



## Retrieving keyworded subgraphs with graph ranking score

Seung Kim<sup>a</sup>, Wookey Lee<sup>b,\*</sup>, Nidhi R. Arora<sup>b</sup>, Tae-Chang Jo<sup>c</sup>, Suk-Ho Kang<sup>a</sup>

<sup>a</sup> Dept. of Industrial Engineering, Seoul National University, 599 Gwanak-ro, Gwanak-gu, Seoul 151-744, Republic of Korea

<sup>b</sup> Dept. of Industrial Engineering, INHA University, 253 Yonghyun-dong, Nam-gu, Incheon 402-751, Republic of Korea

<sup>c</sup> Dept. of Mathematics, INHA University, 253 Yonghyun-dong, Nam-gu, Incheon 402-751, Republic of Korea

### ARTICLE INFO

#### Keywords:

Graph-structured data

Steiner tree

Graph ranking score (GRS)

### ABSTRACT

Keyword queries have long been popular to search engines and to the information retrieval community and have recently gained momentum for its usage in the expert systems community. The conventional semantics for processing a user query is to find a set of top-k web pages such that each page contains all user keywords. Recently, this semantics has been extended to find a set of cohesively interconnected pages, each of which contains one of the query keywords scattered across these pages. The keyword query having the extended semantics (i.e., more than a list of keywords hyperlinked with each other) is referred to the graph query. In case of the graph query, all the query keywords may not be present on a single Web page. Thus, a set of Web pages with the corresponding hyperlinks need to be presented as the search result. The existing search systems reveal serious performance problem due to their failure to integrate information from multiple connected resources so that an efficient algorithm for keyword query over graph-structured data is proposed. It integrates information from multiple connected nodes of the graph and generates result trees with the occurrence of all the query keywords. We also investigate a ranking measure called graph ranking score (GRS) to evaluate the relevant graph results so that the score can generate a scalar value for keywords as well as for the topology.

© 2011 Elsevier Ltd. All rights reserved.

### 1. Introduction

Graph data structure has emerged as an exceedingly popular tool to represent complex relationships among data elements in a wide variety of applications. For instance, it is used to represent data items as nodes and their respective keyrefs or IDREFs in case of semi-structured XML data; to represent RDF triplets as nodes and relationship between the triplets as edges for Semantic Web, Web pages as nodes with hypertext links or DOM element container as edges for the World Wide Web and also in Social Networks as well as many latest projects in the field of bio-informatics. Hence, graphs (directed/undirected) have become conventional data model for structured, semi-structured or unstructured data.

A keyword based information retrieval method from graph structures becomes a popular method to search for information not only in the information retrieval community but also recently in database community because of its simplicity and user friendly query interface. Recently systems using 'schema-agnostic' key-

word search to query Database systems have received a lot of attention (Ding, Jeffrey, Wang, Lu, & Zhang, 2007; He, Wang, Yang, & Yu, 2007; Kimelfeld & Sagiv, 2006; Shin, Huh, & Lee, 2007). The common intuitive query response is in the form of a sub-graph that satisfies all the user requirements, typically at least once occurrence of each query keyword. In the World Wide Web, for instance, the conventional semantics for processing a user query is to find a set of top-k web pages such that each page contains all user keywords but this semantics has been extended to find a set of cohesively interconnected pages, where all the query keywords scattered across these pages. We refer to a keyword query having the extended semantics (i.e., list of keywords hyperlinked with each other) as a graph query. A information retrieval model can be defined as a Quadruple  $[D, Q, F, R(q_i, d_j)]$  where  $D$  represents the document collection,  $Q$  represents user query,  $F$  defines the framework for modeling document-query relationship and  $R(q_i, d_j)$  defines the ranking function (Hammami, Chahir, & Chen, 2006). We therefore adopt graphs to represent the problem domain or in other words as the data model  $D$ , keyword query as the query model  $Q$  and proceed to define our own  $F$  and  $R$ .

The lineage of information in Web and Databases have also modified the answer model from a single web page or relational tuple to physically connected set of webpages, which can be a graph query result. For instance, if user requires to search for a

\* Corresponding author. Tel.: +82 32 860 7371; fax: +82 32 867 1605.

E-mail addresses: [seung275@snu.ac.kr](mailto:seung275@snu.ac.kr) (S. Kim), [trintity@inha.ac.kr](mailto:trintity@inha.ac.kr) (W. Lee), [rustagi.nidhi@gmail.com](mailto:rustagi.nidhi@gmail.com) (N.R. Arora), [taechang@inha.ac.kr](mailto:taechang@inha.ac.kr) (T.-C. Jo), [shkang@snu.ac.kr](mailto:shkang@snu.ac.kr) (S.-H. Kang).

house with reasonable size and price trade off within close proximity to government recognized school. The price and house details can still be discovered on a single webpage in some of the cases, for example, if the house is listed in the local community center but finding out the additional school details in a single webpage is nearly impossible in the current query processing scenarios. This kind of data lineage problem persists across numerous fields like search for conferences with desired time line or category, crime solving applications and database applications. Thus we address the problem of finding top-k graph results rather than top-k single nodes to answer user query.

The problem of finding a minimum cost connected tree or a graph result with fixed set of nodes is well-known in the literature as the Steiner-tree problem. Another immensely applied extension of this problem is the group Steiner tree problem defined as the minimum connected subgraph must contain at least one node from each sets of given vertices. There have been many research heuristic proposals recently based on the versions of Steiner tree problem like (Li, Candan, Vu, & Agrawal, 2001; Li, Ooi, Feng, Wang, & Zhou, 2008; Kasneci, Ramanath, Sozio, & Suchanek, 2009) with claims of efficient query processing. However, we adopt a simple search heuristic to process real time queries by reducing the size of the search space and reducing computation load by restricting the graph-exploration policy. We illustrate the differences between our approach with the rest of these approaches in the Related work Section 7.

We define the query processing framework  $F$  as an intuitive graph-exploration based strategy to integrate information from multiple related graph nodes. We first define the relevant nodes called *Keyword Nodes* as the set of nodes which contain at least one of the query keywords. The identification of *Keyword Nodes* is implemented through a prioritized index in order to reduce the query response model and consequently improving the efficiency. We also make use of the structural relationship to find common connecting paths between the *Keyword Nodes*. Each such connecting path that satisfies all the query keywords is a potential result sub-graph and is stored separately in an answer heap. In case of Web or large Database system, the number of potential candidates in answer heap could be very large, so we restrain and materialize only top  $t$  *Keyword Nodes*, where  $t$  is a fixed integer value (such that  $t > k$ , where  $k$  refers to the top-k result). Specifically, this restrained approach helps to reduce the size of the initial dataset for an  $n$  keyword query up to  $O(tn)$  which is much lesser as compared to the other approaches present in the literature. For instance, (Bhalotia, Hulgeri, Nakhe, Chakrabarti, & Sudarshan, 2002; Li et al., 2001, 2008) execute their search algorithm for all the graph nodes that match to at least one of the query keywords whereas we take only a fraction of the overall match. Typically, such reduction in the size of the search space facilitates our algorithm to explore the structural relationship using both incoming and outgoing edges within a computationally acceptable time frame.

