TOWARDS ADVANCED DATA RETRIEVAL FROM LEARNING OBJECTS REPOSITORIES

VALENTINA PAUNOVIĆ
Belgrade Metropolitan University, valentina.paunovic@fit.edu.rs

SLOBODAN JOVANOVIĆ
Belgrade Metropolitan University, slobodan.jovanovic@metropolitan.edu.rs

Abstract: *In improving efficiency of creating learning materials, fundamental role plays concept of reusability. In order to allow effective exploitation of its content, a repository of learning objects have to enable search procedure which is powerful and at the same time intuitive and simple for use. We propose an architectural solution for enhanced search, such that both requirements are satisfied. A search algorithm based on finding min-cost Steiner trees allows finding not only learning objects which satisfies given query, but at the same time, it enables finding implicit relationships among different concept. To enable application of such algorithm, we developed a novel algorithm for sparse weighted graph representation of a LO repository. In addition, user's ability to retrieve relevant information can be further improved by extension of query language. We proposed one possible extension based on formal logic and designed an algorithm for parsing such language.*

Keywords: *E-Learning, Steiner trees, Query language*

1. INTRODUCTION

Considering the emerging popularity of personalized distance-based learning in many institutions, there is constant growing demand for more effective creation of educational materials. A key concept in improved efficiency of this process is *reusability* of already created learning content. Fundamental working unit of teaching material in e-Learning is *learning object (LO)*, defined by LOM standard as "any entity, digital or non-digital, that may be used for learning, education or training" [1][2]. Created learning objects are organized and stored in *LO repositories*, from where they can be searched and retrieved when necessary. The previous definition of LO allows inclusion of material in various formats (textual, image, video, etc.) which present serious challenge in organizing and searching tasks. In order to improve these procedures, learning objects are enriched by additional description through *metadata*.

Central role in data retrieval from a LO repository is *textual search*. Even if the targeted LO is not of textual type, various metadata are given as plain text (title, keywords, description, etc.) and, therefore, are subject to textual search. In addition, most of teaching material is in textual format, which emphasize necessity for this type of search even more.

The aim of this paper is to present an architectural solution for effective textual search in large LO repositories. Instead of traditional search encountered in web browsers and textual processing applications, we propose search based on finding Steiner trees. This approach has two main advantages:

- Even if there is no object which satisfies all terms from a query, it is possible to detect set of minimal number of closely related objects such that each term from the query is present in at least one of the objects.
- It is possible to detect implicit relationships among learning objects.

Besides search procedure, efficiency of search procedure depends on used query language. Traditionally, query language usually does not provide any operators. Query is composed of terms and it is assumed that only one operator is between them - operator and. For example, query "mathematical physics" is equivalent to request "find all objects which contains term mathematical ***and*** term physics". Such convention does not support submitting query which corresponds to request like "find all objects which contains terms mathematical ***or*** chemical physics."

A novel contribution of this paper consists of the following:

- We design an algorithm for creating sparse weighted graph representation of a learning objects repository, which is suitable for application of algorithm for finding Steiner trees.
- We propose an extension of query language based on formal logic and an algorithm for parsing such language.

Architecture of the system is presented in Figure 1. LORMS (Learning Objects Repository Management System) creates graph representation of LOR which is used by search engine to obtain results. Query has to be processed by query parser before performing search procedure. Details of this steps are explained in the rest of the paper.

[1]

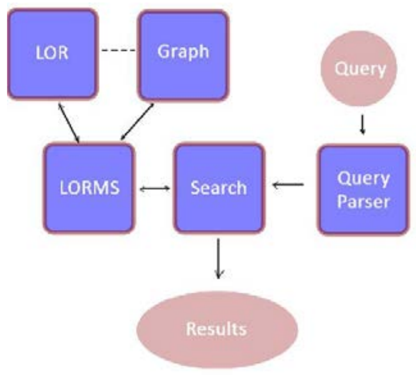


Figure 1. Architecture of search system

2. DATA RETRIEVAL FOR LO REUSABILITY

Starting point in the most common type of textual search is forming a query (Q) composed of relevant terms. Typing *function limit* in a web browser will result in web pages which contain both terms – *function* and *limit*. However, the challenge arises with increasing query complexity. For example, what is a result if there is no page containing all terms from query? The answer depends on search engine. For example, Google search engine will try to eliminate part of the query and perform search on modified queries, thus, retrieving pages which partially satisfy the query.

