JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 551-574 (2018)
DOI: 10.6688/JISE.201803_34(2).0015

Reducing Redundancy in Keyword Query Processing on Graph Databases

CHANG-SUP PARK
Department of Computer Science
Dongduk Women's University
Seoul, 02748 Korea
E-mail: cspark@dongduk.ac.kr

In this paper, we propose a new approach to reducing redundancy in the answers to a keyword query over large graph databases. Aiming to generate query results which are not only relevant but also has diverse structures and content nodes, we propose a method to find top-*k* answer sub-trees which should be in reduced forms and duplication-free in regard to the set of content nodes. To process keyword queries efficiently over large graph data, we suggest an efficient indexing scheme on the most relevant paths from nodes to keyword terms in the graph. We present a top-*k* query processing algorithm which exploits the pre-constructed indexes to search for a set of most relevant and non-redundant answers. We also provide a state space search algorithm to find most relevant duplication-free answers in an efficient way. We show effectiveness and efficiency of the proposed approach in comparison with the previous methods using extensive experiments on real graph datasets.

Keywords: graph database, keyword query, top-*k* query processing, redundancy, indexing scheme

1. INTRODUCTION

Recently, graph-structured data is widely used in various fields such as social networking, semantic web, linked open data, knowledge management and bio-informatics. A relational database also can be modeled as a directed graph based on foreign-key relationships existing among tuples. A graph database consists of nodes and edges, which represent relationships between entities effectively. As the amount of graph data increases rapidly, an efficient and effective query system is much in need. Keyword search has been attracting a lot of attention since it provides a simple and user-friendly interface to querying graph data and allows users to express their information need using only a set of keyword terms [2, 6, 7, 11, 12, 14, 15, 17, 20, 21, 24].

Keyword search on graph data usually returns a set of connected sub-structures, such as sub-trees or sub-graphs, showing that which nodes include query keywords and how they are inter-connected in the graph database. Many approaches find minimal connected sub-trees containing query keywords as succinct answers to a given query [2, 6, 7, 11, 12, 14]. Since there can be a significant number of answer sub-trees in a large graph database, a relevance scoring function is often used to rank candidate answers and select top-*k* ones having the highest relevance. There have been proposed several approaches based on *distinct root semantics*, where for each node in the graph, at most one sub-tree rooted at the node is considered a possible answer to the query [6, 12, 14, 20]. The an-

Received July 4, 2016; revised October 30, 2016; accepted January 14, 2017.
Communicated by Hsin-Hsi Chen.

swer tree consists of a set of content nodes containing all the query keywords as well as the nodes and edges on the shortest paths from the root to each content node. Its relevance is usually computed by a function of the shortest paths, such as the sum of the path lengths. By reducing the number of sub-trees to be explored in the graph significantly, the search methods based on the distinct root semantics can process keyword queries over a large volume of data more efficiently than other approaches. It also facilitates exploiting indexes on graph data to improve query performance [12].

However, the previous methods have a common limitation; they can produce two kinds of ineffective answer trees called a *non-reduced* tree and a *duplicate* tree. The former is an answer tree where the root node has only a single child node and contains no query keyword. Note that a non-reduced answer tree always contains at least one *reduced* sub-tree which shares keyword nodes while it provides no more useful information than the reduced sub-tree. For example, consider a directed weighted graph G in Fig. 1, where nodes v_4-v_9 contain keyword terms and edges are labeled with a weight value indicating distance between adjacent nodes. Given a keyword query $q_1 = \{k_1, k_2, k_3\}$ over graph G, 6 trees shown at the right of G can be answers to query q_1 since they are sub-trees of G which have all the keywords in q_1 in their nodes. Note that T_2 and T'_2 are rooted at the same node v_2 while having different nodes for keyword k_3, *i.e.* v_8 and v_5, respectively. Since the distance from v_2 to v_8 is shorter than that from v_2 to v_5, conventional search methods based on the distinct root semantics usually select T_2 as the answer tree rooted at v_2. However, it should be noted that T_2 is in a non-reduced form and has a smaller reduced answer tree T_1 as a sub-tree, while T'_2 is a reduced answer tree. Assuming that T_1 is included in top-k query results earlier, selecting T'_2 makes the search results more diverse than choosing T_2 even though T'_2 has a lower relevance score than T_2. Similarly, it is also desirable to choose a reduced sub-tree T'_3 instead of a non-reduced sub-tree T_3 as an answer tree rooted at node v_1.

The other ineffectiveness of the previous approaches is that their search results may have a lot of answer trees containing the same set of content nodes for given keywords. In many applications of keyword search on graph data, users often have interest in finding different sets of content nodes which cover all the keywords in the query and are closely related to each other. For instance, in keyword search on the Web, users want to find different sets of Web pages that are close to each other and might not be interested in browsing multiple relations to see how the Web pages are related to each other [16]. However, a large amount of similar answers with a duplicate set of content nodes are often found in search on the Web, as well as in real graph data such as the Internet Movie Database and DBLP computer science bibliography. Thus, it is more desirable to find a set of relevant sub-trees containing a distinct set of content nodes than to retrieve a lot of sub-trees having duplicate content nodes with different connecting structures.

For example, given a query $q_2 = \{k_1, k_3, k_4\}$ over G in Fig. 1, a set of top-3 answers based on the distinct root semantics can be $\{T_4, T_5, T_6\}$. Note that these sub-trees share the same set of content nodes $\{v_4, v_5\}$ even though they have a different root node and connecting structure. Given that T_4 is the most relevant answer with the highest relevance score, the other sub-trees T_5 and T_6 are called *duplicate* with T_4 regarding content nodes. Meanwhile, if we select sub-trees T'_5 and T'_6 instead of T_5 and T_6, the set of answer trees $\{T_4, T'_5, T'_6\}$ are duplication-free and thus can provide more diverse results for q_2. We also observe that in a set of reduced answer trees $\{T_1, T'_2, T'_3\}$ for query $q_1 = \{k_1, k_2, k_3\}$ over G, T'_3 has the same set of content nodes as T'_2. Therefore, it should be replaced with T''_3 in order to avoid duplication in the sets of content nodes of the answer trees.

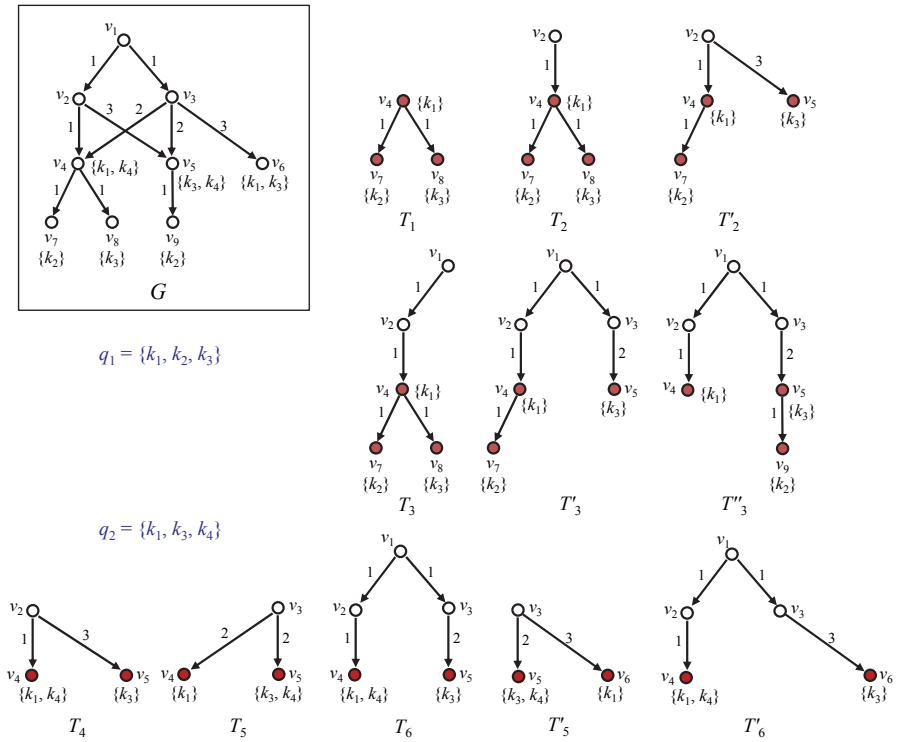


Fig. 1. Answers to example keyword queries q_1 and q_2 over graph data G.

Top-k search results including many redundant answer trees such as non-reduced and duplicate trees have drawbacks. First, a lot of similar answers decrease diversity of the search results and they do not satisfy users who want to get various answers rather than similar ones. Second, if an answer tree in the query results turns out to be irrelevant to the query, most of the non-reduced or duplicate answer trees related with it would be also irrelevant to the query. It can degrade the quality of the query results severely.

