ObjectRank: Authority-Based Keyword Search in Databases

(extended version)

Andrey Balmin
IBM Almaden Research
San Jose, CA 95120
abalmin@us.ibm.com

Vagelis Hristidis
School of Computer Science
Florida International University
Miami, FL 33199
vagelis@cs.fiu.edu

Yannis Papakonstantinou
Computer Science
UC, San Diego
La Jolla, CA 92093
yannis@cs.ucsd.edu

Abstract

The ObjectRank system applies authority-based ranking to keyword search in databases modeled as labeled graphs. Conceptually, authority originates at the nodes (objects) containing the keywords and flows to objects according to their semantic connections. Each node is ranked according to its authority with respect to the particular keywords. One can adjust the weight of global importance, the weight of each keyword of the query, the importance of a result actually containing the keywords versus being referenced by nodes containing them, and the volume of authority flow via each type of semantic connection. Novel performance challenges and opportunities are addressed. First, schemas impose constraints on the graph, which are exploited for performance purposes. Second, in order to address the issue of authority ranking with respect to the given keywords (as opposed to Google's global PageRank) we precompute single keyword ObjectRanks and combine them during run time. We conducted user surveys and a set of performance experiments on multiple real and synthetic datasets, to assess the semantic meaningfulness and performance of ObjectRank.

1 Introduction

PageRank [8] is an excellent tool to rank the global importance of the pages of the Web, proven by the success of Google [1]. However, Google uses PageRank as a tool to measure the global importance of the pages, independently of a keyword query. (Google also uses other IR techniques to estimate the relevance of a page to a keyword query, which is then combined with the PageRank value to calculate the final score of a page.) More recent works [18, 26] apply PageRank to estimate the relevance of pages to a keyword query. We appropriately extend and modify PageRank to perform keyword search in databases for which there is a natural flow of authority between their objects (e.g.: bibliographic or complaints databases as we explain below).

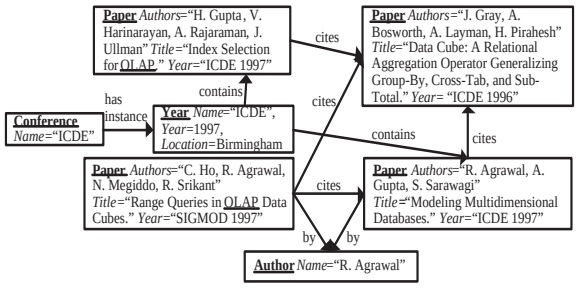

Figure 1: A subset of the DBLP graph

Consider the example of Figure 1, which illustrates a small subset of the DBLP database in the form of a labeled graph (author, conference and year nodes except for "R. Agrawal", "ICDE" and "ICDE 1997" respectively are omitted to simplify the figure). Schema graphs, such as the one of Figure 3, describe the structure of database graphs. Given a keyword query, e.g. the single keyword query "OLAP", ObjectRank sorts the database objects by their importance with respect to the user-provided keywords. Figure 2 illustrates the top-10 "OLAP" papers in the DBLP subset (produced by our online demo at `http://www.db.ucsd.edu/ObjectRank`)[2] currently used by the ObjectRank prototype. Notice that many entries (the "Data Cube" and the "Modeling Multidimensional Databases" papers in Figure 1) of the top-10 list do not contain the keyword "OLAP" ("OLAP" is not even contained in their abstracts) but they clearly constitute important papers in the OLAP area, since they may be referenced by other papers of the OLAP area or may have

[1] http://www.Google.com

[2] The DBLP subset currently used by ObjectRank was copied from our XKeyword [20] database (demo available at `http://www.db.ucsd.edu/XKeyword`) and consists of the available publications in 12 major database conferences, including SIGMOD, VLDB, PODS, ICDE, ICDT and EDBT, up to year 2001.

been written by authors who have written other important "OLAP" papers.

Conceptually, the ranking is produced in the following way: Myriads of random surfers are initially found at the objects containing the keyword "OLAP" and then they traverse the database graph. In particular, at any time step a random surfer is found at a node and either (i) makes a move to an adjacent node by traversing an edge, or (ii) jumps randomly to an "OLAP" node without following any of the links. The probability that a particular traversal happens depends on multiple factors, including the type of the edge (in contrast to the Web link-based search systems [8, 18, 26]). These factors are depicted in an authority transfer schema graph. Figure 4 illustrates the authority transfer schema graph that corresponds to the setting that produced the results of Figure 2. Assuming that the probability that the surfer moves back to an "OLAP" node is 15% (damping factor [8]), the collective probability to move to a referenced paper is up to $85\% \times 70\%$ (70% is the authority transfer rate of the citation edge as we explain below), the collective probability to move to an author of the paper is up to $85\% \times 20\%$, the probability to move from the paper to the forum where the paper appeared is up to $85\% \times 10\%$, and so on. As is the case with the PageRank algorithm as well, as time goes on, the expected percentage of surfers at each node v converges (Section 2) to a limit $r(v)$. Intuitively, this limit is the ObjectRank of the node.