Last but not the least, we propose a novel ranking function  $R(q_i, d_j)$ , where  $R$  is as defined above. Even though there is no standard benchmark to define the characteristics of a scoring function, we have proposed a novel ranking function called the graph ranking score (*GRS*) which incorporates the characteristics of current work from the literature. *GRS* grades the answer trees on the basis of semantic relationship by assigning highest grade to closely connected *Keyword Nodes* because lesser number of hyperlinks would intuitively refer to close affinity among the query keywords. In other words, the answer trees with fewer edges connecting the keyword nodes are ranked higher because of their proximity as compared to larger result trees. In summary, the key contributions of this paper are:

1. An efficient search algorithm that generates answer trees in simple steps without any complex computations.
2. A novel scoring function called the *GRS*, to rank answer trees on the basis of keyword proximity among the result.

The remainder of this paper is organized as follows. We provide a formal problem definition and solution model in Section 2 followed by a detailed graph-exploration algorithm in Section 3. We discuss the computational load of our proposed algorithm in Section 4 and propose our novel relevance score measure called *GRS* in Section 5. We discuss the algorithm performance through an extensive experimental work in Section 6 and finally conclude the paper along with future work in Section 7.

## 2. Problem definition

This Section formally defines the user query model, data model and the query response model adopted by our system. As mentioned earlier, we adopt Web as Graph approach with web pages as nodes and the corresponding hyperlink references as edges in the graph. Connection among the *Keyword Nodes* can be explored using both incoming and outgoing edges thus we prefer to adopt the undirected version of Web Graph. We model user query as a set of individual query keywords over an undirected graph  $G$  and the answer to user query should be in the form of a connected tree such that no proper sub-tree contains all the query keywords. Thus we first need to identify *Keyword Nodes* from the page-keyword mapping  $\sigma$  for each of the query keywords. We next describe the terms and notations used in this research work as follows:

- $G(V, E, W)$  graphical representation of problem domain where  $V$  denotes individual document or a single web page,  $E$  defines relationship connecting between nodes and  $W$  refers to weight function associated with  $V$ .
- $D$  denotes the finite set of indexable keywords.
- $Q = (k_1, \dots, k_n)$  refers to  $n$  keywords user query.
- $\sigma : D \rightarrow V$  is Keyword-to-page mapping, which maps each keyword to the set of nodes containing at least one occurrence of the specific keyword.
- $\rho : V \rightarrow V$  is Page-to-LinkedPage mapping, lists the set of related documents or webpages.
- $R_i$  denote the set of  $t$  *Keyword Nodes* such that  $R_i = \sigma(k_i)$  for any query keyword  $k_i$ .
- $R_i^h$  refers to the set of webpages that can reach through a link to or from at least one of the pages in  $R_i$  with a path of length  $h$ , for any positive integer  $h$ .

Let  $R = R_1 \cup R_2 \cup \dots \cup R_n$  be the initial Root set of relevant nodes for each of the query keywords. The algorithm needs to explore the neighboring nodes from each of these potential result candidates at hop level 1 or more using  $R_i^h$ . The trivial case,  $h = 1$  is the initial Root set only, i.e.,  $R_i = R_i^1$ . The Page-LinkedPage mapping  $\rho$  can then be used for each query keyword in order to expand  $R_i$  to  $R_i^h$  until the threshold values is reached or all the top-k answers have been generated.

**Example 1.** To illustrate the above defined notations, we present an example data graph shown in Fig. 1 where user query consists of three keywords,  $k_1$ ,  $k_2$  and  $k_3$ . Using the above notations,  $R_1 = \{9, 10\}$ ,  $R_2 = \{8, 10\}$  and  $R_3 = \{2, 5\}$ . After one hop expansion,  $R_1^2$  represents the set of nodes directly connected to or from the initial Root set  $R_1$ , i.e.,  $R_1^2 = \{5, 12, 7\}$ . Similarly,  $R_2^2 = \{3, 4, 5, 12, 13, 7\}$  and  $R_3^2 = \{0, 1, 6, 7, 8, 9\}$ . Similarly, we can use  $\rho$  to create other  $R_i^h$  on the fly using Page-to-LinkedPage mapping  $\rho$  as per the requirement of the query processing algorithm.

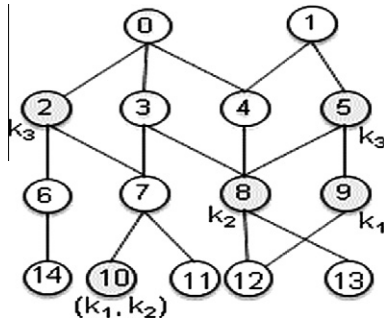


Fig. 1. Sample graph created for illustration of Root set and hop expansion.

### 3. Algorithm

In this section, we elaborate on our proposed query processing algorithms used to generate top- $k$  relevant answer networks in a highly efficient manner. Even though our approach looks similar to the Steiner tree/graph-based approach, we neither aim to construct an MST over the entire graph nor do we aim to compute the shortest path among all the pair of keyword nodes. In fact, we intend to find a common connecting path between *Keyword Nodes* at query execution time with cohesive meaning. We make a reasonable assumption that query keywords are highly correlated with one another and consequently proximally connected through hyperlinks. The algorithm proceeds progressively by generating highly correlated or connected answer first and then prevents the system from generating unnecessary number of results.

The general algorithm idea is similar to BANKS (Bhalotia et al., 2002), BLINKS (He et al., 2007), or Information Unit (Li et al., 2001), but we use different data structures and the expansion policy. We first identify the *Keyword Node* set  $R_i$  corresponding to each query keyword  $k_i$  using the Keyword-to-page mapping  $\sigma$ . In order to improve the efficiency of our conventional system (Lee, Arora, & Jo, 2010), we have already prioritized  $\sigma$  by the IR style textual-relevance-based tf-idf score (Hammami et al., 2006). A Search Engine like Google or AltaVista can also be used to generate this prioritized mapping. The algorithm next generates a boolean adjacency matrix  $R^E$  using Page-to-Linked Page mapping  $\rho$ , such that the nodes are in accordance with the Root set  $R$  and the  $(i, j)$  entry is 1 if node  $i$  and node  $j$  are directly connected through a hyperlink. If there exists a common node  $v \in \cap R_i$ , then this node satisfies user query and is thus a potential answer candidate and consequently added to the answer heap. Next, the algorithm explores the graph by traversing both the incoming and outgoing edges from each of the *Keyword Nodes*. After each exploration stage, the algorithm checks whether any tree exists which contains all the query keywords and appends it to the answer heap if such a subgraph exists else explores the graph one step further with the modified set of *Keyword Nodes*. Finally, all the candidate nodes or result subgraphs are enumerated by the GRS algorithm according to their relevance score.