In this paper we propose a new approach to reducing redundancy in keyword search over graph databases and producing answer trees not only minimal and relevant to the query but also diverse in their structures and content nodes. Main contributions of the paper are as follows:

- We suggest a non-redundant result structure for keyword queries over graph databases which includes only reduced and duplication-free sub-trees.
- We present an effective indexing scheme on a subset of useful paths from nodes to keywords in graph databases.
- We propose an efficient top-k query processing algorithm using the path indexes to find most relevant and non-redundant answer trees.
- The effectiveness and efficiency of the proposed approach are evaluated with extensive experiments on real graph databases.

The rest of the paper is organized as follows. Section 2 describes related work on

keyword search over graph data, and Section 3 defines an answer tree structure and a relevance measure for the answer trees. In Section 4, we propose three kinds of indexes on the paths in the graph. In Section 5, we describe an efficient query processing algorithm to find top-*k* non-redundant answer trees using indexes and a state space search strategy. We demonstrate performance of the proposed method by extensive experimental results in Section 6 and draw a conclusion in Section 7.

2. RELATED WORK

Most previous approaches to keyword search on graph data find minimal sub-trees containing query keywords [2, 6, 7, 11, 12, 14, 20, 24] while some recent studies propose to search for sub-graphs to provide more informative answers [15, 19, 21]. In this paper we adopt a tree-based approach to provide users with succinct answers to a query as well as to support efficient query processing over a large volume of graph databases.

Most approaches searching for sub-trees in the graph are based on two different semantics, namely Steiner tree semantics and distinct root semantics [27]. The Steiner tree semantics defines the weight of an answer tree by the total weight of the edges in the tree. Search methods based on this semantics aim to find answer sub-trees with the smallest weights. However, finding only an optimal answer sub-tree with the smallest weight, called a group Steiner tree, is known to be NP-complete [13]. Under distinct-root semantics, sub-trees returned as query results must be rooted at a distinct node. Thus, for each potential root node in the graph, only a single sub-tree having a minimal weight is considered a candidate answer to the query, where the weight of a sub-tree is typically defined by the sum of the weights of the shortest paths from its root to each keyword node. This semantics can deal with queries over a large graph database more efficiently than Steiner tree semantics [27].

BANKS [2] presents a keyword search strategy performing backward expansion of the graph starting from the set of nodes containing query keywords to find relevant answer sub-trees under the Steiner tree semantics. The Bi-directional Search proposed in [14] is based on distinct root semantics and enhances BANKS by allowing forward expansion from a potential root of an answer tree toward keyword nodes in addition to backward exploration. However, it does not take advantage of any prior knowledge on the graph and depends on a heuristic activation strategy hence it shows poor performance on large graph databases. BLINKS [12] uses an efficient indexing scheme on graph data to speed bi-directional exploration with a good performance guarantee. It pre-computes the shortest paths and their distances from nodes to keywords in the graph and stores them in sorted inverted lists and a hash map. By exploiting indexes, it can avoid a lot of explorations of the graph data and thus can find top-*k* answers efficiently. A study in [6] suggests creating and utilizing a multi-granular representation of a graph data to minimize disk I/O, and presents search algorithms on multi-granular graphs extended from BANKS and Bi-directional Search. A recent work in [20] has proposed an extended answer tree structure and search algorithm to produce various and effective top-*k* answers.

These approaches, however, have a common drawback of producing sub-trees that are non-reduced or duplicate in content nodes as mentioned in the previous section. Although graph exploration approaches such as BANKS and Bi-directional Search can de-

tect and exclude such redundant answers, an exponential number of answer sub-trees should be probed in the graph, resulting in severe performance overhead. BLINKS does not take redundancy among query answers into account and creates indexes only on the single optimal path from each node to a keyword term in the graph. Therefore, even if a redundant answer tree is detected, it cannot find alternative answer trees rooted at the same node as the root of the redundant answer. For example, BLINKS cannot produce answer trees T'_2, T'_3, T'_5, and T'_6 in Fig. 1 as the alternatives to the redundant answers T_2, T_3, T_5, and T_6, respectively.

The problem regarding non-reduced answers and structural similarity in search results has been discussed in [11], but they suggest a search algorithm to enumerate answer sub-trees under the Steiner-tree semantics in an approximate order of their heights with polynomial delay. A recent study in [16] deals with duplication of content nodes in the answers to a keyword query. Their approach is different from ours in that it focuses on enumerating top-k duplication-free sets of content nodes in an approximate order of proximity of the nodes. It only finds a set of content nodes that covers input keywords as an answer to a query while our approach produces a sub-tree which exhibits relationships among the content nodes. Even if the other method is employed to retrieve an optimal sub-tree connecting the content nodes, sub-trees derived from different answers might share the same root node. Moreover, the proximity measure to evaluate the rank of the answers might not accurately represent the relevance of the sub-trees to the given query. The enumeration algorithm produces top-k duplication-free answers one by one with polynomial delay. Even though it exploits a pre-built index on the shortest distance between every pair of nodes and computes the proximity of the answer approximately, it is not practical and scalable for large amount of graph data.

Diversifying search result has been studied in the literature of information retrieval, Web search, and recommender systems [1, 5, 8, 22, 23, 25, 26, 31]. However, it is not straightforward to adopt those approaches to keyword search over graph data since the structures of underlying data and query answers as well as relevance measures are different from those of the previous applications.

On the other hand, spatial keyword search is extensively studied in the literature of spatial databases. It aims to find spatial objects which contain a set of keyword terms and are closest to a specified query location. Most approaches propose to use index structures which are usually based on a spatial indexing scheme such as R-tree and the inverted list index for text retrieval [4, 10, 18, 28, 29, 30]. [10] proposes an Information Retrieval R-Tree which combines R-Tree with text signatures to answer top-k spatial keyword queries. [29] uses a quad-tree structure to decompose data space into hierarchical cells and proposes a scalable integrated inverted index to manipulate spatio-textual information efficiently. [30] also proposes an inverted linear quad-tree indexing scheme based on the inverted index and the linear quad-tree to effectively organize spatio-textual objects and a top-k spatial keyword search algorithm. [18] considers a direction constraint in spatial keyword queries and proposes an effective direction-aware index structure to prune search space in unnecessary directions. Some approaches consider finding a set of objects that cover the query keywords collectively. [28] focuses on the problem of m-closest keyword search which retrieves a set of closest objects matching m keywords. In addition, [4] studies the problem of collective spatial keyword search to find a group of spatial objects that contain query keywords and are nearest to the query location.

However, the spatial keyword search approaches mentioned above are inherently different from the keyword search over graph-structured data because the spatial objects typically do not have relationships represented by a graph, the query includes a specific query location as well as keywords, and the query answers are usually single objects that are not only relevant to the keywords but also closest to the query location. Even though some approaches suggest a method to find spatially related objects as a collective answer to the query, they do not have a sub-tree or a sub-graph structure. Therefore, it is difficult to adopt the indexes and algorithms proposed by the spatial keyword search methods to the problem of keyword search on graph data. Moreover, redundancy in top-k answers has not been addressed in the previous approaches to spatial keyword search.

3. PROBLEM DEFINITION

A data graph $G(V, E)$ is a directed weighted graph where nodes in V contain keywords and edges in E have a weight representing distance between two incident nodes. The nodes containing a keyword k are called *keyword nodes* or *content nodes* regarding k and the set of those nodes is denoted by $V(k)$. The length of a directed path between two nodes in G is defined as the sum of the weights on the edges in the path. Based on the general scheme suggested in [27], we define an answer to a keyword query over graph data as follows.

Definition 1 (An answer to a keyword query). Given a graph $G(V, E)$ and a query $q = \{k_1, k_2, \ldots, k_l\}$ over G, an answer to q is a sub-tree T of G which contains a multiset $C = \{v_1, v_2, \ldots, v_l\}$ of keyword nodes where $v_i \in V(k_i)$ $(1 \le i \le l)$ and satisfies the followings: (a) T contains the shortest path from its root to each node in C, (b) all the leaf nodes of T belong to C, and (c) if the root of T has only one child, it also belongs to C.

We denote an answer tree having a root node n and a multiset C of keyword nodes by $T(n, C)$. The shortest path from root n to a keyword node v_i in C is called a *root-to-keyword path* for k_i and denoted by $n \rightarrow k_i$ or $n \rightarrow v_i$. The conditions in Definition 1 specify that answer trees should only have the nodes that are necessary and sufficient to connect their content nodes. In particular, condition (c) requires that answer trees should be *reduced*, *i.e.*, the root of an answer tree should have at least two child nodes or be a keyword node in itself. Assume that the root of an answer tree T has only one child and is not a keyword node. Then there exists a sub-tree T' in T which is reduced and contains the same set of keyword nodes as T. Since T' is usually given the higher relevance score than T and thus preferred by search methods, T is a redundant answer to the query.