In searching through LO repositories, this might not be the desired solution. Instead of trying to return one LO which satisfies the complete query, which may result in empty resulting set if there is no such query, search engine could try to return a set of learning objects, such that the whole set satisfies query, even if a particular element of the set does not. In the previous example, if there is no such LO which contains both terms (*function* and *limit*), the result of search procedure can be set of two LO, one of which contains *function*, and the other contains *limit*. In addition, this type of search is suitable for finding implicit relationships among concepts, which can be useful tool in creating learning material. Finally, results of this type of search can be used as a recommender system which provides possible directions how to proceed further after completing explanation of one concept.

The proposed approach raises an important issue which accompanies every search process – *ranking* of the obtained results. Obviously, there could be more than one LO which contains term *function* and, similarly, more than one LO which contains term *limit*. Any combination from these two groups is an answer to the query, but it is not obvious in which order these answers should be presented to the user. In addition, queries with more than two terms could result in more complicated structures. For example, in case of a three-term query, the final result can consist of one, two, or three LO, with various combinations which LO contains which term from the query. Intuitively, a valid solution to the previous problem should satisfy the following two conditions:

- [C1] Results which contain smaller number of LO correspond to stronger relationships among terms from query and should have advantage in rankings (the best solution consist of only one LO);
- [C2] Results which contain more similar LO (from the same area or subject) correspond to stronger relationships among terms from query and should have advantage in rankings.

The second condition introduces some ambiguity and needs further clarification. Let’s say that there is a function *sim*(LO1, LO2) which return degree of similarity between learning objects LO1 and LO2. Specifying such function will be one of topic in the rest of the paper. Such function enables creation of a weighted graph whose vertices represent learning objects from repository and edge weights are determined by function *sim*. Each vertex is characterized by additional attributes – terms which corresponds to words from LO’s body or its metadata.

In the rest of the paper, we will see how to specify function *sim*, create a graph representation of the repository and perform search on such graph in an efficient way.

3. LO REPRESENTATION AND SIMILARITY MEASURE

Vector space model

Let $R = \{d_1, d_2, ..., d_r\}$ be a repository of learning objects and $W = \{w_1, w_2, ..., w_k\}$ the set of all distinct terms from R. The set W is obtained by the following procedure:

```
foreach learning object d in R
        foreach word w in d
                if(w not in stop_words)
                        stem(w);
                        add w to W;
```

In the previous algorithm, when searching for words in a particular LO, one has to search for words from the content of LO and various metadata – title, keywords, description, etc. All parts of LO from which words are collected will be referred as LO slots in the rest of the paper. Non-descriptive stop words are excluded (like articles, prepositions, conjunctions, etc.), while other types of words are stemmed to obtain their base or root. For example, words fishing, fished and fisher are all reduced to root fish.

For representing learning objects we will use vector space model, a common way to represent documents in various NLP tasks. According to it, each LO from the repository R is represented as an m-dimensional TF-IDF vector

$$\vec{r}(d) = (tfidf_1, tfidf_2, ..., tfidf_m) . \qquad (1)$$

(

[1] This paper provides some results obrained by the project with code III44006 financed by the Ministry of Education, Science and Technological Development of Rebublic of Serbia

It is determined in the following way. First, for each term of a learning object d, value tf_i is calculated as its weighted frequency:

$$tf_i = \sum_j h_j n(i, j) \ , \tag{2}$$

where

- $n(i, j)$ is number of occurrences of term w_i in the j-th slot of LO d;
- h_j is a weight associated with the j-th slot.

Second, inverse document frequency is calculated in the following way:

$$idf_i = \log \frac{|R|}{|\{d \in R : w_i \in d\}|}, \tag{3}$$

where |A| is cardinal number of a set A. The role of idf component is to reduce impact of words which are frequent across all documents and, thus, have small discriminating power. Finally, i-th component of the vector is calculated as product of term frequency and inverse document frequency:

$$tfidf_i = tf_i * idf_i \, . \tag{4}$$

Weighted sum in tf component is used in order to emphasize impact of certain LO slots. In search process, weights are assigned according to priorities:

- The highest impact (weight) should have terms from metadata title, keywords and description.
- Medium impact is reserved for terms from content (if there is textual content).
- Terms from the rest of searchable metadata should have low impact.

