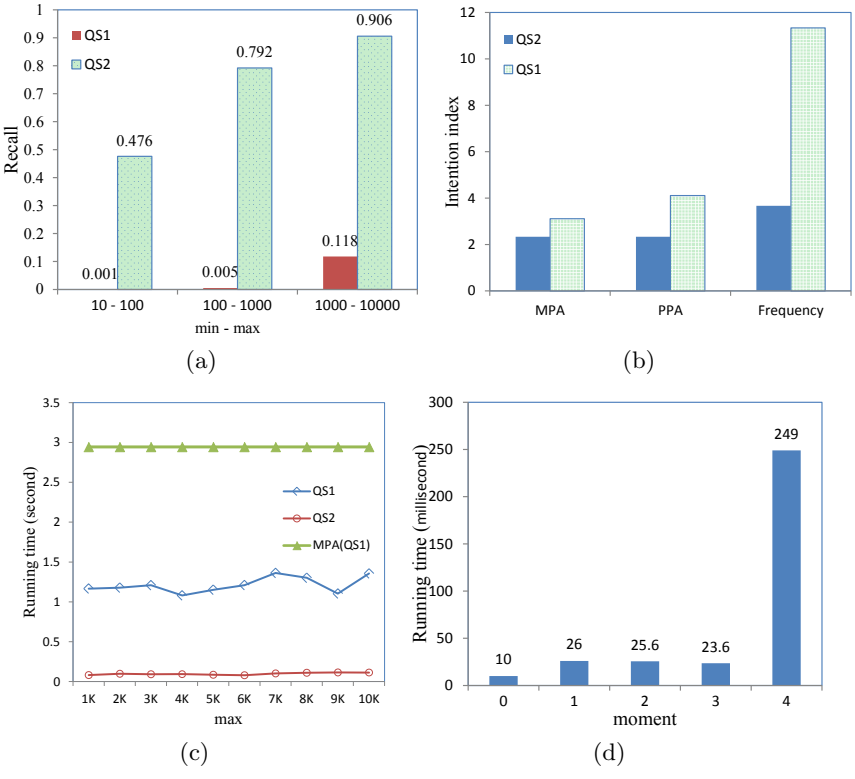


Fig. 4. (a) Recall based on different top k scales. (b) Intention indices of predicate suggestions by MPA, PPA and frequency. (c) The offline running time on different top k scales with $min = 0.1max$. (d) The running time of QS_3 at different moments.

5.3 Performance Evaluation

The predicate suggestion in our method can be divided into offline part and online part. Offline part includes the construction of inverted index between keywords and nodes in the RDF graph, and also the routing of nearby neighbors by shortest distance paths. The performance of PPA using different top k prioritizing strategies and query sets is shown in Fig. 4(c). We only show the timeconsuming of MPA for QS_1 because it is more expensive when there are a huge number of keyword nodes, and therefore suitable for applying PPA. Although going up slightly with fluctuation, the offline running time for QS_1 and QS_2 is around 1 and 0.1 second respectively under different scales of max ($min = 0.1max$).

For online part, we evaluate the performance of predicate suggestion on each moment. QS_3 is used. The results are shown in Fig. 4(d) with setting $max = 10000$. At moment m_0, because no keywords are associated with predicates, PPA is not necessary and a quick frequency-based suggestion is performed. At moment m_1 to m_3, the number of associated keywords increases one by one, which means that #propagators is increasing and #aggreagators is decreasing.

The trend of time cost is slightly descending over these moments. At moment m_4, all the resulting subtrees are generated and ranked, with a time cost relatively higher than the suggestion phases. Overall, the response of predicate suggestion by PPA is rapid even on huge datasets like YAGO and sufficient for online usage.

6 Related Work

RDF is a W3C-endorsed data model that gains popularity in various applications. More and more large-scale RDF collections become available on the Internet, such as DBpedia [4] or YAGO [19], and attract much attention in the academia and industry. Generally, there are two different categories of methods for searching desired information in RDF data. The first category is from the view of database, where RDF statements are stored in SPO triples. The work [1] proposed an impressive RDF storage by vertical partitioning. SPARQL is the W3C standard language for querying RDF. Similar to SQL, it is a structural language that requires users to know the knowledge about the schema and query syntax. Another category of methods tries to avoid this drawback, where users simply need to input some keywords. These methods are based on the view that an RDF dataset can be considered as a graph with each statement corresponding to an edge. In the academia, keyword search on graphs is extensively studied [9,13,12,7,2]. While effective in graphs where edges have no types, it is problematic in the case of RDF graphs, because each edge in the RDF graph is associated with a predicate to indicate the semantics of entities and nodes.

We try to adopt both the simplicity of inputs in keyword search and the awareness of predicates in SPARQL query in this paper. We achieve this goal by adding a predicate selection process, and for each keyword, the suggested predicates will be dynamically ranked to reflect user's intention. Predicate suggestion is not a well-studied area. To achieve a better accuracy and efficiency, we suggest predicates by prioritized propagation and aggregation techniques, which is inspired by authority-based search [11] and bidirectional search [12].

7 Conclusion

In this paper, we propose an effective searching method to answer keyword queries over RDF graphs. Our method takes the advantage of keyword search (easy usage) and that of SPARQL query (better semantic discrimination) by associating predicates with keywords. This is achieved by allowing users to select predicates based on the suggested predicates of keywords. To make predicate suggestion more precisely and efficiently, we adopted the idea of propagation and aggregation of users' preferences. We also examine the issues of keyword search by taking both semantic and structure information into accounts.

Acknowledgments. This work is partially supported by HGJ PROJECT 2010ZX01042-002-002-03, the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China under grant No. 20100303614.

References

1. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable semantic web data management using vertical partitioning. In: VLDB, pp. 411–422 (2007)
2. Achiezra, H., Golenberg, K., Kimelfeld, B., Sagiv, Y.: Exploratory keyword search on data graphs. In: SIGMOD Conference, pp. 1163–1166 (2010)
3. Aldous, D., Fill, J.: Reversible markov chains and random walks on graphs. Materials, `http://www.stat.berkeley.edu/aldous/RWG/book.html`
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A Nucleus for a Web of Open Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
5. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using banks. In: ICDE, pp. 431–440 (2002)
6. Elbassuoni, S., Ramanath, M., Schenkel, R., Weikum, G.: Searching rdf graphs with sparql and keywords. IEEE Data Eng. Bull. 33(1), 16–24 (2010)
7. Golenberg, K., Kimelfeld, B., Sagiv, Y.: Keyword proximity search in complex data graphs. In: SIGMOD Conference, pp. 927–940 (2008)
8. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked keyword search over xml documents. In: SIGMOD Conference, pp. 16–27 (2003)
9. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: ranked keyword searches on graphs. In: SIGMOD Conference, pp. 305–316 (2007)
10. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: VLDB, pp. 850–861 (2003)
11. Hristidis, V., Hwang, H., Papakonstantinou, Y.: Authority-based keyword search in databases. ACM Trans. Database Syst., 33(1) (2008)
12. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB, pp. 505–516 (2005)
13. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD Conference, pp. 903–914 (2008)
14. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: SIGMOD Conference, pp. 563–574 (2006)
15. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
16. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999)
17. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Trans. Database Syst., 34(3) (2009)
18. Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics (2003)
19. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW, pp. 697–706 (2007)
20. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: ICDE, pp. 405–416 (2009)