To find the most relevant answers to a given query, we propose a measure to the relevance of answer trees taking both content nodes and root-to-keyword paths into account. First, given a node v having a query keyword k, the relevance of v to k can be computed based on the TF-IDF weighting scheme which is usually employed in information retrieval over text documents [3]. For instance, adopting the weighting scheme used in Apache Lucene text search engine [32], the relevance of v to k is defined by

$$rel(v, k) = \sqrt{tf(k, v)} \cdot \left(1 + \log\left(\frac{|V|}{|V(k)| + 1}\right)\right)^2 \quad (1)$$

where $|V|$ and $|V(k)|$ are the numbers of nodes in V and $V(k)$, respectively, and $tf(k, v)$ is the number of occurrences of k in v.

We also consider the length of the shortest path from a possible root node of an answer tree to a keyword node containing a query keyword to measure the structural relevance of the answer. Since the root node connects all the keyword nodes with each other in a given answer tree, it can be considered that the smaller the sum of the shortest distances from the root to each keyword node is, the more relevant to the query the answer tree is. We measure the relevance of a root-to-keyword path $n \rightarrow v_i$ by

$$rel(n, v_i) = 1 + \log\left(\frac{1}{dist(n, v_i) + 1}\right) \tag{2}$$

where $dist(n, v_i)$ denotes the length of the shortest path from n to v_i. Now, the relevance measure for answer trees can be defined based on the above two scoring factors as follows.

Definition 2 (A relevance scoring function for answer trees). Given an answer tree $T(n, \{v_1, v_2, \ldots, v_l\})$ to a query $q = \{k_1, k_2, \ldots, k_l\}$, the relevance of T to q is defined by

$$rel(T, q) = \sum_{1 \le i \le l} rel(n, v_i, k_i) = \sum_{1 \le i \le l} rel(n, v_i) \cdot \frac{rel(v_i, k_i)}{r_{\max}}$$

where r_{max} is the maximal value of $rel(v, k)$ for all keyword terms and nodes in G. ❑

In the above definition, the relevance of an answer tree to the given query is measured by the sum of relevance scores $rel(n, v_i, k_i)$ of each root-to-keyword path from root n to keyword node v_i for keyword k_i contained in the answer tree. Given a graph data, we can pre-compute the shortest paths from each node to keyword nodes containing keyword terms, as well as their relevance scores based on the above definition. Note that the pre-computed information can be used to build an effective index for processing queries, which will be described in detail in Section 4. Therefore, the proposed scoring function allows us to find the most relevant answer trees in an efficient way by exploiting path index.

In this paper, we take a set of answer trees into consideration which are not only reduced but also duplication-free in regard to their content nodes containing query keywords. It means that answer trees returned as a result of a query should have different sets of content nodes as well as a distinct root node to enhance diversity of the query result. Under these constraints, we search for the most relevant answers to the given query in an efficient way.

4. INDEXING SCHEME

To enable efficient search of top-k answers in a large graph database, we propose an indexing scheme for selected node-to-keyword paths in the graph data. Similar to the indexes used in BLINKS [12], it pre-computes the most useful paths using the relevance

measure proposed in the previous section and stores them in the keyword-node lists and node-keyword maps.

Keyword-node lists, denoted by *KNLists*, are a set of inverted lists defined for each keyword term in the graph. A list *KNList*(k) for keyword k stores the most relevant node-to-keyword path from each node in the graph to a node containing k. Formally, let $P(n, k) = \{n \rightarrow v_i \mid v_i \in V(k)\}$ for node n and keyword k, and $p_m(n, k)$ be the optimal path in $P(n, k)$ which has the highest relevance score, *i.e.*,

$$p_m(n, k) = \arg\max_{n \rightarrow v \in P(n, k)} rel(n, v, k).$$

KNList(k) stores index entries for the optimal paths $p_m(n, k)$ for all nodes n in the graph. The entry of $p_m(n, k)$ contains a quadruple (n, v_m, f_m, r_m) where v_m is the end node of $p_m(n, k)$ containing keyword k, f_m is the *first node* on the path besides node n (*i.e.*, the next node of n), and r_m is the relevance score of the path, *i.e.*, $rel(n, k, v_m)$. All the entries in *KNList*(k) are sorted in a decreasing order of relevance.

KNLists index allows us to find the paths from a node to a keyword which are most relevant to a given query in an efficient way without performing backward exploration in the graph data proposed by BANKS [2]. Backward exploration means expanding nodes in the directed graph backwards starting from the set of keyword nodes containing any query keyword to find relevant paths for query answers. Our query processing algorithm can avoid such graph exploration using the proposed index. Specifically, given a query containing keyword k, the keyword-node list for k, *i.e.* *KNList*(k) is read sequentially and the most relevant paths from a node to a keyword node for k can be easily found in the order of relevance scores. The start nodes of the retrieved paths can be a potential root of a candidate answer tree.

A primary node-keyword map, denoted by *NKMap*, is a hash map to store information on the most relevant paths for each pair of node and keyword in the graph. Formally, for a pair of node n and keyword k, it stores an ordered list of entries for a predefined number of n-to-k paths which have the highest relevance scores in $P(n, k)$, including the optimal path $p_m(n, k)$ defined above. An entry for an n-to-k path contains a triple (v_i, f_i, r_i), where the attributes denote the end node, first node, and relevance of the path, respectively, similar to the entry in the keyword-node list. The entries are stored and looked up in a hash map using the pair of n and k as a search key.

NKMap index is used to find the optimal paths from the root node of a potential answer tree to the query keywords efficiently. We can find the most relevant paths from a given root node to any query keyword directly from the *NKMap*, and thus we can avoid forward exploration of the graph data which expands a lot of nodes along the directed edges starting from a set of potential root nodes towards some keyword nodes to find relevant paths for query keywords [14]. Moreover, since it stores alternative paths in addition to the optimal path from a node to a keyword term, our search algorithm can find an alternative answer tree for a given root node efficiently in case duplication of content nodes occurs among the candidate answer trees, which will be described in detail in Section 5.3.

In addition to *NKMap*, we also introduce a secondary node-keyword map, denoted by $NKMap_s$, to store a node-to-keyword path as an alternative to the optimal path $p_m(n, k)$ for each pair of node n and keyword term k in the graph. It is the most relevant one

among the paths in $P(n, k)$ which have a *first node* different from that of $p_m(n, k)$. This index is used to find the optimal reduced answer tree rooted at a given node, which will be detailed in Section 5.2.

5. QUERY PROCESSING

In this section, we propose a query processing algorithm to find k best non-redundant answer trees for a keyword query over graph databases, based on the answer structure, relevance measure, and path indexes presented in the previous sections.

5.1 Outline of Top-k Query Processing

Our query processing model is based on the Threshold Algorithm [9] which is popularly used to evaluate top-k queries on multi-dimensional data such as multimedia objects. It performs sequential scan on the pre-computed index lists of data items, which are sorted in a descending order of per-attribute scores, and searches for top-k items with the highest total scores aggregated from the index lists.

Algorithm 1: Keyword Query Processing

Input: a keyword query $q = \{w_1, w_2, \ldots, w_l\}$ and $k \in Z^+$
Output: a set of top-k answer trees for q

1: a priority queue $Q_t \leftarrow \phi$, a set $C \leftarrow \phi$
2: $curRel[i] \leftarrow 0.0$ for all $i \in [1, l]$
3: Let $L(q) = \{L_i = KNList(w_i) \mid w_i \in q\ (1 \le i \le l)\}$.
4: **while** an entry exists in a list in $L(q)$ **do**
5: Select a list L_i in $L(q)$ in a round-robin manner.
6: Read an entry (n, v, f, r) at the current position in L_i.
7: $curRel[i] \leftarrow r$
8: **if** $n \notin C$ **then**
9: $V[i] \leftarrow (v, f, r)$
10: **for-each** $w_j \in q$ such that $j \neq i$ **do**
11: Look up the first entry (v_j, f_j, r_j) with key (n, w_j) in *NKMap*.
12: **if** the entry was found **then** $V[j] \leftarrow (v_j, f_j, r_j)$
13: **else goto** line 21
14: **if** $T(n, V)$ is a non-reduced sub-tree **then**
15: $V \leftarrow findReducedAnswer(n, V, q)$
16: **if** $V \neq \phi$ **and** $T(n, V)$ is a duplicate answer tree **then**
17: $V \leftarrow findUniqueAnswer(n, V, q)$
18: **if** $V \neq \phi$ **then** $Q_t \leftarrow Q_t \cup \{(n, V)\}$
19: $C \leftarrow C \cup \{n\}$
20: **if** $|Q_t| = k$ **and** $rel_k \ge \sum_{1 \le i \le l} curRel[i]$ **then break**
21: Derive top-k answer trees from the top-k entries in Q_t.