An alternative way to conceive the intuition behind ObjectRank is to consider that authority/importance flows in the database graph in the same fashion that [24] defined authority-based search in arbitrary graphs. Initially the "OLAP" authority is found at the objects that contain the keyword "OLAP". Then authority/importance flows, following the rules in the authority transfer schema graph, until an equilibrium is established that specifies that a paper is authoritative if it is referenced by authoritative papers, is written by authority authors and appears in authority conferences. Vice versa, authors and conferences obtain their authority from their papers. Notice that the amount of authority flow from, say, paper to cited paper or from paper to author or from author to paper, is arbitrarily set by a domain expert and reflects the semantics of the domain. For example, common sense says that in the bibliography domain a paper obtains very little authority (or even none) by referring to authoritative papers. On the contrary it obtains a lot of authority by being referred by authoritative papers. Our DBLP demo offers to the user more than one authority flow settings, in order to accommodate multiple user profiles/requirements. We believe the ability to customize authority flow schemes is central to ObjectRank, since we should not assume that "one size fits all" when it comes to opinions about authority flow. For example, there is one setting for users that primarily care for papers with high global importance and another for users that primarily care for papers that are directly or indirectly heavily referenced by papers that have the keywords. We expect that multiple settings make sense in all non-trivial ObjectRank applications.

Keyword search in databases has some unique characteristics, which make the straightforward application of the random walk model as described in previous work [8, 18, 26] inadequate. First, every database has different semantics, which we can use to improve the quality of the keyword search. In particular, unlike the Web, where all edges are hyperlinks[3], the database schema exhibits the types of edges, and the attributes of the nodes. Using the schema we specify the ways in which authority flows across the nodes of the database graph. For example, the results of Figure 2 were obtained by annotating the schema graph of Figure 3 with the authority flow information that appears in Figure 4.

Furthermore, previous work [8, 18, 26] assumes that, when calculating the global importance (in our framework we make a clear distinction between the global importance of a node and its relevance to a keyword query), the random surfer has the same probability to start from any page p of the base set (we call this probability *base ObjectRank* of p). However, this is not true for every database. For example, consider a product complaints database (Figure 21). In this case, we represent the business value of a customer by assigning to his/her node a base ObjectRank proportional to his/her total sales amount.

Another novel property of ObjectRank is adjustability, which allows for the tuning of the system according to the domain- and/or user-specific requirements. For example, for a bibliographic database, a new graduate student desires a search system that returns the best reading list around the specified keywords, whereas a senior researcher looks for papers closely related to the keywords, even if they are not of a high quality. These preference scenarios are made possible by adjusting the weight of the global importance versus the relevance to the keyword query. Changing the damping factor d offers another calibration opportunity. In particular, larger values of d favor nodes pointed by high-authority nodes, while smaller values of d favor nodes containing the actual keywords (that is, nodes in the base set). The handling of queries with multiple keywords offers more flexibility to the system as we describe in Section 3. For example, we may want to assign a higher weight to the relevance of a node to an infrequent keyword.

On the performance level, calculating the ObjectRank values in runtime is a computationally intensive operation, especially given the fact that multiple users query the system. This is resolved by precomputing an inverted index where for each keyword we have a sorted list of the nodes with non-trivial ObjectRank for this keyword. During runtime we employ the *Threshold Algorithm* [13] to efficiently combine the lists. However, our approach induces the cost of precomputing and storing the inverted index. Regarding the space requirements, notice that the number of keywords

[3]Previous works [26, 9, 7] assign weights on the edges of the data graph according to the relevance of the incident nodes' text to the keywords. In contrast, we assign authority transfer rates on the schema graph, which captures the semantics of the database, since the relevance factor is reflected in the selection of the base set.

41.34 Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. Jim Gray, ICDE 1996
36.62 Index Selection for OLAP. Himanshu Gupta, ICDE 1997
35.11 Range Queries in OLAP Data Cubes. Ching-Tien Ho, SIGMOD 1997
31.03 Discovery-Driven Exploration of OLAP Data Cubes. Sunita Sarawagi, EDBT 1998
30.7 OLAP and Statistical Databases: Similarities and Differences. Arie Shoshani, PODS 1997
30.23 Implementing Data Cubes Efficiently. Venky Harinarayan, SIGMOD 1996
29.42 Relative Prefix Sums: An Efficient Approach for Querying Dynamic OLAP Data Cubes. Steven Geffner, ICDE 1999
28.49 Modeling Multidimensional Databases. Rakesh Agrawal, ICDE 1997
26.96 Summarizability in OLAP and Statistical Data Bases. Hans-J. Lenz, SSDBM 1997
26.75 Data Warehousing and OLAP for Decision Support (Tutorial). Surajit Chaudhuri, SIGMOD 1997

Figure 2: Top 10 papers on "OLAP" returned by ObjectRank

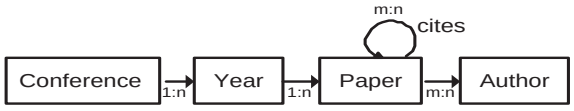

Figure 3: The DBLP schema graph.

of a database is typically limited, and less than the number of users in a personalized search system [22]. Furthermore, we do not store nodes with ObjectRank below a threshold value (chosen by the system administrator), which offers a space versus precision tradeoff. In Section 7 we show that the index size is small relative to the database size for two bibliographic databases.