#### Algorithm 1 $RG(Q, t, k)$

INPUT:  $Q, t, k$   
 OUTPUT:  $A$   
 1:  $R_i \leftarrow \phi, A \leftarrow \phi$   
 2: **for all**  $k_i \in Q$  **do**  
 3:    $R_i \leftarrow t$  pages from  $\sigma(k_i)$   
 4: **end for**  
 5:  $R \leftarrow R_i \cup R_j (\forall i, j)$   
 6: Create  $R^E$  from the elements of  $R$   
 7:  $FINDANSWER(R^E)$

#### Algorithm 1 (continued)

8: **while**  $(|A| < k)$   
 9:    $EXPAND(R)$   
 10:    $FINDANSWER(R^E)$   
 11: **end while**  
 12:  $GRS(A)$   
 13: **return**  $A$

#### Algorithm 2 $EXPAND(R)$

1: **forall**  $R_i \in R$  **do**  
 2:   **forall** node  $v \in R_i$  **do**  
 3:      $temp \leftarrow \rho(v)$   
 4:      $R_i \leftarrow R_i \cup temp$   
 5:      $NEWENTRY \leftarrow NEWENTRY \cup temp$   
 6:   **end for**  
 7: **end for**  
 8: Add each  $v \in NEWENTRY \rightarrow R^E$

We first describe the basic RETRIEVEGRAPH (RG) which accepts user query  $Q$ , the threshold value  $t$  to restrict the size of the individual Root set and  $k$  for retrieving the top- $k$  result. Line 1 initializes the basic variables used by the algorithm, i.e.,  $R_i$  and the answer heap  $A$  to null. Lines 2–4 identify the  $t$  most relevant pages  $R_i$  for each of the query keyword using the Keyword-to-page mapping  $\sigma$  (Lines 1–3). In order for efficient query processing,  $\sigma$  has already been prioritized using IR style based tf-idf relevance score. In Line 6, we create the reduced adjacency matrix  $R^E$  induced only on the elements of  $R$  using Definition 1. This reduced matrix  $R^E$  is next examined by the FINDANSWER subroutine to check for the presence of common nodes among  $R_i (\forall i)$  in Line 7. If all the desired top- $k$  results have been found, the algorithm scores the candidate answer tree through GRS otherwise, the algorithm explores the adjacent nodes using the EXPAND subroutine by traversing both the incoming and outgoing edges. Since we have reduced the size of our initial Root set to  $tn$  nodes ( $t$  pages for all the  $n$  nodes), it has become computationally feasible to expand the Root set for each and every *Keyword Node*. Moreover, we believe that the query keywords are not any random selection of keywords but they are logically correlated with each other and thus belong to the set of connected graph nodes. Since Page-to-LinkedPage mapping can be easily constructed from the problem domain graph  $G$  through the set of edges coming in and out of a node, hence it is query-independent and can be materialized in the memory beforehand.

#### Algorithm 3 $FINDANSWER(R)$

1:  $\phi \leftarrow \bigcup (R_i \cap R_j) (\forall i, j)$   
 2: **for all**  $v \in \phi$  **do**  
 3:   **if**  $v \in \cap R_i (\forall i)$  **then**  
 4:      $A \leftarrow A \cup v$   
 5:   **end if**  
 6: **end for**  
 7:  $P \leftarrow$  Set of all possible connected paths from  $R^E$   
 8: **for each** path  $p \in P$   
 9:   **if**  $p$  contains all query keywords **then**  
 10:      $A \leftarrow A \cup p$   
 11:   **end if**  
 12: **return**  $A$

**Definition 1.** The reduced symmetric adjacency matrix  $R^E$  is defined by its element  $e_{ij}$  as follows:

$$e_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ or } (j,i) \in E \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The purpose of the second algorithm *EXPAND(R)* is to manage graph-exploration separately from the other query processing activities. The algorithm accepts  $R$  the union of all Root sets  $R_i$  ( $\forall i$ ) as input and then for each node  $v \in R_i$  for each query keyword  $k_i$ , the algorithm explores the undirected edge and includes the newly accessed graph nodes in the original Root set  $R$  (Line 4). The reduced adjacency matrix  $R^E$  is also extended to include the newly visited nodes accordingly using the *FINDANSWER* algorithm. We have extended the adjacency matrix so that it becomes much more easy and efficient to find a common connecting path between different *Keyword Nodes* sets. Finally, the algorithm can be used to obtain the answer trees using both the modified Root set  $R$  and the extended adjacency matrix  $R^E$ . We illustrate the working of *FINDANSWER* algorithm in Fig. 4 for the example graph shown in Fig. 1. The algorithm examines all the connected paths through the column for example, there are two connected paths from node 5, one to node 8 and the other to node 9. The algorithm next examines the rows of node 8 and 9 to find out the other connecting nodes. Since they are zero for both node 8 and 9, the next step is to verify that the tree formed by connecting nodes 8–5–9 satisfies all the query keywords. In this case, the tree 8–5–9 contains all the query keywords and hence added to the answer heap  $A$ .

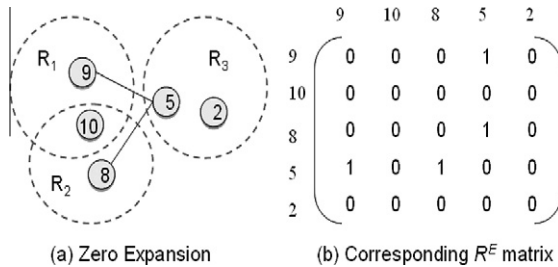


Fig. 2. The Root set expansion using RG algorithm. (a) Zero expansion refers to the case with no expansion (b) reduced adjacency matrix  $R^E$  corresponding to the nodes in the initial Root set.

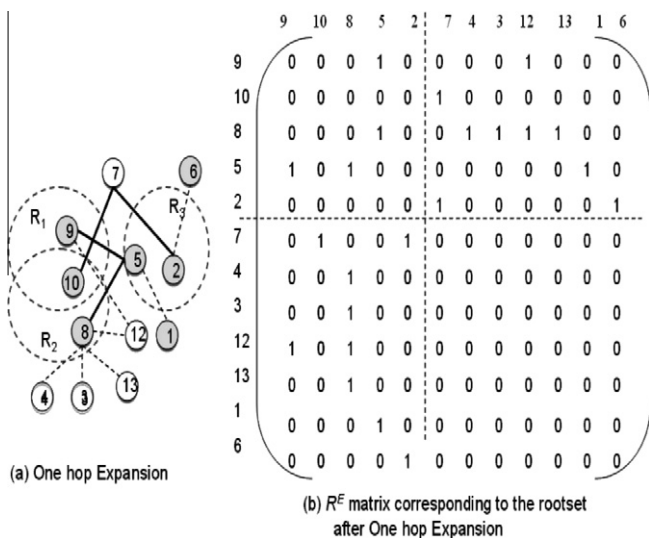


Fig. 3. The Root set expansion using RG algorithm. (a) One expansion refers to the case with one hop expansion (b) expanded adjacency matrix  $R^E$  corresponding to the nodes after one hop expansion.

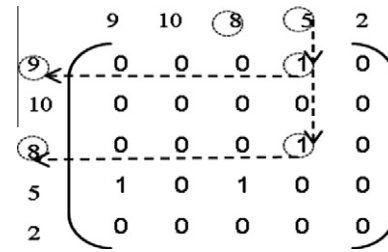


Fig. 4. Construction process of the final result subgraph using reduce matrix  $R^E$ .

**Example 2.** The RG algorithm when executed on the example graph 1, results in the initial Root set represented by Fig. 2(a). Since there is no common node with all the query keywords, thus the algorithm explores the neighboring nodes and then reexamines the new Root set for result subgraphs. The algorithm generates the first result as shown in Fig. 3. In order to highlight the presence of the result subgraph we have shown the connection edges using solid lines and the remaining exploring edges using dotted line. The algorithm proceeds further to the next hop level and generates the second result subgraph shown in Fig. 3(b).