Algorithm 1 shows a sketch of our keyword search algorithm. It employees a priority queue Q_t to maintain k most relevant answer trees retrieved from the graph data. Given a keyword query $q = \{w_1, w_2, \ldots, w_l\}$, let $L(q)$ be a set of keyword-node lists $KNList(w_i)$ for all w_i in q. The algorithm performs sequential scan on the lists in parallel. That is, it

selects a list in $L(q)$ in a round-robin manner and reads an entry from the list (see lines 5-6). Whenever a new entry is retrieved from a list, its relevance value is recorded in an array *curRel* of the current relevance from each list in $L(q)$ (line 7). If an entry (n, v, f, r) read from a list for a query keyword is the first one regarding an optimal root-to-keyword path from node n to a query keyword in q, entries regarding the optimal paths from n to all the other keywords in q are looked up in the node-keyword map *NKMap* and are stored in an array V (lines 8-13). If all the optimal paths for query keywords are found from *NKMap*, an optimal answer tree rooted at n can be derived. Then, the algorithm examines whether the tree is in a reduced form and contains a unique set of content nodes compared with the other candidate trees in top-k queue Q_t. If it does not, an alternative answer tree rooted at n should be searched for using two algorithms which will be detailed later (lines 14-17). The answer tree is stored in Q_t if it is one of the k most relevant which have been found (line 18).

In the scan of the keyword-node lists, if at least k answer trees are found and their relevance scores are no less than those of the answer trees that are not found yet from the lists, the query processing algorithm can terminate safely with the correct top-k answer trees in Q_t for the given query. Since the entries in keyword-node lists are sorted in a decreasing order of relevance, the sum of the relevance values in the array *curRel* can serve as an upper bound of the relevance of the answer trees that have not been found yet. Thus, the algorithm stops list scan and returns the top-k answers in Q_t if the following condition is satisfied:

$$|Q_t| = k \text{ and } rel_k \geq \sum_{1 \leq i \leq l} curRel[i],$$

where rel_k is the relevance score of the kth relevant answer tree in Q_t (lines 20-21).

5.2 Finding Reduced Answers

Given a potential root node, Algorithm 1 searches for an optimal answer tree consisting of the most relevant root-to-keyword paths for query keywords. It is a non-reduced answer tree if and only if the first nodes of all the root-to-keyword paths in the tree are equal to the only child of the root. Thus, we can see if an answer tree is reduced or not by examining the first nodes of all root-to-keyword paths. However, it should be considered that if the root is selected as a keyword node for all the query keywords, the root node itself can be a reduced answer tree.

Assume that an optimal answer tree $T(n)$ rooted at node n is not a reduced tree. If there exist more than one reduced answer tree rooted at the same node n, the one with the highest relevance score should be chosen as an alternative to $T(n)$. For a keyword w in q, let $p_a(n, w)$ be the path from n to a keyword node v in $V(w)$ which has the first node different from that of $p_m(n, w)$ and has the highest relevance score. Formally, assuming that $P_a(n, w) = \{n \rightarrow v \mid v \in V(w), f(n \rightarrow v) \neq f(p_m(n, w))\}$ where f is a function from a path to the first node of the path,

$$p_a(n, w) = \underset{n \rightarrow v \in P_a(n,w)}{\arg\max} \; rel(n, v, w).$$

Also suppose that $T_i(n)$ be a sub-tree derived from $T(n)$ by replacing the optimal path $p_m(n, w_i)$ with $p_a(n, w_i)$ for a keyword w_i in q. That is,

$$T_i(n) = \{p_a(n, w_i)\} \cup \{p_m(n, w_j) \mid w_j \in q, j \neq i\}.$$

Note that $T_i(n)$ is a reduced answer to q since the first node of $p_a(n, w_i)$ is not equal to those of the other root-to-keyword paths $p_m(n, w_j)$ for keywords w_j in q ($j \neq i$). Now, among l alternative sub-trees $T_i(n)$ of $T(n)$ ($1 \leq i \leq l$), the one with the highest relevance is the best reduced answer tree rooted at n.

For example, a graph shown in Fig. 2 contains keywords w_1 and w_2 in the set of nodes $V(w_1)$ and $V(w_2)$, respectively. Assuming that all nodes have the same relevance to the keywords they contain, the most relevant paths $p_m(n, w_1)$ from n to w_1 is $n \rightarrow v_1$ and $p_m(n, w_2)$ from n to w_2 is $n \rightarrow v_2$. Thus, given a query $q = \{w_1, w_2\}$ over the graph, the optimal answer tree $T(n)$ rooted at n consists of these two paths. However, $T(n)$ is a non-reduced sub-tree since two paths share the same first node f_3. Note that an alternative path $p_a(n, w_1)$ from n to a node in $V(w_1)$ is $n \rightarrow v_4$ and a path $p_a(n, w_2)$ from n to a node in $V(w_2)$ is $n \rightarrow v_5$. Using one of these paths, we can get two relevant reduced answer trees rooted at n, *i.e.*, $T_1(n) = \{p_a(n, w_1)\} \cup \{p_m(n, w_2)\} = \{n \rightarrow v_4\} \cup \{n \rightarrow v_2\}$ and $T_2(n) = \{p_m(n, w_1)\} \cup \{p_a(n, w_2)\} = \{n \rightarrow v_1\} \cup \{n \rightarrow v_5\}$. Since $T_1(n)$ has the higher relevance score than $T_2(n)$, it is the most relevant one among 36 *reduced* answer sub-trees rooted at n in the graph, and thus it should be selected as a candidate answer to the query. Note that if we choose path $n \rightarrow v_3$ as an alternative path to $p_m(n, w_1)$ or $p_m(n, w_2)$, it will lead to another non-reduced sub-tree the root of which has only one child node f_3.

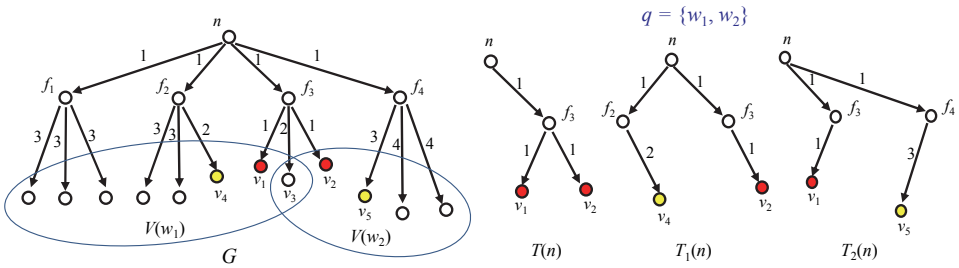


Fig. 2. An example of finding the optimal reduced answer tree.

Algorithm 2: Find Reduced Answer

Input: a *nodeID* n, an array V of (*nodeID*, *nodeID*, *rel*)'s, and a query q
Output: an array $V[1..l]$ of (*nodeID*, *nodeID*, *rel*)'s

1: $A[i] \leftarrow$ *null* for all $i \in [1, l]$
2: **for-each** $w_i \in q$ **do**
3: Look up an entry (v_i, f_i, r_i) with key (n, w_i) in $NKMap_s$.
4: **if** the entry was found **then** $A[i] \leftarrow (v_i, f_i, r_i)$
5: **if** $A[i] =$ *null* for all $i \in [1, l]$ **then return** ϕ
6: **else**
7: Find $i \in [1, l]$ such that $(V[i].rel - A[i].rel)$ is minimal.
8: $V[i] \leftarrow A[i]$
9: **return** V

To find optimal reduced answer trees efficiently without exploring the entire graph data, Algorithm 2 exploits the $NKMap_s$ index proposed in Section 4.1, which stores the optimal alternative paths $p_a(n, w)$ defined above for all pairs of node n and keyword w in the graph. Given a non-reduced optimal answer tree $T(n)$, it first looks up in $NKMap_s$ the entries for alternative paths from n to all the query keywords (lines 2-4). An optimal reduced answer tree can be easily obtained by selecting such a query keyword w_i that difference between $p_m(n, w_i)$ and $p_a(n, w_i)$ is the smallest and then by replacing $p_m(n, w_i)$ with $p_a(n, w_i)$ in $T(n)$ (lines 7-8).

5.3 Finding Duplication-Free Answers

Now, we focus on finding top-k answer trees which are duplication-free in regard to content nodes. An answer tree T_1 which belongs to a set A of top-k answer trees to a given query is a duplicate answer if and only if there exists an answer tree T_2 in A ($T_2 \neq T_1$) which contains the same set of content nodes as T_1 and has no smaller relevance score than T_1. When a duplicate answer tree $T(n, C)$ rooted at a specific node n is produced in query processing, we aim to find an alternative answer tree $T(n, C')$ which is rooted at the same node n and has a set C' of content nodes different from those of all the other candidate answer trees in top-k queue Q_t. Assuming that the graph has at most p paths from node n to each keyword in the query of size l, p^l answer trees rooted at n can be derived from the combinations of root-to-keyword paths. To find an optimal one which is not a duplicate tree and has the highest relevance score efficiently without taking all the possible answer trees into consideration, we suggest a state space search algorithm based on a branch-and-bound strategy.