Regarding the index computation, we present and experimentally evaluate two classes of optimizations. First, we exploit the structural properties of the database graph. For example, if we know that the objects of a subgraph of the schema form a Directed Acyclic Graph (DAG), then given a topological sort of the DAG, there is an efficient straightforward one-pass ObjectRank evaluation. We extend the DAG case by providing an algorithm that exploits the efficient evaluation for DAGs in the case where a graph is "almost" a DAG in the sense that it contains a large DAG subgraph. In particular, given a graph G with n nodes, which is reduced to a DAG by removing a small subset of m nodes, we present an algorithm which reduces the authority calculation into a system of m equations - as opposed to the usual system of n equations. Furthermore, we present optimization techniques when the data graph has a small vertex cover, or if it can be split into a set of subgraphs and the connections between these subgraphs form a DAG.

Second, notice that the naive approach would be to calculate each keyword-specific ObjectRank separately. We have found that it is substantially more efficient to first calculate the global ObjectRank, and use these scores as initial values for the keyword-specific computations. This accelerates convergence, since in general, objects with high global ObjectRank, also have high keyword-specific ObjectRanks. Furthermore, we show how storing a prefix of the inverted lists allows the faster calculation of the ObjectRanks of all nodes.

The semantic and performance contributions of this paper are evaluated using two user surveys and a detailed experimental evaluation respectively. We have implemented a web interface, available at `http://www.db.ucsd.edu/ObjectRank`, to query a subset of the DBLP database using the ObjectRank technique.

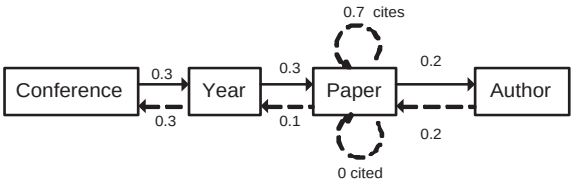

Figure 4: The DBLP authority transfer schema graph.

The essential formal background on PageRank and authority search is presented in Section 2. Section 3 presents the semantics of ObjectRank and Section 4 describes the system's architecture. The algorithms used to calculate ObjectRank are presented in Section 5 and are experimentally evaluated in Section 7. We present the results of two user surveys in Section 6. Furthermore, related work is discussed in Section 9. Finally, conclusions and future work are discussed in Section 11.

2 Background

We describe next the essentials of PageRank and authority-based search, and the random surfer intuition. Let (V, E) be a graph, with a set of nodes $V = \{v_1, \ldots, v_n\}$ and a set of edges E. A surfer starts from a random node (web page) v_i of V and at each step, he/she follows a hyperlink with probability d or gets bored and jumps to a random node with probability $1 - d$. The PageRank value of v_i is the probability $r(v_i)$ that at a given point in time, the surfer is at v_i. If we denote by $\mathbf{r}$ the vector $[r(v_1), \ldots, r_(v_i), \ldots, r(v_n)]^T$ then we have

$$\mathbf{r} = d\mathbf{A}\mathbf{r} + \frac{(1-d)}{|V|}\mathbf{e} \quad (1)$$

where $\mathbf{A}$ is a $n \times n$ matrix with $A_{ij} = \frac{1}{OutDeg(v_j)}$ if there is an edge $v_j \rightarrow v_i$ in E and 0 otherwise, where $OutDeg(v_j)$ is the outgoing degree of node v_j. Also, $\mathbf{e} = [1, \ldots, 1]^T$.

The above PageRank equation is typically precomputed before the queries arrive and provides a global, keyword-independent ranking of the pages. Instead of using the whole set of nodes V as the *base set*, i.e., the set of nodes where the surfer jumps when bored, one can use an arbitrary subset S of nodes, hence increasing the authority associated with the nodes of S and the ones most closely associated with them. In particular, we define a *base vector* $\mathbf{s} = [s_0, \ldots, s_i, \ldots, s_n]^T$ where s_i is 1 if $v_i \in S$ and 0 otherwise. The PageRank equation is then

$$\mathbf{r} = d\mathbf{A}\mathbf{r} + \frac{(1-d)}{|S|}\mathbf{s} \quad (2)$$

Regardless of whether one uses Equation 1 or Equation 2 the PageRank algorithm solves this fixpoint using a simple iterative method, where the values of the (k+1)-th execution are calculated as follows:

$$\mathbf{r}^{(k+1)} = d\mathbf{A}\mathbf{r}^{(k)} + \frac{(1-d)}{|S|}\mathbf{s} \quad (3)$$

The algorithm terminates when $\mathbf{r}$ converges, which is guaranteed to happen under very common conditions [25]. In particular, $\mathbf{A}$ needs to be irreducible (i.e., (V, E) be strongly connected) and aperiodic. The former is true due

Parameter property	*Parameters*
`Application - specific`	`authority transfer rates, global ObjectRank calculation, damping factor`
`Combination of scores`	`normalization scheme, global ObjectRank weight, AND or OR semantics`
`Performance`	`epsilon, threshold`

Table 1: Parameters of ObjectRank

to the damping factor d, while the latter happens in practice.