LO similarity

Text similarity has been studied in various contexts of NLP tasks. Probably the most popular and used is cosine distance which corresponds to correlation between two vectors. For two learning objects LO1 and LO2, cosine similarity is defined as cosine of angle between vectors which represent these learning objects in vector space model:

$$sim(d1, d2) = \frac{\vec{r}(d1) \bullet \vec{r}(d2)}{\|\vec{r}(d1)\| * \|\vec{r}(d1)\|} \ , \tag{5}$$

where $\bullet$ indicates scalar product of two vectors and $\|\vec{r}(d)\|$ denotes the intensity of vector $\vec{r}(d)$. Although cosine can have values in interval [-1, 1], function *sim* can have values from interval [0, 1] because all components of vectors in vector space model are positive. If similarity is perfect, *sim* returns 1; lower values correspond to lower similarity degree.

Similarity measure defined by (5) returns larger values for objects which exhibits more significant similarity. There are situations where the opposite is required, in which case distance measure is used instead of similarity measure. Obviously, distance and similarity measure are correlated and each of them can be defined by using the other one. For cosine similarity, we can define distance measure in one of the following ways:

$$dist(d1, d2) = 1 - sim(d1, d2) \, , \tag{7}$$

$$dist(d1, d2) = -\log(sim(d1, d2)) \, . \tag{8}$$

4. GRAPH REPRESENTATION OF LO REPOSITORY

Specified similarity measure between learning objects enables creating graph representation of repository. Graph G={V, E} is created in the following way:

- The set of vertices V corresponds to the set of learning objects. Each LO is represented in the graph by one vertex.
- Each vertex of the graph is enriched by the set of attributes A, which corresponds to the set of words by which LO can be searched. These words are from all searchable slots of the LO.
- Initially the set of edges E contains edges between every two vertices. The weight of each edge is defined by the function (4) *dist* and corresponds to the distance between appropriate learning objects. The reason for using distance instead of similarity measure for edge weights will be explained later in section which deals with the search algorithm.

Graph representation allows us to formalize through graph-theory terminology the central problem in this paper:

Problem definition (MCGST-k) – Let G={V, E} be an undirected weighted graph with set of vertices V and set of weighted edges E. Each vertex v_i from V is characterized with set of attributes A_i. Let Q be a query which consists of n terms. An answer tree to a query is any tree from G, such that each term from Q is contained in the set of attributes of at least one vertex of the tree. The task is to find top-k answer trees. Ranking is performed such that conditions C1 and C2 from Section 2 are satisfied.

For a three-term query which consists of terms w1, w2, w3, examples of possible results are given in Figure 2. A result can contain only one object which satisfies all turns from query (a). Such result is the best ranked. Result which consist of two objects (b) should be higher ranked then result which consists of three objects (c). It is possible that a resulting tree contains nodes which do not contain any term from the query (dark node in d).

The main problem in performing any kind of search on a graph created as previously explained is performance limited by the complexity of graph. Number of learning objects r induces $r^2 / 2$ edges, but significant number among them has small weights close to zero, which indicates there is no relevant similarity. Such edges should be eliminated from the graph. In other words, after graph creation, it is necessary to perform graph *sparsification*.