As shown in Fig. 3, search space consists of an ordered tree of states. Each state represents a unique combination of l paths from the same node to all query keywords and thus defines an answer tree. In the tree of states, each state has p child states, which choose one of p paths from the root to keyword nodes containing the same keyword in a decreasing order of relevance from left to right. Without loss of generality, we assume that the states at level i ($1 \leq i \leq l$) choose a different root-to-keyword path for a keyword w_i in a given query $q = \{w_1, w_2, \ldots, w_l\}$. We also assume that they inherit from their parent state the root-to-keyword paths to the keywords in $\{w_1, w_2, \ldots, w_{i-1}\}$ while they have the most relevant root-to-keyword path for the keywords in $\{w_{i+1}, w_{i+2}, \ldots, w_l\}$. For example, suppose that a query $q = \{w_1, w_2, w_3\}$ is given and the graph has three different paths from root n to each query keyword. Fig. 3 shows a part of the state tree where each state contains an array of index numbers indicating root-to-keyword paths selected for the query keywords. We can see that the sibling states shown at level 2 contain one of three paths to keyword w_2 identified by a different index number, and all the states inherit a root-to-keyword path to keyword w_1 identified by the index number 1 from their parent state s_0. For keyword w_3, they all have the optimal path to w_3 denoted by the index number 1 stored in the third entry of the arrays. Note that in the tree of states, an answer tree represented by state s has no smaller relevance score than those derived from the descendent states of s. It should be also noted that since sibling states contain a different path to the same query keyword in a descending order of relevance, the answer tree derived from state s has no smaller relevance than those derived from the right siblings of s.

Considering these characteristics, we can explore the search space in an efficient way by pruning a large number of irrelevant states.

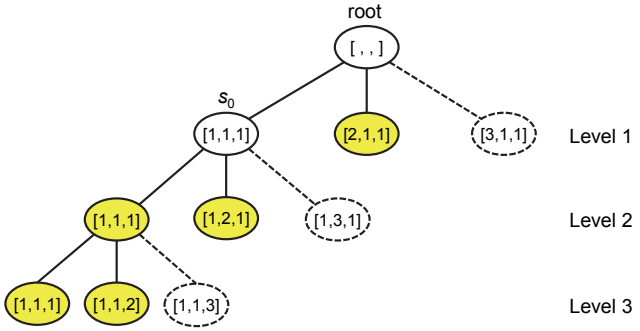


Fig. 3. A tree of states generated in state space search to find an optimal duplication-free answer.

Algorithm 3: Find Unique Answer.

Input: a *nodeID* n, an array V of (*nodeID*, *nodeID*, *rel*)'s, and a query q
Output: an array $V'[1..l]$ of (*nodeID*, *nodeID*, *rel*)'s

```
1:  UB ← rel(T(n, V), q), LB ← rel_k, bestSolution ← φ
2:  a priority queue Q_s ←{s_0}, where s_0 is an initial state.
3:  while there exists a state in Q_s do
4:      e ← a state in Q_s whose score is maximal
5:      Q_s ← Q_s / {e}
6:      if score(e) ≤ LB then break
7:      repeat
8:          if e has the next sibling state then
9:              Generate the next sibling state s of e using NKMap.
10:             if score(s) > LB then
11:                 if score(s) ≤ UB then
12:                     if s is a solution state then
13:                         bestSolution ← s; LB ← score(s)
14:                     else Q_s ← Q_s ∪ {s}
15:                 else Q_s ← Q_s ∪ {s}
16:         if e is a non-leaf state then
17:             Generate the first child state c of e.
18:             e ← c
19:         else   e ← φ
20:     until e = φ
21: if bestSolution ≠ φ then
22:     for-each w_i ∈ q do
23:         V'[i] ← (v_i, f_i, r_i) of the path from n to w_i selected by bestSolution
24:     return V'
25: else return φ
```

Algorithm 3 shows a pseudo-code of our best-first state space search algorithm. It uses a priority queue Q_s to store non-solution states which should be further expanded. As shown in line 2, it is initialized by state s_0 which represents a duplicate answer sub-

tree given to the algorithm. We assume that it consists of the best root-to-keyword paths for all the query keywords (see Fig. 3). The state search starts from the initial state s_0 in Q_s. At each round of the algorithm, a state e with the highest score is selected from the priority queue (line 4), where the score of a state is defined by the sum of relevance of the paths chosen by the state, *i.e.* the relevance of the answer tree defined by the state. If the score of e is greater than that of the best state found, denoted by LB, the next sibling state s of e is generated and investigated (lines 8-15). *NKMap* index is used to retrieve information on a relevant path for a new state efficiently. If the score of s is no greater than LB, all of its descendent states can be safely excluded from further exploration (refer to line 10). Otherwise, if the score of s is no greater than that of the duplicate answer tree denoted by state s_0 and s is a solution state, *i.e.*, a reduced and unique answer tree with respect to the candidate answers in top-k queue, it is a new best solution (lines 11-13). Otherwise, it is stored in the queue Q_s. Subsequently, if e is not a leaf state, the first child of e is generated and the above process is repeated on it (lines 16-18). In Fig. 3, for example, the shaded nodes indicate the states generated and investigated in the first round of the outermost loop of the algorithm. Note that during the best-first search, if a state chosen in Q_s derives no better answer trees than the current best solution state, the algorithm terminates and returns the information on the root-to-keyword paths selected by the best solution state. It represents the most relevant and unique answer tree rooted at a given node n (line 6 and lines 21-24).

It should be noted that if the new answer tree found replaces one of the previous candidate answer trees in the top-k queue which is duplicate with regard to the new answer (line 18 of Algorithm 1), an alternative to the duplicate answer tree should be subsequently searched in the graph. We omit the details in the algorithm above for the sake of simplicity of description.

5.4 Performance Analysis

The query processing algorithm proposed above exploits the path indexes consisting of inverted lists for keywords and hash maps for node-keyword pairs. As shown in Algorithm 1, it performs parallel scan of the inverted lists for query keywords in parallel based on the Threshold Algorithm [9]. BLINKS method also adopts a similar strategy, which is proven to be optimal within a factor of the size of the query [12].

Given a graph with n nodes, the size of the proposed *Keyword-Node List* for a keyword term k is $O(n)$ since for each node in the graph it stores the most relevant one among the paths from the node to the keyword, if any. Thus, given an l-keywords query, lines 5-20 in the outermost loop in Algorithm 1 execute in $O(l \cdot n)$ in worst case. In practice, however, the number of entries in $KNList(k)$ is much smaller than n if a keyword k appears a subset of nodes and they are also reachable from a subset of nodes in the graph.

Note that lines 9-19 in Algorithm 1 executes at most n times since our approach is based on the distinct root semantics. *NKMap* index is looked up $l-1$ times in line 11, and $NKMap_s$ is used l times in line 3 of Algorithm 2 which is invoked in line 15 of Algorithm 1. On the other hand, Algorithm 3 which is invoked in line 17 generates at most p^l states in state space search where p is the number of relevant paths stored in *NKMap* index for a pair of node and keyword. Thus, it costs $O(p^l)$ in worst case while the proposed best-first search algorithm can prune a large portion of irrelevant states in the search space, as

shown in the experiments in Section 6. As a result, the total time complexity of the proposed top-k query processing algorithm is $O((l + p^l) \cdot n)$.

6. PERFORMANCE EVALUATION

We evaluate effectiveness and efficiency of the proposed approach by conducting experiments using real graph datasets. Two variants of our approach are tested: a method called *Reduced* which searches only for reduced answer trees without considering content nodes duplication and the other one called *Red&Dup-free* which produces answers that are both reduced and duplication-free. We selected BLINKS as a baseline method for performance comparison since it is most popular and practical top-k keyword search algorithm for a large amount of generic graph data. Similar to our approach, it finds sub-trees under the distinct root semantics and exploits pre-computed indexes on the graph to find optimal top-k answer trees efficiently in large graph data. We also experiment with a modified version of BLINKS, called *BLINKS-N*, which detects non-reduced or duplicate sub-trees and excludes them from the candidate answers to the query during search process. All the considered algorithms are implemented in Java 6.

As for the test dataset, we use two real graph data, a geographic database Mondial[1] and a movie database IMDB[2]. From Mondial, we select a subset of entities and relationships and build a graph including 6,431 nodes, 19,951 edges, and 15,815 keyword terms. In IMDB, we use data on about 147K movies as well as data on actors, actresses, and directors related to the movies to construct a large graph consisting of about 831K nodes, 2.82M edges, and 303K keyword terms. In implementation of indexing and search algorithms, we used JGraphT[3] library to construct in-memory data structures and to compute the shortest paths between pairs of nodes in the graphs based on Bellman-Ford algorithm. We also exploited Apache Lucene[4] library to extract keywords from nodes in the graph and compute the relevance of the nodes to keyword terms based on Eq. (1). For the sake of simplicity and efficiency of experiments, we assume that all edges have the same weight of 1 and consider only the node-to-keyword paths the length of which are no longer than 5. Table 1 shows a set of test keyword queries over two graph datasets, which have been processed by considered methods to find top-k answers. The experiments have been conducted on a LINUX server having 10 1.7GHz hexa-core CPUs and 32GB RAM.