The notion of the base set S was suggested in [8] as a way to do personalized rankings, by setting S to be the set of bookmarks of a user. In [18] it was used to perform topic-specific PageRank on the Web. We take it one step further and use the base set to estimate the relevance of a node to a keyword query. In particular, the base set consists of the nodes that contain the keyword as explained next.

3 ObjectRank Semantics

In this section we formally define the framework of this work, and show how ObjectRank ranks the nodes of a database with respect to a given keyword query, given a set of calibrating (adjusting) parameters (Table 1). In particular, Section 3.1 describes how the database and the authority transfer graph are modeled. Section 3.2 shows how the keyword-specific and the global ObjectRanks are calculated and combined to produce the final score of a node. Finally, Section 3.3 presents and addresses the challenges for multiple-keyword queries.

3.1 Database Graph, Schema, and Authority Transfer Graph

We view a database as a labeled graph, which is a model that easily captures both relational and XML databases. The *data graph* $D(V_D, E_D)$ is a labeled directed graph where every node v has a label $\lambda(v)$ and a set of keywords. For example, the node "ICDE 1997" of Figure 1 has label "Year" and the set of keywords {``ICDE'', ``1997'', ``Birmingham''}. Each node represents an *object* of the database and may have a sub-structure. Without loss of generality, ObjectRank assumes that each node has a tuple of attribute name/attribute value pairs. For example, the "Year" nodes of Figure 1 have `name`, `year` and `location` attributes. Notice that the keywords appearing in the attribute values comprise the set of keywords associated with the node. One may assume richer semantics by including the metadata of a node in the set of keywords. For example, the metadata "Forum", "Year", "Location" could be included in the keywords of a node. The specifics of modeling the data of a node are orthogonal to ObjectRank and will be neglected in the rest of the discussion.

Each edge e from u to v is labeled with its *role* $\lambda(e)$ (we overload λ) and represents a relationship between u and v. For example, every "paper" to "paper" edge of Figure 1 has the label "cites". When the role is evident and uniquely defined from the labels of u and v, we omit the edge label. For simplicity we will assume that there are no parallel edges and we will often denote an edge e from u to v as "$u \to v$".

The *schema graph* $G(V_G, E_G)$ (Figure 3) is a directed graph that describes the structure of D. Every node has an associated label. Each edge is labeled with a role, which may be omitted, as discussed above for data graph edge labels. We say that a data graph $D(V_D, E_D)$ *conforms* to a schema graph $G(V_G, E_G)$ if there is a unique assignment μ such that:

1. for every node $v \in V_D$ there is a node $\mu(v) \in V_G$ such that $\lambda(v) = \lambda(\mu(v))$;
2. for every edge $e \in E_D$ from node u to node v there is an edge $\mu(e) \in E_G$ that goes from $\mu(u)$ to $\mu(v)$ and $\lambda(e) = \lambda(\mu(e))$.

Authority Transfer Schema Graph. From the schema graph $G(V_G, E_G)$, we create the *authority transfer schema graph* $G^A(V_G, E^A)$ to reflect the authority flow through the edges of the graph. This may be either a trial and error process, until we are satisfied with the quality of the results, or a domain expert's task. In particular, for each edge $e_G = (u \to v)$ of E_G, two *authority transfer edges*, $e_G^f = (u \to v)$ and $e_G^b = (v \to u)$ are created. The two edges carry the label of the schema graph edge and, in addition, each one is annotated with a (potentially different) *authority transfer rate* - $\alpha(e_G^f)$ and $\alpha(e_G^b)$ correspondingly. We say that a data graph conforms to an authority transfer schema graph if it conforms to the corresponding schema graph. (Notice that the authority transfer schema graph has all the information of the original schema graph.)

Figure 4 shows the authority transfer schema graph that corresponds to the schema graph of Figure 3 (the edge labels are omitted). The motivation for defining two edges for each edge of the schema graph is that authority potentially flows in both directions and not only in the direction that appears in the schema. For example, a paper passes its authority to its authors and vice versa. Notice however, that the authority flow in each direction (defined by the authority transfer rate) may not be the same. For example, a paper that is cited by important papers is clearly important but citing important papers does not make a paper important.

Notice that the sum of authority transfer rates of the outgoing edges of a schema node u may be less than 1[4], if the administrator believes that the edges starting from u do not transfer much authority. For example, in Figure 4, conferences only transfer 30% of their authority.

Authority Transfer Data Graph. Given a data graph $D(V_D, E_D)$ that conforms to an authority transfer schema graph $G^A(V_G, E^A)$, ObjectRank derives an *authority transfer data graph* $D^A(V_D, E_D^A)$ (Figure 5) as follows. For every edge $e = (u \to v) \in E_D$ the authority transfer data graph has two edges $e^f = (u \to v)$ and $e^b = (v \to u)$.

[4] In terms of the random walk model, this would be equivalent to the disappearance of a surfer.