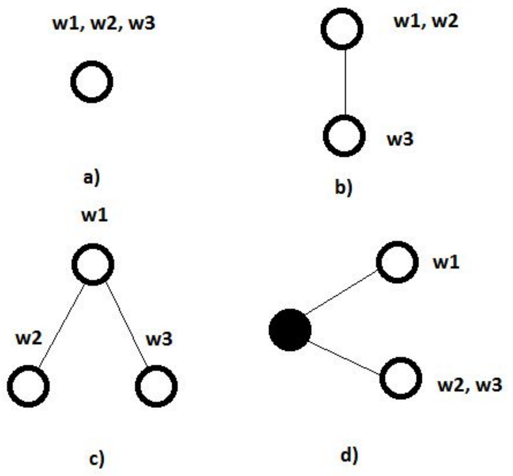


Figure 2. Examples of Steiner-trees search

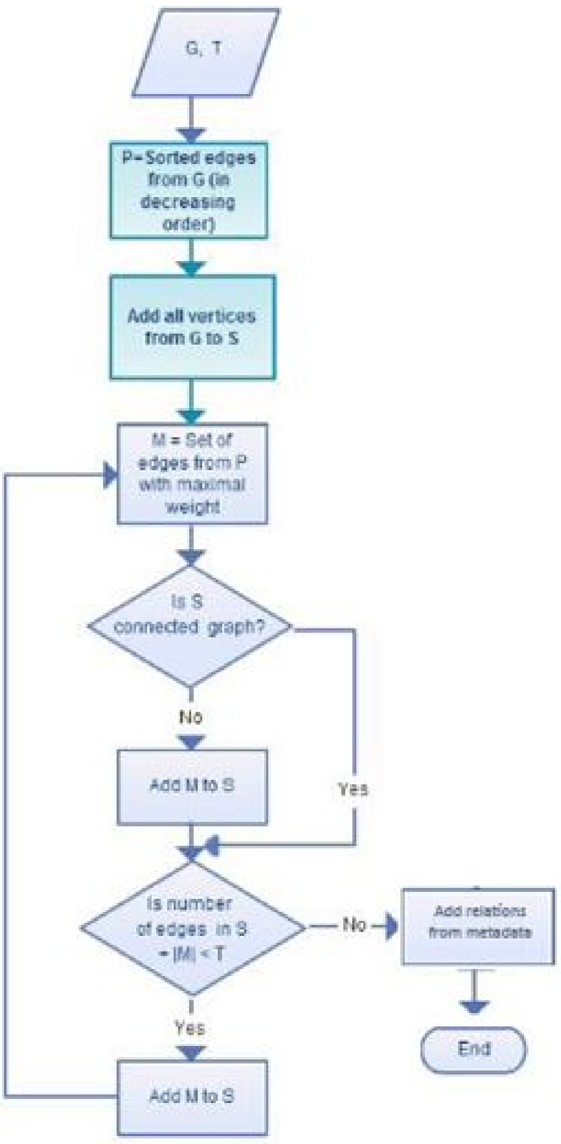
The following requirements should be satisfied:

- No vertex should be removed from the graph.
- Edges which represent low similarity should be removed from the graph.
- Edge removal should not violate graph connectivity.
- Targeted number of edges which should remain in the graph is specified by threshold value T. Graph obtained by sparsification process should have less than T edges, unless it violates connectivity constraint.
- If an edge with weight w is in S, then all edges with weights greater than or equal to w should be in S because there should be no priority among edges of equal weights.
- If two learning objects are in relationship specified by the appropriate metadata *relation*, it should be preserved in the graph regardless of similarity degree between these two learning objects.

For graph sparsification which satisfies all previous requirements, we propose the following algorithm:

```
Sparsify(G, T)
        sort in decreasing order all  edges
according to their weights    and    put them in
priority queue P;
        add all vertices from G to S;
        while(true)
                wMax <- weight of the edge
        with maximum weight from P;
                nMax <- number of edges from
        P with weight wMax;
                if (S is not connected)
                        add to S all edges
                        from P with weight
                        wMax;
                        remove  all  edges  from  P
                        with weight wMax;
                else
                        if (number of edges
                in S + nMax < T)
                                add to S all
                                edges from P
                                with weight
                                wMax;
                                remove all
                                edges from P
                                with weight
                                wMax;
                        else
                                break;

        foreach edge e in P
                if (exists relationship
                metadata between LO
        represented by vertices of            the
edge)
                        add e to S;
```


G, T
P=Sorted edges from G (in decreasing order)
Add all vertices from G to S
M = Set of edges from P with maximal weight
Is S connected graph?
No
Yes
Add M to S
Is number of edges in S + |M| < T
No
Add relations from metadata
End
Yes
Add M to S
Figure 3. Blok diagram of sparsify algorithm

The complexity of the proposed algorithm is dominated by the sorting procedure from the beginning of the algorithm. The rest of the procedure is linear in number of edges. Considering the fact that initially number of edges is quadratic function of number of learning objects, complexity of the algorithm is

$$O(|E|\log|E|) = O(r^2) .$$

5. SEARCH

Search algorithm

Search problem MCGST-k, as defined in the previous section, is an extension of the problem of finding the highest ranked tree known as minimum cost group Steiner tree problem (MCGST-1). The problem belongs to class of NP-complete problems which is proved by reducing it to minimum set cover problem [3]. Various approximate solutions of MCGST-1 problem are proposed. Some of them [3]-[7] are not easily extended to solving MCGST-k in an efficient manner because it would require finding all possible solutions [8]. Recently, several extendable solutions are proposed [8]-[10]. For our purpose, satisfying solution is algorithm DBPF-k proposed in [11] because it satisfies the following conditions:

- Although the solution of MCGST-k is approximate, the first returned result is optima, i.e., solution of MCGST-1 is optimal.
- Solution is obtained in polynomial time.
- Efficiency of DBPF-k algorithm depends on graph sparseness. In the previous section, we showed how to perform sparsification of a graph. Therefore, we can expect DBPF-k to perform well on graphs obtained from repositories of learning objects.
- Efficiency of DBPF-k algorithm depends on number of terms in query (|Q|<<log|V|). Typical usage scenario in searching for learning objects satisfies this condition. Therefore, we do not expect this condition to be an obstacle in efficiency.

Algorithm DBPF is developed to work on databases, but it is essentially search on underlying graph. Therefore, once the graph representation of a repository is created, its application to our problem is straightforward. Considering the fact that DBPF is min-cost type of algorithm, it becomes obvious why distance measure is used for graph weights instead of similarity measure. More details on design of DBPF, proof of its correctness and analysis of complexity can be find in [11].

Query language

In previous discussion, a search query Q was defined as set of terms. Submitting such query to the search engine implies finding all trees such that all terms from the query are contained in each of the returned answers. For example, query "math function" is a simple query and requires finding results which contain both words "math" *and* "function". This type of queries is common in search engines of web browsers and textual processors. In searching through a LO repository, often there is a certain level of uncertainty about terminology and relationships among concepts.

To avoid missing existent learning objects, it would be convenient to expand query language to allow resolving such disambiguates. For example, "math function" is possibly in some learning objects named as "mathematical function". In this simple example, it is possible for user to type two different queries and obtain wanted results. However, in longer and more complex queries, it would be more convenient and efficient for user to allow submitting a query which finds all learning objects (or, more precisely, trees of learning objects) which corresponds to demand "find math or mathematical function". For this purpose, we propose a simple extended query language based on rules of formal logic. The language is enriched by two operators:

- Operator ***and***, marked by reserved word %AND.
- Operator ***or***, marked by reserved word %OR.

In order to simplify use of these operators, the following convention is established:

- Both operators have the same precedence priority.
- Expressions are evaluated from left to right.
- If there is no operator between two terms, implicitly is assumed %AND operation. For example, "math function" is evaluated as "math %AND function".
- Associativity rule is preserved from formal logic.

By using these two operators, as well as parentheses, a user can form more complex queries. A query for finding mathematical or math functions can be specified as "(math %OR mathematical) %AND function". Query "(operations management) %OR (business operations control)" will treat "operations management" and "business operations control" as two separate queries and return union of their results.

Before explaining an algorithm for parsing queries enriched by previously described operators, first we will introduce some formal definitions.

Definition 1 (Term) – Term (denoted by lower case letter t) is a word which is used in a query.

Definition 2 (Simple Query) – Simple query (denoted by capital letter Q) is defined as a set of terms:

$$Q = \{t_1, t_2, ..., t_{|Q|}\}\,. \qquad ()$$

Definition 3 (Expression) – Expression (denoted by capital letter E) is defined as a set of simple queries:

$$E = \{Q_1, Q_2, ..., Q_{|E|}\}\,. \qquad ()$$

Query "(operations management) %OR (business operations control)" would be evaluated as an expression which consists of two simple queries:

$$Q_1 = \{operations, management\}\,,$$

$$Q_2 = \{bu\sin ess, operations, control\}\,,$$

$$E = \{Q_1, Q_2\}\,.$$

Evaluation of more complex queries requires defining two binary operations on expression, which corresponds to previously introduced operators (%AND, %OR) of query language:

- Operation $\wedge$ corresponds to operator %AND:

$$E_1 \wedge E_2 = \{Q_i \cup Q_j \mid Q_i \in E_1, Q_j \in E_2\}\,. \qquad ()$$

- Operation $\vee$ corresponds to operator %OR:

$$E_1 \vee E_2 = E_1 \cup E_2\,. \qquad ()$$

We will explain effect of these two operations on a simple example. Let say that we have four terms a, b, c, d. Query (a %AND b) %OR (c %AND d) can be evaluated as follows:

$$Q_1 = \{a, b\},\ Q_2 = \{c, d\},\ E = \{Q_1, Q_2\}.$$

Therefore, search algorithm has to be applied on two simple queries and the final result is union of them. Alternatively, evaluation can be performed in the following way:

$$Q_1 = \{a\},\ Q_2 = \{b\},\ Q_3 = \{c\},\ Q_4 = \{d\},$$

$$E_1 = \{Q_1\},\ E_2 = \{Q_2\},\ E_3 = \{Q_3\},\ E_4 = \{Q_4\},$$

$$E = (E_1 \wedge E_2) \vee (E_3 \wedge E_4).$$

In either case, the final result of search procedure is the same.