#### 4. Computational load and expansion limits

As explained earlier in Section 3, the naive expansion approach will increase the problem size exponentially and if any one of the Root set pages is a portal node then, the algorithm will expand exponentially and might incur a serious computational hazard. Though, the algorithm will still perform correctly and find the result tree by exploring a common connection path. Therefore, we need an appropriate heuristic which can reasonably decrease the search space. In this paper, we make use of a heuristic that a node with higher relevance weight can expand more than the others. This is feasible if we decrease the node weight with each hop expansion, so that the nodes with higher weight can expand more than the nodes with smaller node weights. In simpler words, the weight of a page is set inversely proportional to the square of the expansion number. Therefore, we modify the weight of *Keyword Nodes* at each expansion step using Eq. (2) and the node expansion can occur only till the decreased weight is greater than or equal to the threshold value,  $\theta$ .

$$w' = \frac{w}{(h+1)^2} \geq \theta, \quad (2)$$

where  $w'$  is updated weight,  $h$  is node's expansion (hop) limit and  $\theta$  is a threshold.

Alternatively, we can state that the expansion limit ( $h$ ) of a *Keyword Node* with weight  $w$  can be obtained using  $\sqrt{\frac{w}{\theta}} - 1$ . We explain

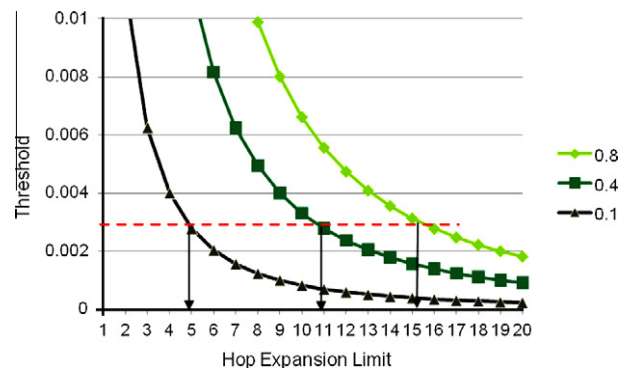


Fig. 5. Expansion limit for the graph nodes.



the relationship between node weight  $w$  and hop expansion limit  $h$  in Fig. 5. Given the initial node weights as 0.8, 0.4, 0.1 respectively for three different keyword nodes and assuming the threshold value  $\theta = 0.003$ , the corresponding node expansion limits can be found as 15, 10, 4 respectively, i.e., the most relevant *Keyword Node* with weight 0.8 can have expansion till 15 hops whereas the least relevant node with weight 0.1 can only have 4 hop expansion. The main advantage of restricting the expansion by modifying the node weights is to search among the neighbors of highly relevant nodes and reduce the computational load of the system. But a major disadvantage is that expansion limit decreases drastically with reductions in the node weights. For instance, if the node weights are 0.8 and 0.4 respectively but the threshold value is increased from 0.003 to 0.03, then the hop expansion limit also changes from 15 to 25 and 10 to 12. Thus with same edge weights as shown in Fig. 5 but a different threshold value, the expansion limit varies drastically up to 50% (from 25 to 12).

## 5. Relevance score

In this section, we discuss the other equally important aspect of an efficient IR system, meticulous ranking of the answer trees. The search result being a collection of trees, the optimal ranking score should not only incorporate the node weights and link score but also the subgraph topology. Though there has been no standard benchmark for enumeration scores, it must satisfy the following well accepted principles from previous ranking function:

1. **Representativeness:** Ranking function must assign a unique score to each distinct tree topology.
2. **Graph topology:** Any search result with compact tree structure, i.e., directly connected *Keyword Nodes* should be ranked higher as compared to the tree with Steiner nodes. This is so because larger tree structure would mean loose semantic relationship among the elements and thus the score should degrade with increase in number of edges.
3. **Monotonically increasing:** If the topology of the two result answer trees is identical but with different nodes from graph  $G$ , GRS should assign higher score to the result sub-tree with more relevant documents.
4. **Monotonically decrease with hop length:** If the weight of *Keyword Nodes* in two different networks are identical, the hop number must have a monotonically decreasing relationship with the GRS, i.e., relevance score should decrease with increase in hop number.

In addition to the above basic principles, we also need to focus on the difference between *Keyword Nodes* and connecting nodes. It is easier to calculate the relevance of *Keyword Node* with regard to query using the textual-relevance-based tf-idf measure but connecting nodes must have zero weight as their only significance is to connect the *Keyword Node*.

First, we define the column-normalized reduced symmetric adjacency matrix  $NR^E$  as follows:

**Definition 2.** The column-normalized reduced symmetric adjacency matrix  $NR^E$  is defined by its element  $n_{ij}$  as follows:

$$GRS_G = \left[ \frac{eig(NR^E)}{\|eig(NR^E)\|_1} \right] \cdot W^T, \quad (3)$$

where,  $eig(NR^E)$  is the principal eigenvector of the adjacency matrix  $NR^E$ .  $\|eig(NR^E)\|_1 = \sum_{i=1}^n$   $i$ th element of  $eig(NR^E)$  is the  $L_1$  norm of  $eig(NR^E)$ . And,  $W^T$  represents weight vector assigned to each node based on its textual relevance to the given query.

We use the eigenvector centrality to determine the centroid of the result subgraph, i.e., to find out which node of the network has more influence over the others. Eigenvector of the adjacency matrix represents eigenvector centrality of the nodes and has been used extensively in the social network search domain. We have adopted a similar semantic in the GRS, where the normalized principal eigenvector of the adjacency matrix,  $[eig(NR^E)/\|eig(NR^E)\|_1]$ , reflects the topological importance of the nodes. The eigenvector centrality is calculated such that each node's influence is propagated to its neighboring nodes but in decreasing order of the distance. Simply said, this implies that a central node with high edge connectivity generally has a higher value. GRS score satisfies monotonic decreasing relationship with hop number and monotonic increasing relationship with node's weight score.

**Example 3.** Consider the two result scenarios presented in row 6 and 7 of Table 1, where the two result trees have identical topology. Let us assume that node weights in the two result trees are  $w_1 = 0.5$ ,  $w_2 = 0.7$  and  $w'_1 = 0.7$ ,  $w'_2 = 0.7$  for row 6 and row 7 respectively. We have deliberately selected such node weights to highlight the fact that result tree for row 7 will be assigned higher GRS score due to their higher individual node weights. Table 1 also emphasizes another important aspect of GRS, the first and the second row (1,2) are highly ranked due to the closer connectivity of the graph nodes whereas rows 3 and 4 have lesser rank because of the presence of intermediate nodes between the *Keyword Node*. It also highlights the essential properties of GRS, i.e., it satisfies monotonic decreasing relationship with hop number and monotonic increasing relationship with individual weights of *Keyword Nodes*. It can easily be verified that nodes within close proximity of each other are assigned higher rank as compared to result trees with greater number of steiner nodes. The following theorem can tell us that the GRS is used as a graph ranking factor for this approach.

**Theorem 1.** Given the reduced symmetric adjacency matrix  $R^E$ , the GRS is monotonic in terms of the path length  $h$ , where  $h = n$  for  $n \geq 2$ .