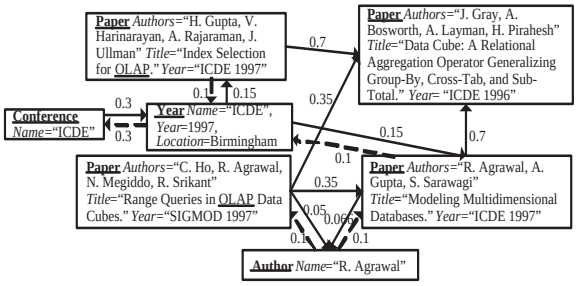

Figure 5: Authority transfer data graph

The edges e^f and e^b are annotated with authority transfer rates $\alpha(e^f)$ and $\alpha(e^b)$. Assuming that e^f is of type e_G^f, then

$$\alpha(e^f) = \begin{cases} \frac{\alpha(e_G^f)}{\mathit{OutDeg}(u, e_G^f)}, \ if\ \mathit{OutDeg}(u, e_G^f) > 0 \\ 0, \ if\ \mathit{OutDeg}(u, e_G^f) = 0 \end{cases} \quad (4)$$

where $\mathit{OutDeg}(u, e_G^f)$ is the number of outgoing edges from u, of type e_G^f. The authority transfer rate $\alpha(e^b)$ is defined similarly. Figure 5 illustrates the authority transfer data graph that corresponds to the data graph of Figure 1 and the authority schema transfer graph of Figure 4. Notice that the sum of authority transfer rates of the outgoing edges of a node u of type $\mu(u)$ may be less than the sum of authority transfer rates of the outgoing edges of $\mu(u)$ in the authority transfer schema graph, if u does not have all types of outgoing edges.

3.2 Importance vs. Relevance.

The score of a node v with respect to a keyword query w is a combination of the global ObjectRank $r^G(v)$ of v and the keyword-specific ObjectRank $r^w(v)$. We propose the following combining function, although other functions may be used as well:

$$r^{w,G}(v) = r^w(v) \cdot (r^G(v))^g \quad (5)$$

where g is the *global ObjectRank weight*, which determines how important the global ObjectRank is. Notice that g may be accessible to the users or fixed by the administrator. The calculations of the keyword-specific and the global ObjectRank are performed as follows (we assume single-keyword queries at this point).

Keyword-specific ObjectRank. Given a single keyword query w, ObjectRank finds the *keyword base set* $S(w)$ (from now on referred to simply as base set when the keyword is implied) of objects that contain the keyword w and assigns an ObjectRank $r^w(v_i)$ to every node $v_i \in V_D$ by resolving the equation

$$\mathbf{r}^w = d\mathbf{A}\mathbf{r}^w + \frac{(1-d)}{|S(w)|}\mathbf{s} \quad (6)$$

where $A_{ij} = \alpha(e)$ if there is an edge $e = (v_j \rightarrow v_i)$ in E_D^A and 0 otherwise, d controls the base set importance, and $\mathbf{s} = [s_1, \ldots, s_n]^T$ is the base set vector for $S(w)$, i.e., $s_i = 1$ if $v_i \in S(w)$ and $s_i = 0$ otherwise.

The damping factor d determines the portion of ObjectRank that an object transfers to its neighbors as opposed to keeping to itself. It was first introduced in the original PageRank paper [8], where it was used to ensure convergence in the case of PageRank sinks. However, in addition to that, in our work it is a calibrating factor, since by decreasing d, we favor objects that actually contain the keywords (i.e., are in base set)as opposed to objects that acquire ObjectRank through the incoming edges. The value for d used by PageRank [8] is 0.85, which we also adopt when we want to balance the importance of containing the actual keywords as opposed to being pointed by nodes containing the keywords.

Global ObjectRank. The definition of global ObjectRank is different for different applications or even users of the same application. In this work, we focus on cases where the global ObjectRank is calculated applying the random surfer model, and including all nodes in the base set. The same calibrating parameters are available, as in the keyword-specific ObjectRank. Notice that this way of calculating the global ObjectRank, which is similar to the PageRank approach [8], assumes that all nodes (pages in PageRank) initially have the same value. However, there are many applications where this is not true, as we discuss in Section 11.

3.3 Multiple-Keywords Queries.

We define the semantics of a multiple-keyword query "$w_1, \ldots, w_m$" by naturally extending the random walk model. We consider m independent random surfers, where the ith surfer starts from the keyword base set $S(w_i)$. For AND semantics, the ObjectRank of an object v with respect to the m-keywords query is the probability that, at a given point in time, the m random surfers are simultaneously at v. Hence the ObjectRank $r_{AND}^{w_1,\ldots,w_m}(v)$ of the node v with respect to the m keywords is

$$r_{AND}^{w_1,\ldots,w_m}(v) = \prod_{i=1,\ldots,m} r^{w_i}(v) \quad (7)$$

where $r^{w_i}(v)$ is the ObjectRank with respect to the keyword w_i.

For OR semantics, the ObjectRank of v is the probability that, at a given point in time, *at least one* of the m random surfers will reach v. Hence, for two keywords w_1 and w_2 it is

$$r_{OR}^{w_1,w_2}(v) = r^{w_1}(v) + r^{w_2}(v) - r^{w_1}(v)r^{w_2}(v) \quad (8)$$

and for more than two, it is defined accordingly. Notice that [18] also sums the topic-sensitive PageRanks to calculate the PageRank of a page.