Extending query language by two operators (%AND, %OR) requires algorithm for parsing a query before search algorithm can be applied. Complex expressions like (a %OR b) %AND ((c %AND d) %OR e), for example, cannot be evaluated directly. We propose the following algorithm for performing this task:

```
initialize S as empty stack of expressions;
initialize empty set of search results R;

foreach token w of query
        switch(w):
        case “(”,“%AND”,“%OR”: push w to S;
        case “)”:
                E<-evaluateTopExpression(S);
                push E to S;
        default:
                if(previous token is term)
                        push “%AND” to S;
                Q = {w};
                E = {Q};
                push E to S;
        end switch;

E<-evaluateTopExpression(S);
foreach simple query Q from E
        result = DBPF-k(Q);
        add result to R;
```

In the previously described algorithm, help function evaluateTopExpression is used to evaluate value of expression from the top of the stack. It is realized in the following way:

6. CONCLUSION

In this paper, we proposed an architectural solution for enhanced search through repositories of learning objects. Traditional textual search encountered in web browsers and text processing applications does not satisfy needs for data retrieval from learning object repositories mainly because it ignores existent explicit and implicit relationships among objects. In addition, a complex query with more words can result in an empty result set. As a solution to these problems, we proposed search based on finding top-k min-cost Steiner trees.

In particular, we developed an algorithm for sparse weighted graph representation of a LO repository, which is suitable for application of algorithm for finding Steiner trees proposed in [11]. To further improve searching capabilities, we proposed extension of query language based on formal logic and designed an algorithm for parsing it. In order to keep simplicity, the extension is reduced to only two additional operators, which corresponds to logical AND and logical OR.

ACKNOWLEDGMENT

This work was supported by Ministry of Education, Science and Technology (Project III44006).

```
valuateTopExpression(S)
{
initialize SH as empty stack;
while (S not empty)
        wh<-pop from S;
        if(wh = “(”)
                break;
        push wh to SH;

while (true)
        first<-pop from SH;
        if (SH is empty) return first;
        operator<-pop from SH;
        second<-pop from SH;
        switch(operator)
        case “%AND”:
                result = first ^ second;
        case “%OR”:
                result = first v second;
        end switch;
        push result to SH;
}
```

LITERATURE

[1] LOM (2002). *Final Draft Standard for Learning Object Metadata IEEE 14854.12.1-2002.* On-line available: http://ltsc.ieee.org/.

[2] IEEE Learning Technology Standards Committee [Online] Available: http://ltsc.ieee.org/

[3] G. Reich and P. Widmayer. *Beyond steiner's problem: A vlsi oriented generalization.* In Proc. of WG'89, 1989.

[4] E. Ihler. *Bounds on the quality of approximate solutions to the group steiner problem.* In Proc. of WG'90, 1990.

[5] C. D. Bateman, C. S. Helvig, G. Robins, and A. Zelikovsky. *Probably good routing tree construction with multi-port terminals.* In Proc. of ISPD'97, 1997.

C. Helvig, B. Robins, and A. Zelikovsky. *Improved approximation bounds for the group Steiner problem.* Networks,37(1),2001.

[6] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. *Approximation algorithms for directed Steiner problems.* Journal of Algorithms, 33(1),1999.

[7] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal. *Query relaxation by structure and semantics for*

retrieval of logical web documents. IEEE Trans. Knowl.Data.Eng.,14(4),2002.

[8] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. *Keyword searching and browsing in databases using banks.* In Proc. of ICDE'02, 2002.

[9] K. Varun, P. Shashank, C. Soumen, S. Sudarshan, D. Rushi, and K. Hrishikesh. *Bidirectional expansion for keyword search on graph databases.* In Proc. of VLDB'05,2005.

[10] B. Ding, J.X. Yu, S. Wang, L. Qing, X. Zhang, and X. Lin. *Finding top-k min-cost connected trees in databases.*ICDE,2007