**Proof.** Let  $M_n$  be a  $n \times n$  tridiagonal matrix whose diagonal is  $(\frac{1}{2}, \frac{1}{3}, \dots, \frac{1}{3}, \frac{1}{2})$ , whose subdiagonal is  $(\frac{1}{2}, \frac{1}{3}, \dots, \frac{1}{3})$ , and whose superdiagonal is  $(\frac{1}{3}, \dots, \frac{1}{3}, \frac{1}{2})$ . Then the principal eigenvector is the non-trivial solution of the equation

$$(M_n - I)x = 0, \quad (4)$$



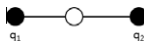
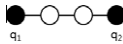
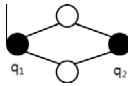
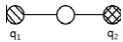
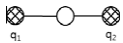
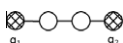
where  $I$  be the  $n \times n$  identity matrix. By matrix calculation, the non-trivial solution normalized by its one norm can be written as

$$x = \left( \frac{2}{3n-2}, \frac{3}{3n-2}, \dots, \frac{3}{3n-2}, \frac{2}{3n-2} \right)^T. \quad (5)$$

Since the  $w = (1, 0, \dots, 0, 1)$ , the  $GRS = \frac{4}{3n-2}$  for  $n \geq 3$ . Note that  $GRS = 1$  when  $n = 2$ . Therefore GRS decreases as  $n$  increases when  $n \geq 2$ .  $\square$

Another issue is whether the matrix  $E$  is stable or not to be used as a ranking measure for the graph result. It is well known that if a matrix  $E$  satisfies nonnegative, irreducible, and symmetric properties then a matrix  $E$  have a definitive, non duplicate positive dominant eigenvalue  $\lambda$  and corresponding eigenvector  $x$  (Brualdi, 2009). The adjacency matrix  $NR^E$  defined above contains connection or link information among the nodes, so, its element value is positive number. Therefore, the adjacency matrix  $NR^E$  satisfies the nonnegative property. Also, the networks node are wholly connected to each others, the adjacency matrix of the networks

**Table 1**  
GRS cases for results subgraph topologies.

	Topology	Adj. Matrix ( $NR^E$ )	$\frac{eig(NR^E)}{\ eig(NR^E)\ _1} \times W^T$	GRS
(1)		[1]	$(1)(1)^T$	1
(2)		$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$	$(0.5 \ 0.5) \begin{pmatrix} 1 \\ 1 \end{pmatrix}$	1
(3)		$\begin{bmatrix} \frac{1}{2} & \frac{2}{3} & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$	$(0.286 \ 0.428 \ 0.286) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$	0.572
(4)		$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{2} \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$	$(0.2 \ 0.3 \ 0.3 \ 0.2) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	0.4
(5)		$\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$	$(0.25 \ 0.25 \ 0.25 \ 0.25) \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	0.5
(6)		$\begin{bmatrix} \frac{1}{2} & \frac{2}{3} & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$	$(0.286 \ 0.428 \ 0.286) \begin{pmatrix} 0.5 \\ 0 \\ 0.7 \end{pmatrix}$	0.343
(7)		$\begin{bmatrix} \frac{1}{2} & \frac{2}{3} & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$	$(0.286 \ 0.428 \ 0.286) \begin{pmatrix} 0.7 \\ 0 \\ 0.7 \end{pmatrix}$	0.4
(8)		$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{2} \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} \end{bmatrix}$	$(0.2 \ 0.3 \ 0.3 \ 0.2) \begin{pmatrix} 0.7 \\ 0 \\ 0 \\ 0.7 \end{pmatrix}$	0.28

satisfies the irreducible property and symmetric property self-evidently. Consequently, it is guaranteed that there will always be a principal eigenvector and values available for computing the GRS.

## 6. Experiment

In this section, we present the empirical analysis of our graph-exploration-based RG with result network topology based ranking measure GRS to substantiate their respective operational ability and usefulness for real world applications. We have compared our work with the highly well known search heuristic of Information Unit (IU) (Li et al., 2001) because of its highly similar query processing algorithm. Even though EASE (Li et al., 2008) and BLINKS (He et al., 2007) are also well known among the research community, but they both partition the data graph and maintain indices on these partitions making it unsuitable for comparison with our approach. Difference from other state-of-the-art search

systems have been explained in Section 7 (related work). The aim of this experimental analysis is to evaluate the search efficiency and performance of our approach. The query processing time and precision of the search result have been the conventional measures to evaluate system performance. But in both the systems, query processing starts with the identification of keyword nodes followed by an attempt to find a common connecting path, hence the percentage of graph explored also forms an important parameter of the search heuristic especially in comparison to the other proposed heuristics. Last but not the least, we compare the average cost of producing the top-k result using the weight of the result subgraphs.

**Implementation:** The experiments have been performed on an Intel Core 2 Duo personal computer with 2.13 GHz processing power and 1 GB of RAM on Windows XP platform. We implemented the three algorithms defined earlier as RG, EXPAND and FINDANSWER using Java programming language. The Keyword-to-page mapping  $\sigma$  and Page-to-LinkedPage mapping  $\rho$

**Table 2**

Parameters used for generating the Synthetic data set.

No. of Nodes	No. of Arcs	Min degree	Max degree
100	464	3	15
500	2022	6	22
1000	4413	8	50

are implemented as in-memory structures using Java HashMap class. Each document or page is assigned a unique integral identifier using another in-memory hash map structure. The high priority  $t$  nodes can then be retrieved from  $\sigma$  and  $\rho$  within one disk access. We experimented for different values of  $t$  using various distributions of keyword nodes among synthetic data graphs and find out that  $t = 30$  works well for generating up to top 50 results. Even though we report results for up to top 60 subgraphs, it is not unreasonable to assume that no user actually visits after top-30 results. Hence we have assumed  $t = 30$  for our experimental study. The ranking function GRS is based upon eigenvalues and matrix multiplication operation, so we have used MATLAB for computing the rank of result subgraphs.

**Data set:** To test the operational ability of our system, we first use dummy data created synthetically using random numbers and then proceed with real world data. In order to focus on both the heavily connected graphs with larger number of edges connecting fewer nodes and highly populated graphs, we use the parameters listed in Table 2 to create 15 synthetic data graphs; 5 for each row combination. The real web data has been obtained from the Stanford University dataset which generates an undirected graph with 538 nodes and 26,698 edges respectively. For each evaluation parameter, we tested system performance with different distribution of *Keyword Node* in the case of synthetic data set and 10 queries in case of Stanford data set. The query keywords have been uniformly distributed over the search graphs so that the desired query result is a combination of single nodes, node pairs and sub-graph. Also, the query length varies from two to six keywords for both data sets. We have not considered single keyword query because it can be very easily solved using the