3.4 Weigh keywords by frequency

A drawback of the *combining function* of Equation 7 is that it favors the more popular keywords in the query. The reason is that the distribution of ObjectRank values is more skewed when the size $|S(w)|$ of the base set $S(w)$ increases, because the top objects tend to receive more references. For example, consider two results for the query

(a)

47.31	11.44	An XML Indexing Structure with Relative Region Coordinate. Dao Dinh Kha, ICDE 2001
41.02	3.08	DataGuides: Enabling Query ... Optimization in Semistructured... Roy Goldman, VLDB 1997
7.44	28.43	Access Path Selection in a RDBMS. Patricia G. Selinger, SIGMOD 1979
31.44	3.24	Querying Object-Oriented Databases. Michael Kifer, SIGMOD 1992
26.73	3.09	A Query ... Optimization Techniques for Unstructured Data. Peter Buneman, SIGMOD 1996

(b)

47.31	11.44	An XML Indexing Structure with Relative Region Coordinate. Dao Dinh Kha, ICDE 2001
7.44	28.43	Access Path Selection in a RDBMS. Patricia G. Selinger, SIGMOD 1979
2.04	102.1	R-Trees: A Dynamic Index Structure for Spatial Searching. Antonin Guttman, SIGMOD 1984
1.73	112.7	The K-D-B-Tree: A Search Structure For Large ... Indexes. John T. Robinson, SIGMOD 1981
41.02	3.08	DataGuides: Enabling Query ... Optimization in Semistructured... Roy Goldman, VLDB 1997

Figure 6: Top 5 papers on "XML Index", with and without emphasis on "XML"

"XML AND Index" shown in Figure 6. Result (b) corresponds to the model described above. It noticeably favors the "Index" keyword over the "XML". The first paper is the only one in the database that contains both keywords in the title. However, the next three results are all classic works on indexing and do not apply directly to XML. Intuitively, "XML" as a more specific keyword is more important to the user. Indeed, the result of Figure 6 (a) was overwhelmingly preferred over the result of Figure 6 (b) by participants of our relevance feedback survey (Section 6). The latter result contains important works on indexing in semistructured, unstructured, and object-oriented databases, which are more relevant to indexing of XML data. This result is obtained by using the modified formula:

$$r^{w_1,\ldots,w_m}(v) = \prod_{i=1,\ldots,m} (r^{w_i}(v))^{g(w_i)} \qquad (9)$$

where $g(w_i)$ is a *normalizing exponent*, set to $g(w_i) = 1/log(|S(w_i)|)$. Using the normalizing exponents g("XML") and g("Index") in the above example is equivalent to running in parallel g("XML") and g("Index") random walks for the "XML" and the "Index" keywords respectively.

3.5 Compare to single base set approach

One can imagine alternative semantics to calculate the ObjectRank for multiple keywords, other than combining the single-keyword ObjectRanks. In particular, consider combining all objects with at least one of the keywords into a single base set. Then a single execution of the ObjectRank algorithm is used to determine the scores of the objects. Incidentally, these semantics were used in the HITS system [24]. We show that such "single base set" semantics can be achieved by combining single-keyword ObjectRank values applying appropriate exponents. Furthermore, we explain how our semantics avoid certain problems of "single base set" semantics.

In order to compare to the "single base set" approach for AND semantics (Equation 7), we consider two scenarios and assume without loss of generality that there are two keywords. First, assume that we only put in the base set S objects that contain both keywords. These objects will be in both keyword-specific base sets as well, so these objects and objects pointed by them will receive a top rank in both approaches. Second, if S contains objects containing any of the two keywords, we may end up ranking highest an object that is only pointed by objects containing one keyword. This cannot happen with the keyword-specific

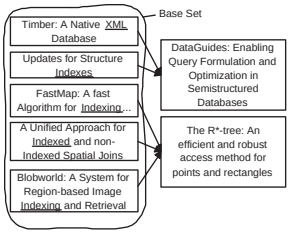

Figure 7: Example where "HITS" approach fails in AND semantics.

base sets approach. For example, in Figure 7, the "single base set" approach would rank the R^* paper higher than the DataGuides paper for the query "XML AND Index", even though the R^* paper is irrelevant to XML.

For OR semantics (Equation 8), the base set S in the"single base set" approach is the union of the keyword-specific base sets. We compare to an improved version of the "single base set" approach, where objects in base set are weighted according to the keywords they contain, such that infrequent keywords are assigned higher weight. In particular, if an object contains both keywords, for a two keyword query, it is assigned a base ObjectRank of $(1-d)\cdot(\frac{1}{|S(w_1)|}+\frac{1}{|S(w_2)|})$. Then, using the Linearity Theorem in [22], we can prove that the ObjectRanks calculated by both approaches are the same.