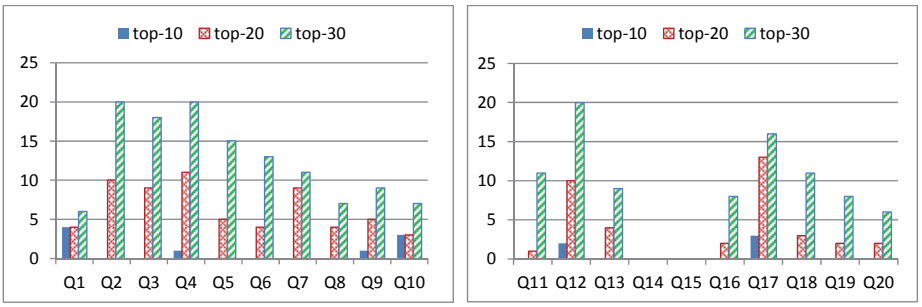
Table 1. Test queries.

Dataset	Query	Keyword list	Dataset	Query	Keyword list
Mondial	Q_1	Alaska, arctic, sea	IMDB	Q_{11}	elf, dwarf, fantasy
	Q_2	cape, gulf, Africa		Q_{12}	earthquake, flood, disaster
	Q_3	Vienna, Donau, Alps		Q_{13}	alien, robot, attack
	Q_4	caldera, lake, America		Q_{14}	explosion, collapse, rescue
	Q_5	lake, Quebec, Canada		Q_{15}	disaster, rescue, hero
	Q_6	Himalaya, India, Pakistan		Q_{16}	emperor, war, battle
	Q_7	river, Minnesota, Louisiana		Q_{17}	space, earth, return
	Q_8	lake, Michigan, Ontario		Q_{18}	travel, moon, mars
	Q_9	city, desert, California		Q_{19}	earth, sea, ocean
	Q_{10}	island, Vancouver, Seattle		Q_{20}	time, travel, future

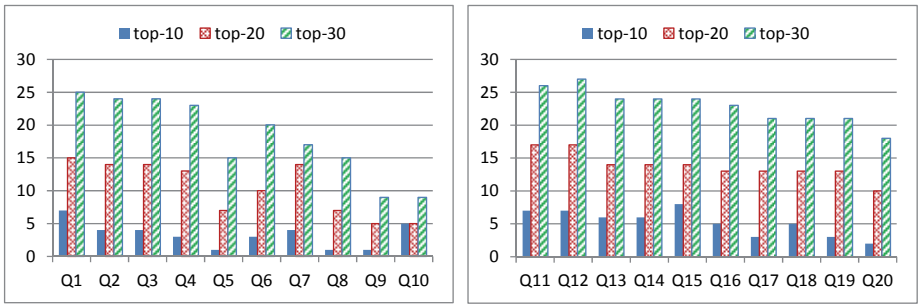
[1] http://www.dbis.informatik.uni-goettingen.de/Mondial/
[2] http://www.imdb.com/
[3] http://www.jgrapht.org/
[4] http://lucene.apache.org/

6.1 Goodness of Answers

Fig. 4 shows the number of non-reduced and duplicate sub-trees included in the answers to the test queries produced by BLINKS method. It indicates that top-10, top-20, and top-30 answers to the queries over Mondial data have 1, 6, and 13 non-reduced answers and 3, 10, and 18 duplicate answers respectively on average. For the queries over IMDB data, BLINKS returns 1, 4, and 9 non-reduced trees and 5, 14, and 23 duplicate trees respectively in top-10, top-20, and top-30 answers on average. We can observe that as the number of query answers increases, the proportions of non-reduced trees and duplicate trees in the results also grow. The average numbers of non-reduced and duplicate trees in top-30 answers to a test query generated by BLINKS amount to about 37% and 70%, respectively, of the entire answers. In contrast, *BLINKS-N* and *Reduced* have produced no answer tree in a non-reduced form, and *Red&Dup-free* has returned neither non-reduced nor duplicate answer trees for all the test queries.



(a) Number of answers in a non-reduced form.



(b) Number of answers having a duplicate set of content nodes.

Fig. 4. Redundant answers in top-k results produced by BLINKS.

To evaluate and compare the effectiveness of different search methods, we use a measure for goodness of top-k answers produced by each method, which is based on two factors, *i.e.* diversity and relevance of the answers. Given a keyword query q, let $A = \{T_1, T_2, \ldots, T_k\}$ be an ordered set of top-k answer trees which satisfies $rel(T_i, q) \geq rel(T_j, q)$ for all $i < j$, and let $o(T_i, A)$ denote the order of T_i in A. The diversity of A is defined as the proportion of the answer trees in A which are reduced and also unique in terms of content nodes containing query keywords. Formally,

$$div(A) = \frac{|N(A)|}{|A|},$$

where $N(A) = \{T_i \mid T_i$ is a reduced tree in A, and there is no T_j in A such that $o(T_j, A) < o(T_i, A)$ and T_j has the same set of content nodes as $T_i\}$.

The relevance of A is measured by the average of the relevance scores of the answer trees in A normalized by the highest relevance score of the best answer tree. That is,

$$rel(A) = \frac{1}{k \cdot rel(T_1, q)} \sum_{1 \le i \le k} rel(T_i, q),$$

where $rel(T_i, q)$ is the relevance score of the ith answer tree T_i in A for query q. Finally, the goodness of A is computed as the multiplicative combination of diversity and relevance of A as follows.

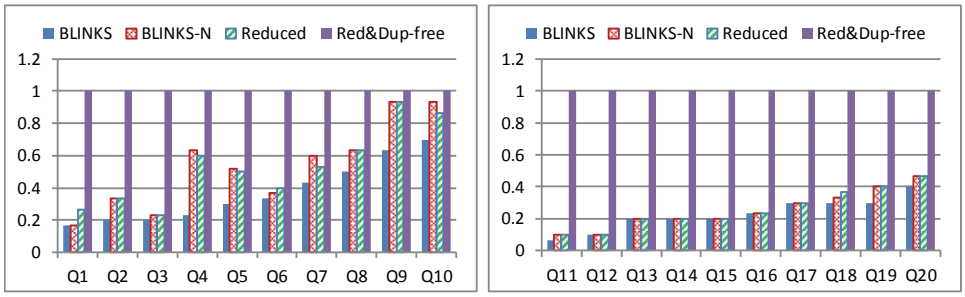
$$goodness(A) = div(A)^{\lambda} \cdot rel(A),$$

where λ is a weighting parameter, which is set to 1 in our experiments.

Figs. 5-7 respectively present the diversity, relevance, and goodness of top-30 answers to each test query obtained by each method. As shown in Fig. 5, it is obvious that *Red&Dup-free* produces the most diverse results without either non-reduced or duplicate answer trees. Meanwhile, diversity of the answers by *BLINKS* is the lowest among all methods, which is only about 37% of that of the answers by *Red&Dup-free* on average over Mondial data. The query answers generated by *Reduced* and *BLINKS-N* include many duplicate trees hence their average diversity scores are also small, which are about 54% of that of *Red&Dup-free*. For the queries on IMDB, differences in diversity between *Red&Dup-free* and the other methods are larger than those for the queries over Mondial.

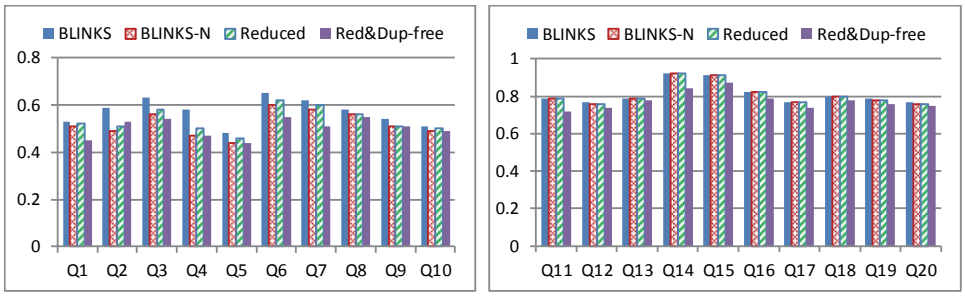
On the other hand, relevance of query answers is shown to be in contrast with their diversity. Fig. 6 indicates that all relevance scores obtained by the methods other than *BLINKS* are no higher than those by *BLINKS* and the result by *Red&Dup-free* has the lowest in the most of queries. Specifically, the relevance of the query answers on Mondial generated by *BLINKS-N*, *Reduced*, and *Red&Dup-free* is respectively about 8.8%, 6.1%, and 11.8% lower than that of *BLINKS* on average. This is due to the fact that the methods besides *BLINKS* make an attempt to avoid non-reduced and/or duplicate answer trees even though they are the best answers in terms of relevance to the query. Note that our *Reduced* method has achieved more relevant answers than *BLINKS-N* in the most queries by finding the most relevant reduced answer tree as an alternative to a non-reduced one. For the queries on IMDB data, the differences in relevance of the answers are not significant and the average relevance score of our *Red&Dup-free* method is only about 4.4% lower than that of *BLINKS*.