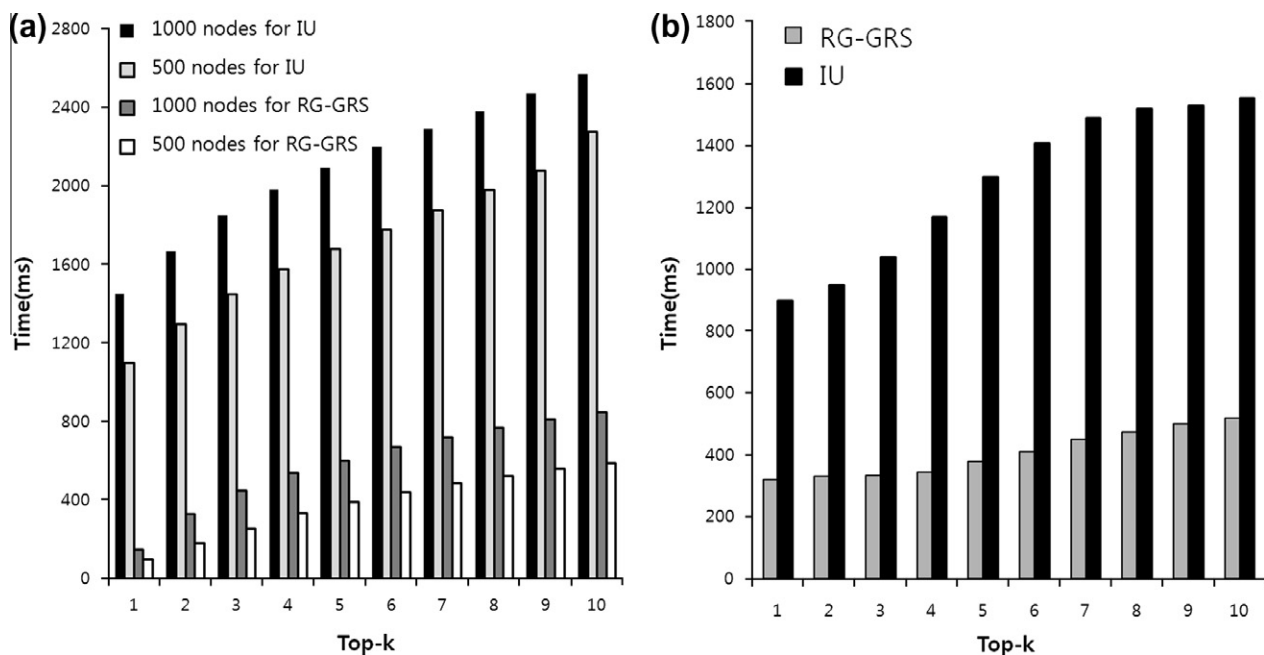
maintained index in case of IU or mappings in case of RG with GRS.

### 6.1. Search efficiency and performance

The first test we conducted was to evaluate the overall search performance of our system using the query processing time and precision of the top-k result. We first compare the query processing time to explore how fast the two search algorithms can respond to user query for varied top-k result. The results for processing time metrics are summarized in Fig. 6(a) for synthetically created graphs and 6(b) for Stanford dataset based real web data respectively. Since we created 5 data graphs each for 500 and 1000 nodes, we took an average over these graphs and present the result for 3 keyword query.

As shown in Fig. 6(a), the query processing time does not increase much with increase in the input graph size for RG-GRS. In contrast, the processing time increases rapidly with increase in graph size for IU (Li et al., 2001). The reason for such behavior is selection of only top  $t$  nodes from the prioritized keyword-page mapping  $\sigma$  which is independent of the given data graph size whereas IU (Li et al., 2001) processes the induced subgraph over all the *Keyword Nodes*. The same deduction can be made for real web data. As expected, query processing increases with top-k result for both dummy and real data set. We have not reported the result for 4, 5 and 6 query keywords because they also had similar behavior. Hence RG-GRS can process large data sets with same efficiency as smaller data graphs due to the selection of prioritized nodes for query processing.

The next performance metric is top-k precision which indicates the fraction of relevant results among top-k result. The result sub-graph relevance is judged by human experts for real web data and by the highest score of result subgraphs in case of synthetic data-set. The results have been summarized in Fig. 7(a) and (b). Due to the prioritized keyword-page mapping  $\sigma$ , highly relevant web pages are always included in the initial Root set  $R_i \forall i$ . Even the experimental results indicate that our proposed search system satisfies the most fundamental IR property that none of the relevant answers are missed by the system.

**Fig. 6.** Query processing time for (a) synthetic data with 500 and 1000 nodes (b) real web data.

## 6.2. Graph exploration statistics

We next evaluate the practicality of the two algorithms in terms of the number of nodes and edges explored for generating the top-k result. Fig. 8 reports the explored percentage of graph as ratio of number of nodes visited to total number of nodes in graph. As can be seen from Fig. 8, IU explores much larger number of nodes as compared to our system because of creating MST over a large induced subgraph which increases drastically with the number of keywords in the user query.

## 6.3. Average cost

Finally, we compare the average cost of our proposed RG with IU and report our findings in Fig. 9. This plot represents average cost as function of number of keywords and indicates the difference in ranking measure adopted by the two schemes. IU adopts distance based heuristic and does not consider the relevance of node weights while computing the cost of generated result subgraph. Further, final result is displayed by sorting all the generated result subgraphs by their cost. RG on the other hand selects only a fraction of all the *Keyword*

*Node* prioritized by their relevance to keyword queries. The result subgraphs are then generated progressively in decreasing order of proximity or increasing order of distance. GRS assigns cost based on the subgraph topology so that directly connected or closely connected *Keyword Node* are ranked higher. The score decreases or in other words the cost of the subgraph increases with increase in number of connecting Steiner nodes. Fig. 9 clearly illustrates the difference in cost between GRS and IU. As depicted in plots of Fig. 9 the curve trend is very slightly upwards with increase in the number of keywords for RG-GRS because as the number of keywords increase so does the probability that they are distributed among distant nodes in the graph. But the graph for RIU increases by very large extent due to its simpler distance based heuristic and sub-optimal nature. Once more we found that our selective node expansion based RG and topology based GRS (with integral node weight and edge weight components) produces much better result as compared to IU.

## 7. Related work

There have been extensive research works that investigate the topology of graph searches and web or xml searching (Bhalotia

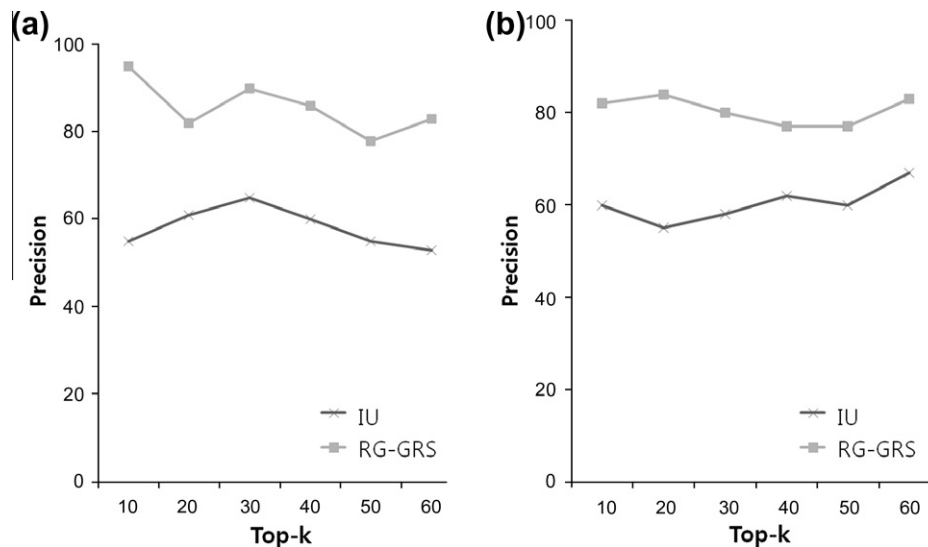


Fig. 7. Search accuracy for (a) synthetic data (b) real web data.