4 Architecture

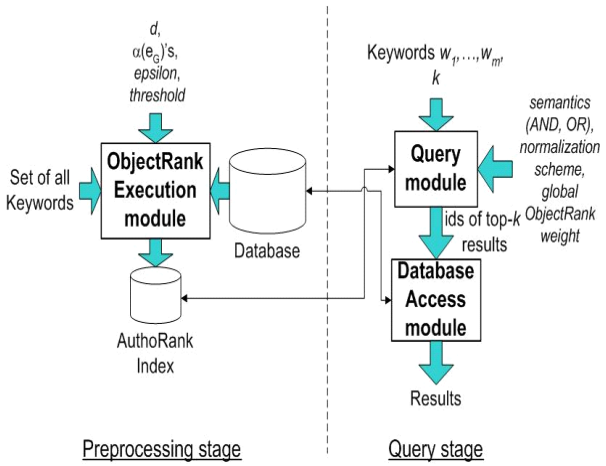

Figure 8: System Architecture.

Figure 8 shows the architecture of the ObjectRank system, which is divided into two stages. The preprocessing stage consists of the *ObjectRank Execution module*, which inputs the database to be indexed, the set of all keywords that will be indexed, and a set of parameters (the rest of the adjusting parameters are input during the query stage). In particular these parameters are: (i) the damping factor d, (ii) the authority transfer rates $\alpha(e_G)$'s of the authority transfer schema graph G^A, (iii) the convergence constant $epsilon$ which determines when the ObjectRank algorithm converges, and (iv) the $threshold$ value which determines

the minimum ObjectRank that an object must have to be stored in the ObjectRank Index.

The ObjectRank Execution module creates the *ObjectRank Index*, which is an inverted index, indexed by the keywords. For each keyword w, it stores a list of $\langle id(u), r^w(u)\rangle$ pairs for each object u that has $r^w(u) \geq threshold$. The pairs are sorted by descending $r^w(u)$ to facilitate an efficient querying method as we describe below. The ObjectRank Index has been implemented as an index-based table, where the lists are stored in a CLOB attribute. A hash-index is built on top of each list to allow for random access, which is required by the Query module.

The *Query module* inputs a set of sorted $\langle id(u), r^w(u)\rangle$ pairs lists $L_1, \ldots, L_m$ and a set of adjusting parameters, and outputs the top-k objects according to the combining function (Equation 7 or 8). In particular, these parameters are: (i) the semantics to be used (AND or OR), (ii) the normalization scheme, i.e., the exponents to use, and (iii) the global ObjectRank weight. The naive approach would be to make one pass of all lists to calculate the final ObjectRank values for each object and then sort this list by final ObjectRank. Instead, we use the *Threshold Algorithm* [13] which is guaranteed to read the minimum prefix of each list. Notice that the Threshold Algorithm is applicable since both combining functions (Equations 7 and 8) are monotone, and random access is possible on the stored lists.

Finally, the *Database Access module* inputs the result ids and queries the database to get the suitable information to present the objects to the user. This information is stored into an id-indexed table, that contains a CLOB attribute value for each object id. For example, a paper object CLOB would contain the paper title, the authors' names, and the conference name and year.

5 ObjectRank Index creation

This section presents algorithms to create the ObjectRank index. Section 5.1 presents an algorithm for the case of arbitrary authority transfer data graphs D^A. Sections 5.2 and 5.3 show how we can do better when D^A is a directed acyclic graph (DAG) and "almost" a DAG respectively (the latter property is explained in Section 5.3). Sections 5.4, 5.5 present optimizations when the authority transfer graph has a small vertex cover, or is a DAG of subgraphs. Finally, Section 5.6 presents optimization opportunities based on manipulating the initial values of the iterative algorithm.

5.1 General algorithm

Figure 9 shows the algorithm that creates the ObjectRank Index. The algorithm accesses the authority transfer data graph D^A many times, which may lead to a too long execution time if D^A is very large. Notice that this is usually not a problem, since D^A only stores object ids and a set of edges which is small enough to fit into main memory for most databases. Notice that lines 2-4 correspond to the original PageRank calculation [8] modulo the authority transfer rates information.

```
CreateIndex(keywordsList, epsilon, threshold, α(.), d){
01. For each keyword w in keywordsList do {
02.   While not converged do
03.   /*i.e., ∃v, |r^(k+1)(v) − r^(k)(v)| > epsilon*/
04.     MakeOnePass(w,α(.), d);
05.   StoreObjectRanks();
06. }
}
MakeOnePass(w,α(.), d) {
07. Evaluate Equation 6 using the r from
      the previous iteration on the right side;
}
StoreObjectRanks() {
08. Sort the ⟨id(i), r(v_i)⟩ pairs list by r(v_i) and
    store it in inverted index, after removing pairs with
    r(v_i) < threshold;
}
```