Fig. 7 shows goodness of top-30 answers to each test query generated by each method. For the most queries, goodness of the answers by *Red&Dup-free* is much higher than that of the results by *BLINKS* due to a large gap in diversity of the answers between two methods.



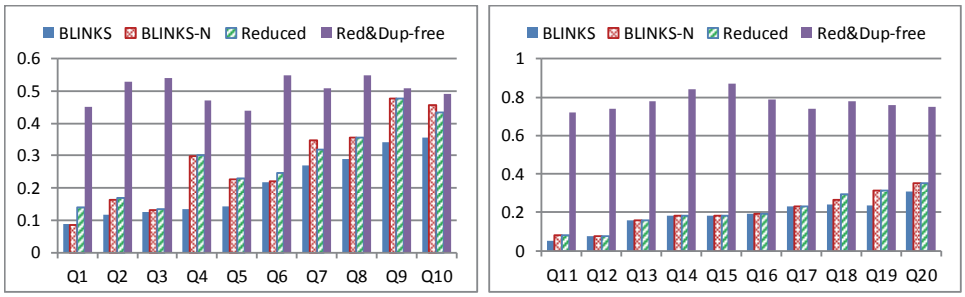
(a) Queries on Mondial. (b) Queries on IMDB.

Fig. 5. Diversity of top-30 answers.



(a) Queries on Mondial. (b) Queries on IMDB.

Fig. 6. Relevance of top-30 answers.



(a) Queries on Mondial. (b) Queries on IMDB.

Fig. 7. Goodness of top-30 answers.

Fig. 8 indicates that average goodness of top-30 answers by *Red&Dup-free* on the Mondial and IMDB data has improved by more than 140% and 310%, respectively, over the answers by *BLINKS*. Meanwhile, the goodness score of *Reduced* has increased by about 34.4% on Mondial data and by about 10% over IMDB data on average compared to the result by *BLINKS*. Figs. 9 (a) and (b) show that goodness of the answers by all methods degrades as the number of answers to be found increases. However, we can observe in Figs. 9 (c) and (d) that goodness of the answers by three methods besides *BLINKS* improves more over the results by *BLINKS* as the value of k grows. Especially, *Red&Dup-free* method has achieved significant enhancement on the goodness of answers, which is nearly proportional to the number of query answers to be returned.

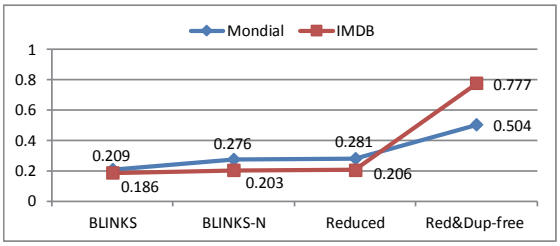
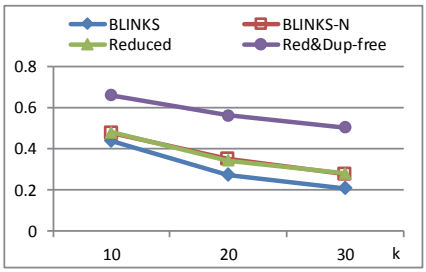
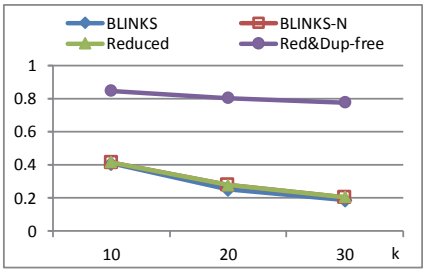


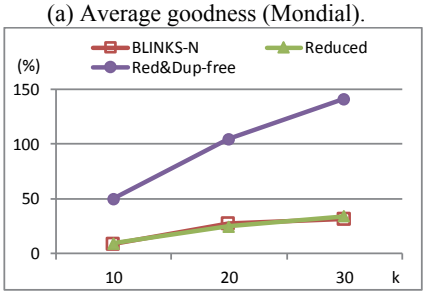
Fig. 8. Average goodness of top-30 answers obtained by each method.



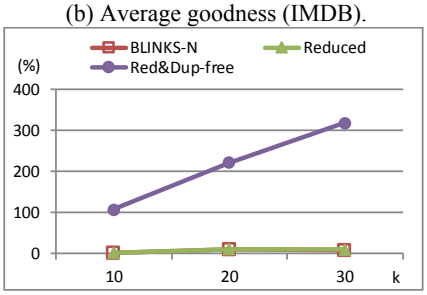
(a) Average goodness (Mondial).



(b) Average goodness (IMDB).



(c) Improvement over BLINKS (Mondial).



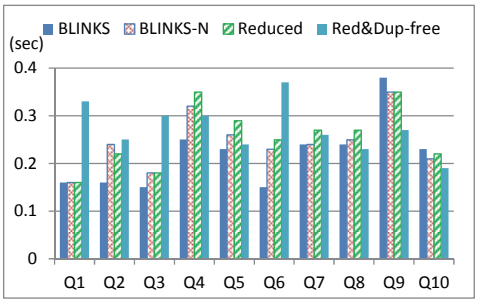
(d) Improvement over BLINKS (IMDB).

Fig. 9. Average goodness of top-*k* answers with varying *k*.

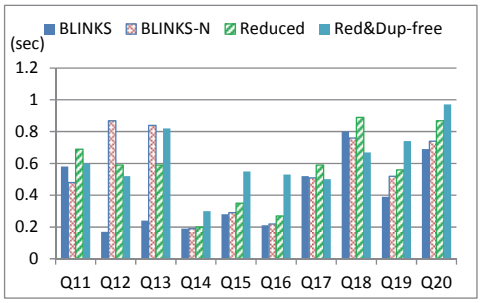
6.2 Execution Efficiency

To evaluate execution efficiency of our approach, we measure and compare the time spent by each method in processing test queries. Fig. 10 presents the execution time of each method in finding top-30 answers to the queries. While the results vary depending on queries, average execution time tends to increase in the order of *BLINKS*, *BLINKS-N*, *Reduced*, and *Red&Dup-free*, as shown in Fig. 10(c). For the queries over Mondial data, *Reduced* and *Red&Dup-free* take time respectively about 17% and 25% more than *BLINK* on average. On IMDB data, their average execution times increase by about 37% and 51% respectively than that of *BLINKS*. This is due to the overhead required in the search of optimal reduced answers and duplication-free answers to the given queries.

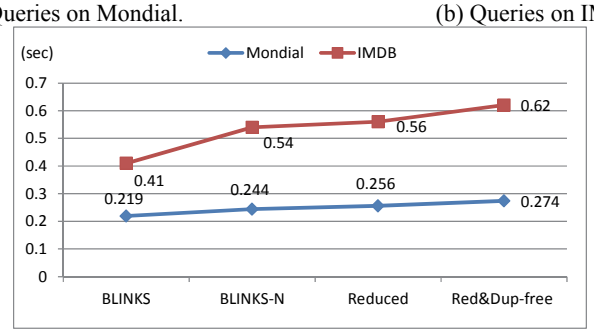
Fig. 11 shows that average execution time of each method spent in finding top-*k* answers to the queries over IMDB data increases linearly with the number *k* of answers to be found. The proposed methods have a little larger growth rate than *BLINKS* while the execution time of *BLINKS-N* increases most rapidly.



(a) Queries on Mondial.



(b) Queries on IMDB.



(c) Average execution time.

Fig. 10. Execution time to find top-30 answers.