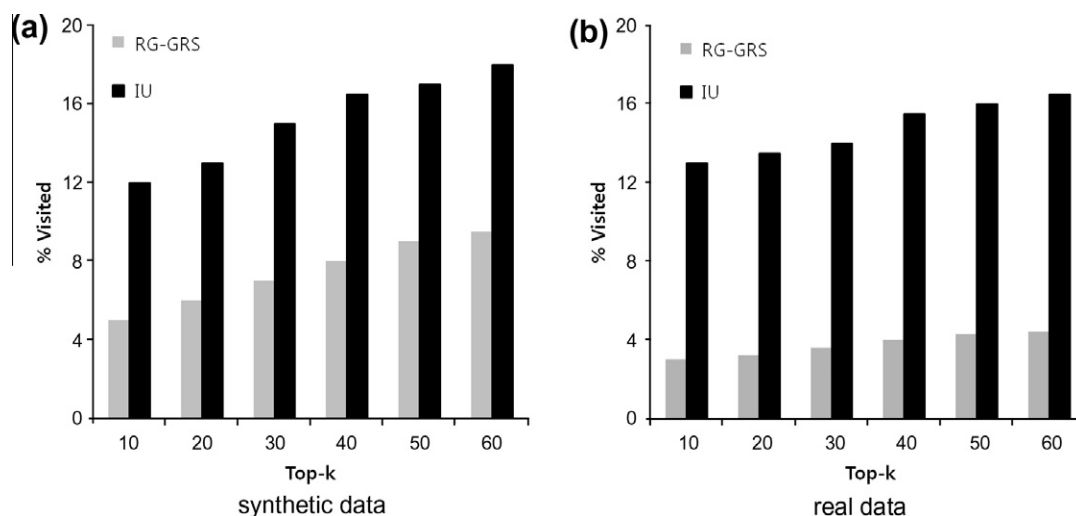


Fig. 8. Percentage of nodes visited as against the total graph size by RG-GRS and IU algorithms over real and synthetic data.



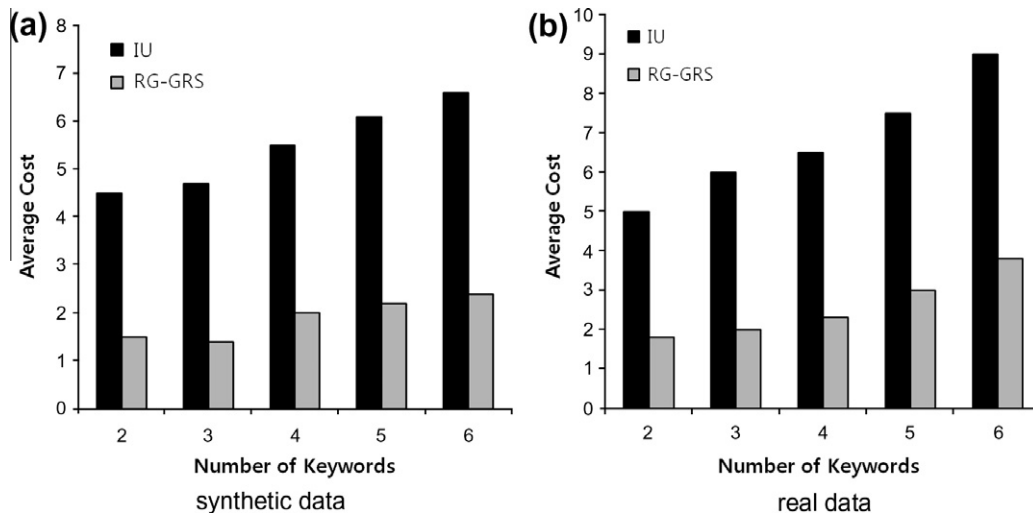


Fig. 9. Average cost computed over real and synthetic dataset for top-k result.

et al., 2002; Chan, Jagadish, Tan, Tung, & Zhang, 2006; Cohen, Kimelfeld, & Sagiv, 2009; Shin et al., 2007). However, for the navigation perspective, graph per se is too complex and inefficient because there are so many hyperlinks and cycles in the graph. Graph theoretical approaches have known to be useful for estimating the overall size of the Web (Jeong, Tombor, Albert, Oltvai, & Barabasi, 2000).

There are learning-based Web query processing approaches (Batzios, Dimou, Symeonidis, & Mitkas, 2008; Das & Türkoglu, 2009). Infilling the gulf between the user interest and keywords in the user query, their systems learn from the user when the user browses through the first answers. In order to obtain useful information, the user may need to browse through a ranked set of URLs. Moreover, their system reduces the user's burden of webpage browsing by returning only some relevant portions of a webpage, instead of the entire webpage (Lee, Leung, & Lee, 2011). However, if the user's desired information is not on a single webpage, their system would not give all the information in one shot. In contrast, our proposed method provides pairs of Web objects as answers so that user can get his desired information without browsing many different webpages. Moreover, we use the hyperlink structure to obtain the Web object pairs as the final answer.

Regarding the internal links of a Website follow a different structure than the link structure of the Web (Xue et al., 2003), they used implicit links to analyze the internal links of a Website or a domain. They found user browsing behaviors by calculating the conditional probability which represents that a webpage is visited after visited webpages. They maintained a user log for each user session, extracted information from this log to create implicit links, and thus detected the user browsing process. They used probability theory to decide which sample set of WebPages can be used to represent a Website, so that the numerical confidence value for each Webpage is calculated to decide whether or not the Webpage can represent the Website. Hence, the approaches provide a set of WebPages for every Website that might be relevant to the user query (Lee & Geller, 2004). In contrast, our system provides Web graphs. Instead of creating and maintaining user logs, we use existing hyperlinks or given links. Moreover, our system does not need any prior knowledge for interpreting user requirements as we follow the existing link structure. Since web log mining can be used to optimize web domain structures based on browsing patterns; it can also be exploited to measure the degree of webpages (Das & Türkoglu, 2009; Hammami et al., 2006; Tao, Hong, Lin, & Chiu, 2009). However, these Web log mining approaches ignore personal user preference and assign a weight to a Web object primarily

based on the number of visits or "hit ratio" or "exposure" for the users.

As stated by Neumann and Weikum (2010), the research communities need to adopt the non-technical user model and the query processing should include both matching of keywords and ranking the result obtained. During the last decade, there have been various research heuristics to address the issue of keyword query processing in relational databases (Luo, Lin, Wang, & Zhou, 2007; Shin et al., 2007). On the other hand, a proximity search based on the Steiner tree algorithm have been used to find an approximate tree including webpages or xml to cover user keywords (Bhalotia et al., 2002; Kimelfeld & Sagiv, 2007), but they were not designed for web navigation nor for graph relevance. Moreover, there are no arc weights for comparing alternatives (Luo et al., 2007). The use of keyword proximity search for finding subgraphs as k-fragment (Barcelo, Hurtado, Libkin, & Wood, 2010; Cohen et al., 2009; Luo et al., 2007), which combines database queries with information retrieval can be applicable to find approximate top-k results (Ajayi, Aderounmu, & Soriyan, 2010; Lee & Geller, 2004; Shin et al., 2007).

There are a lot of linkage based measures such as variation of PageRank or HITS, or a semantic for Web service (Kimelfeld & Sagiv, 2006; Langville & Meyer, 2006; Tseng & Wu, 2007; Yang, 2008). Link ranking measure for hyperlink representations can be incorporated by the descriptive information on the target document. Empirical results on some related work showed (Langville & Meyer, 2006). There are many weight or rank measurements, including global weights (e.g., PageRank and HITS measures) (Kimelfeld & Sagiv, 2006; Langville & Meyer, 2006; Lee & Lim, 2007) which can be derived independently from user queries-and local weights (e.g., tf-idf from vector space models) (Kimelfeld & Sagiv, 2007; Hammami et al., 2006; Li et al., 2001). There are also some other measures-such as probabilistic model (e.g., Google Similarity (Cilibrasi & Vitanyi, 2007)), two Poisson models (Hammami et al., 2006) that are worth considering for web objects, not for the graph.

## 8. Conclusions

A graph ranking method is presented to discover subgraphs to process keyword queries over graph data. We investigated a prioritized keyword to document mapping mechanism so that the graph alternatives are derived in terms of the textual relevance. Secondly, in order to have an in-memory representation with

efficient storage we maintain a relevant amount of the total available data, by selecting only  $t$  prioritized graph nodes so that the size of the search space can be reduced. Additionally, the algorithm produces results in progressive order so that the system can produce results as they are generated by the algorithm. To facilitate efficient query processing, we also propose a graph-based ranking measure called *GRS*. The method assigns a relevance score such as the smaller the steiner node are, or nodes within close proximity. In the experiment, our method outperformed to conventional approaches so that the high performance and accuracy comes from effectively evaluating subgraphs over the graph-structured data.

## Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0026441).

## References

- Ajayi, A. O., Aderounmu, G. A., & Soriyan, H. A. (2010). An adaptive fuzzy information retrieval model to improve response time perceived by e-commerce clients. *Expert Systems with Applications*, 37(1), 82–91.
- Barcelo, P., Hurtado, C. A., Libkin, L., & Wood, P. T. (2010). Expressive languages for path queries over graph-structured data. In *PODS* (pp. 3–14).
- Batzios, A., Dimou, C., Symeonidis, A. L., & Mitkas, P. A. (2008). Biocrawler: An intelligent crawler for the semantic web. *Expert Systems with Applications*, 35(1–2), 524–530.
- Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., & Sudarshan, S. (2002). Keyword searching and browsing in databases using banks. In *ICDE* (pp. 431–440).
- Brualdi, R. A. (2009). Combinatorial matrix analysis. *Encyclopedia of Optimization*, 377–382.
- Chan, C.-Y., Jagadish, H. V., Tan, K. -L., Tung, A. K. H., & Zhang, Z. (2006). Finding k-dominant skylines in high dimensional space. In *SIGMOD* (pp. 503–514).
- Cilibrasi, R. L., & Vitanyi, P. M. B. (2007). The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3), 370–383.
- Cohen, S., Kimelfeld, B., & Sagiv, Y. (2009). Running tree automata on probabilistic XML. In *PODS* (pp. 227–236).
- Das, R., & Türkoglu, I. (2009). Creating meaningful data from web logs for improving the impressiveness of a website by using path analysis method. *Expert Systems with Applications*, 36(3), 6635–6644.
- Ding, B., Jeffrey, X., Wang, S., Lu, Q., Zhang, X., & Lin, X. (2007). Finding top-k min-cost connected trees in databases. In *ICDE* (pp. 836–845).
- Hammami, M., Chahir, Y., & Chen, L. (2006). Webguard: A web filtering engine combining textual, structural, and visual content-based analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2), 272–284.
- He, H., Wang, H., Yang, J., & Yu, P. S. (2007). Blinks: Ranked keyword searches on graphs. In *SIGMOD* (pp. 305–316). New York, NY, USA: ACM.
- Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N., & Barabasi, A.-L. (2000). The large-scale organization of metabolic networks. *Nature*, 407(6804), 651–654.
- Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F. M., & Weikum, G. (2009). Star: Steiner-tree approximation in relationship graphs. In *ICDE* (pp. 868–879).
- Kimelfeld, B., & Sagiv, Y. (2006). Finding and approximating top-k answers in keyword proximity search. In *PODS* (pp. 173–182).
- Kimelfeld, B., & Sagiv, Y. (2007). Matching twigs in probabilistic XML. In *VLDB* (pp. 27–38).
- Langville, A. N., & Meyer, C. D. (2006). A reordering for the pagerank problem. *SIAM Journal on Science on Computing*, 27(6), 2112–2120.
- Lee, W., & Geller, J. (2004). Semantic hierarchical abstraction of web site structures for web searchers. *Journal of Research and Practice in Information Technology*, 36(1), 23–34.
- Lee, W., & Lim, T. (2007). Architectural measurements on the world wide web as a graph. *Journal of Information Technology and Architecture*, 4(1), 61–69.
- Lee, W., Arora, N. R., & Jo, T. -C. (2010). Discover hierarchical subgraphs with network-topology based ranking score. In *C3S2E* (pp. 66–74).
- Lee, W., Leung, C. K., & Lee, J. J. (2011). Mobile Web Navigation in Digital Ecosystems Using Rooted Directed Trees. *IEEE Transactions on Industrial Electronics*, 58(6), 2154–2162.
- Li, G., Ooi, B. C., Feng, J., Wang, J., & Zhou, L. (2008). Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, (pp. 903–914).
- Li, W.-S., Candan, K. S., Vu, Q., & Agrawal, D. (2001). Retrieving and organizing web pages by information unit. In *WWW* (pp. 230–244).
- Luo, Y., Lin, X., Wang, W., & Zhou, X. (2007). Spark: Top-k keyword query in relational databases. In *SIGMOD* (pp. 115–126). New York, NY, USA: ACM.
- Neumann, T., & Weikum, G. (2010). The rdf-3x engine for scalable management of rdf data. *VLDB Journal*, 19(1), 91–113.
- Shin, M. K., Huh, S. Y., & Lee, W. (2007). Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy. *Expert Systems with Applications*, 32(2), 469–484.
- Tao, Y.-H., Hong, T.-P., Lin, W.-Y., & Chiu, W.-Y. (2009). A practical extension of web usage mining with intentional browsing data toward usage. *Expert Systems with Applications*, 36(2), 3937–3945.
- Tseng, J. C. R., & Wu, C.-H. (2007). An expert system approach to improving stability and reliability of web service. *Expert Systems with Applications*, 33(2), 379–388.
- Xue, G., Zeng, H., Chen, Z., Ma, W., Zhang, H., & Lu, C. (2003). Implicit link analysis for small web search. In *SIGIR* (pp. 56–63).
- Yang, S.-Y. (2008). An ontological website models-supported search agent for web services. *Expert Systems with Applications*, 35(4), 2056–2073.