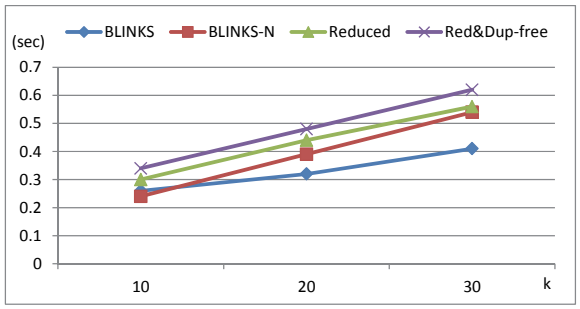
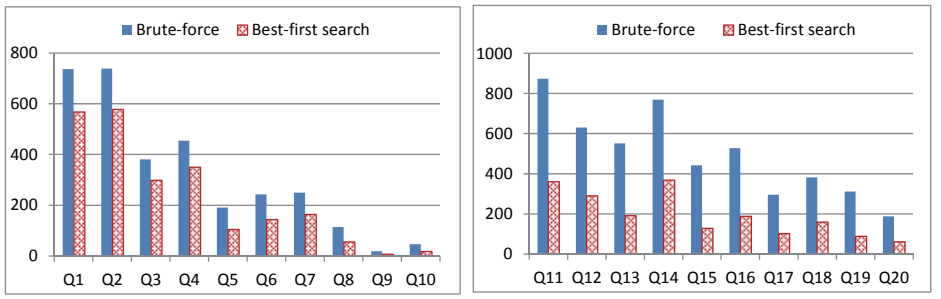


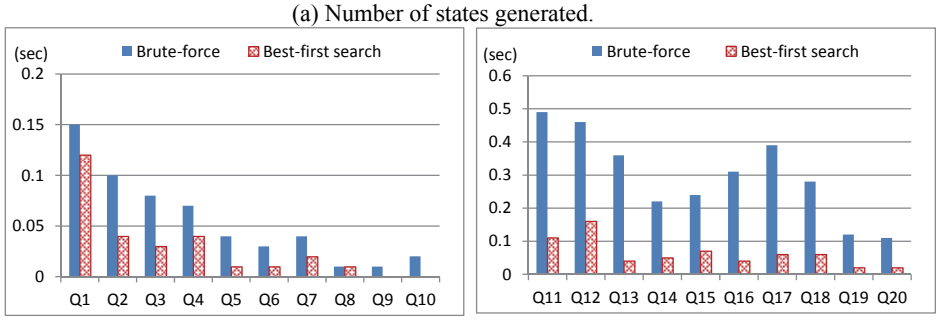
Fig. 11. Average execution time to find top-*k* answers with varying *k*.

Finally, we evaluate performance of the best-first state space search strategy to find a set of optimal duplication-free answers proposed in Section 5.3. It is compared with a naïve strategy conducting brute-force exploration of the entire candidate answer trees which can be derived from the path indexes for a given root node. Fig. 12 compares the number of states, *i.e.*, answer trees examined by each strategy to find top-30 duplication-free answers to the test queries, as well as their execution times. We can observe that for the queries over Mondial and IMDB data, the average number of states generated by our best-first search strategy reduces by about 30% and 61.2% respectively compared to the brute-force search strategy. Execution time of the proposed approach decreases by about 50% and 80% respectively on Mondial and IMDB on average. Fig. 13 indicates that in both strategies, execution time as well as the numbers of states generated increas-

es in proportion to the number of answers to be returned, while our best-first search strategy can achieve a large amount of performance improvement over the naïve approach with increasing values of *k*.

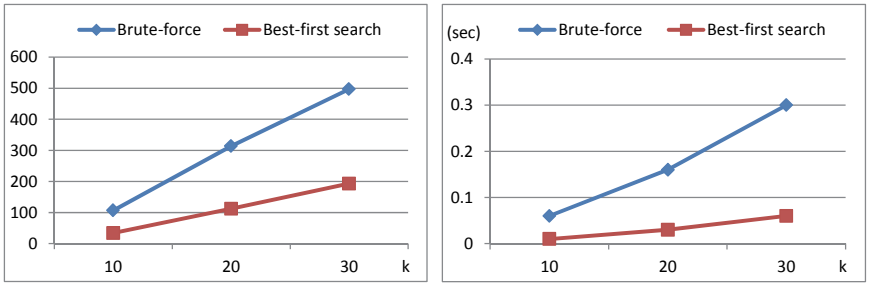


(a) Number of states generated.



(b) Execution time.

Fig. 12. Performance of the state search algorithm to find top-30 duplication-free answers.



(a) Average number of states generated. (b) Average execution time.

Fig. 13. Performance of the state space search algorithm with varying *k*.

7. CONCLUSIONS

In this paper, we proposed a new approach to processing keyword queries over graph databases to find a set of answers that are not only minimal and relevant to the query but also non-redundant and diverse in their structures and content nodes. We suggested an efficient indexing scheme to compute and store the most relevant paths from nodes to keywords in the graph. We also proposed a query processing algorithm to find

top-*k* answers efficiently by exploiting the pre-constructed indexes. Effectiveness and efficiency of the proposed method have been demonstrated by extensive experimental study using real graph datasets in comparison with the previous methods. By producing non-redundant and relevant answer trees for a given keyword query, our approach can provide the users with diverse and meaningful results to satisfy their various information need on large graph databases.

REFERENCES

1. A. Angel and N. Koudas, "Efficient diversity-aware search," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 2011, pp. 781-792.
2. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *Proceedings of IEEE International Conference on Data Engineering*, 2002, pp. 431-440.
3. S. Buttcher, C. Clarke, and G. Cormack, *Information Retrieval*: *Implementing and Evaluating Search Engine*, MIT Press, MA, 2010.
4. X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 2011, pp. 373-384.
5. G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri, "Efficient diversification of web search results," in *Proceedings of the VLDB Endowment*, Vol. 4, 2011, pp. 451-459.
6. B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword search on external memory data graphs," in *Proceedings of the VLDB Endowment*, Vol. 1, 2008, pp. 1189-1204.
7. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-*k* min-cost connected trees in databases," in *Proceedings of IEEE International Conference on Data Engineering*, 2007, pp. 836-845.
8. M. Drosou and E. Pitoura, "Search result diversification," *ACM SIGMOD Record*, Vol. 39, 2010, pp. 41-47.
9. R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, Vol. 66, 2003, pp. 614-656.
10. I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proceedings of IEEE International Conference on Data Engineering*, 2008, pp. 656-665.
11. K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword proximity search in complex data graphs," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 2008, pp. 927-940.
12. H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: ranked keyword searches on graphs," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 2007, pp. 305-316.
13. F. K. Hwang and D. S. Richards, "The Steiner tree problem," *Networks*, Vol. 22, 1992, pp. 55-89.
14. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proceedings of the 31st VLDB Conference*, 2005, pp. 505-516.

15. M. Kargar and A. An, "Keyword search in graphs: finding r-cliques," in *Proceedings of the VLDB Endowment*, Vol. 4, 2011, pp. 681-692.
16. M. Kargar, A. An, and X. Yu, "Efficient duplication free and minimal keyword search in graphs," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26, 2014, pp. 1657-1669.
17. W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable keyword search on large RDF data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26, 2014, pp. 2774-2788.
18. G. Li, J. Feng, and J. Xu, "Desks: direction-aware spatial keyword search," in *Proceedings of IEEE International Conference on Data Engineering*, 2012, pp. 474-485.
19. G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 2008, pp. 903-914.
20. C.-S. Park and S. Lim, "Efficient processing of keyword queries over graph databases for finding effective answers," *Information Processing and Management*, Vol. 51, 2015, pp. 42-57.
21. L. Qin, J. X. Yu, L. Chang, and Y. Tao, "Querying communities in relational databases," in *Proceedings of IEEE International Conference on Data Engineering*, 2009, pp. 724-735.
22. F. Radlinski and S. Dumais, "Improving personalized web search using result diversification," in *Proceedings of the 29th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, pp. 691-692.
23. D. Rafiei, K. Bharat, and A. Shukla, "Diversifying web search results," in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 781-790.
24. T. Tran, S. Rudolph, P. Cimiano, and H. Wang, "Top-*k* exploration of query candidates for efficient keyword search on graph-shaped (RDF) data," in *Proceedings of IEEE International Conference on Data Engineering*, 2009, pp. 405-416.
25. D. Vallet and P. Castells, "Personalized diversification of search results," in *Proceedings of the 35th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2012, pp. 841-850.
26. S. Vargas and P. Castells, "Rank and relevance in novelty and diversity metrics for recommender systems," in *Proceedings of the 5th ACM Conference on Recommender Systems*, 2011, pp. 109-116.
27. J. X. Yu, L. Qin, and L. Chang, "Keyword search in relational databases: a survey," *Bulletin of IEEE CS on Data Engineering*, Vol. 33, 2010, pp. 67-78.
28. D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proceedings of IEEE International Conference on Data Engineering*, 2009, pp. 688-699.
29. D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-*k* spatial keyword search," in *Proceedings of International Conference on Extending Database Technology*, 2013, pp. 359-379.
30. C. Zhang, Y. Zhang, W. Zhang, and X. Li, "Inverted linear quadtree: Efficient top-*k* spatial keyword search," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 28, 2016, pp. 1706-1721.
31. C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th International*

Conference on World Wide Web, 2005, pp. 22-32.
32. Apache Software Foundation, "Apache lucene 5.5.0 documentation," https://lucene.apache.org/core/5_5_0/index.html.

Chang-Sup Park received his Ph.D. degree in Computer Science from Korea Advanced Institute of Science and Technology in 2002. He is currently an Associate Professor at Dongduk Women's University in Seoul, Korea. His research interests include graph database, semantic web, information retrieval, big data analysis, and query optimization.