Figure 9: Algorithm to create ObjectRank Index

5.2 DAG algorithm

There are many applications where the authority transfer data graph is a DAG. For example a database of papers and their citations (ignoring author and conference objects), where each paper only cites previously published papers, is a DAG. Figure 10 shows an improved algorithm, which makes a single pass of the graph D^A and computes the actual ObjectRank values. Notice that there is no need for $epsilon$ any more since we derive the precise solution of Equation 6, in contrast to the algorithm of Figure 9 which calculates approximate values. The intuition is that ObjectRank is only transferred in the direction of the topological ordering, so a single pass suffices. Notice that topologically sorting a graph $G(V, E)$ takes time $\Theta(V + E)$ [11] in the general case. In many cases the semantics of the database can lead to a better algorithm. For example, in the papers database, we can efficiently topologically sort the papers by first sorting the conferences by date. This method is applicable for databases where a temporal or other kind of ordering is implied by the link structure.

```
CreateIndexDAG(keywordsList, threshold, α(.), d){
01. Topologically sort nodes in graph D^A;
02. /*Consecutive accesses to D'^A are in topological order.*/
03. For each keyword w in keywordsList do {
04.   MakeOnePass(w,α(.), d);
05.   StoreObjectRanks();
06. }
}
```

Figure 10: Algorithm to create ObjectRank Index for DAGs

In the above example, the DAG property was implied by the semantics. However, in some cases we can infer this property by the structure of the authority transfer schema graph G^A, as the following theorem shows.

Theorem 5.1 *The authority transfer data graph D^A is a DAG if and only if*

- *the authority transfer schema graph G^A is a DAG, or*

- *for every cycle c in G^A, the subgraph D'^A of D^A consisting of the nodes (and the edges connecting them), whose type is one of the schema nodes of c, is a DAG.*

5.3 Almost-DAG algorithm

The most practically interesting case is when the authority transfer data graph D^A is *almost* a DAG, that is, there is a "small" set U of *backedges*, and if these edges are removed, D^A becomes a DAG. Notice that the set U is not unique, that is, there can be many *minimal* (i.e., no edge can be removed from U) sets of backedges. Instead of working with the set of backedges U, we work with the set L of *backnodes*, that is, nodes from which the backedges start. This reduces the number of needed variables as we show below, since $|L| \leq |U|$.

In the papers database example (when author and conference objects are ignored), L is the set of papers citing a paper that was not published previously. Similarly, in the complaints database (Figure 21), most complaints reference previous complaints. Identifying the minimum set of backnodes is NP-complete[5] in the general case. However, the semantics of the database can lead to efficient algorithms. For example, for the databases we discuss in this paper (i.e, the papers and the complaints databases), a backnode is simply an object referencing an object with a newer timestamp.

The intuition of the algorithm (Figure 11) is as follows: the ObjectRank of each node can be split to the DAG-ObjectRank which is calculated ignoring the backedges, and the backedges-ObjectRank which is due to the backedges.

To calculate backedges-ObjectRank we assign a variable c_i to each backnode c_i (for brevity, we use the same symbol to denote a backnode and its ObjectRank), denoting its ObjectRank. Before doing any keyword-specific calculation, we calculate how c_i's are propagated to the rest of the graph D^A (line 5), and store this information in $\mathbf{C}$. Hence C_{ij} is the coefficient with which to multiply c_j when calculating the ObjectRank of node v_i. To calculate $\mathbf{C}$ (lines 13-15) we assume that the backedges are the only source of ObjectRank, and make one pass of the DAG in topological order.

Then, for each keyword-specific base set: (a) we calculate the DAG-ObjectRanks $\mathbf{r}'$ (line 7) ignoring the backedges (but taking them into account when calculating the outgoing degrees), (b) calculate c_i's solving a linear system (line 8), and (c) calculate the total ObjectRanks (line 10) by adding the backedge-ObjectRank ($\mathbf{C} \cdot \mathbf{c}$) and the DAG-ObjectRank($\mathbf{r}'$). Each line of the system of line 8 corresponds to a backnode $c_i \equiv v_j$ (i.e., the ith backnode is the jth node of the topologically sorted authority transfer data graph D'^A), whose ObjectRank c_i is the sum of the backedge-ObjectRank ($\mathbf{C}_j \cdot \mathbf{c}$) and the DAG-ObjectRank ($\mathbf{r}'_j$). The overline notation on the matrices of this equation selects the L lines from each table that correspond to the backnodes. We further explain the algorithm using an example.

[5]Proven by reducing Vertex Cover to it.

```
CreateIndexAlmostDAG(keywordsList, threshold, α(.), d){
01. c: vector of ObjectRanks of backnodes;
02. Identify backnodes, and topologically sort
    the DAG (D^A without the backedges) D'^A;
03. /*Consecutive accesses to D'^A are in topological order.*/
04. /*Backedges are considered in D'^A for α(.) .*/
05. C=BuildCoefficientsTable();
06. For each keyword w in keywordsList do {
07.   Calculate ObjectRanks vector r' for D'^A executing
      MakeOnePass(w,α(.), d);
08.   Solve c = C̄ · c + r̄';
09.   /*D̄ denotes keeping only the lines of D
      corresponding to backnodes.*/
10.   r = C · c + r'
11.   StoreObjectRanks();
12.  }
}
BuildCoefficientsTable(){
13. For each node v_j do
14.   r(v_j) = d · Σ_{backnode c_i points at v_j} (α(c_i → v_j) · c_i) +
      d · Σ_{non−backnode v_l points at v_j} (α(v_l → v_j) · r(v_l));
15. Return C, such that r = C · c
}
```

Figure 11: Algorithm to create ObjectRank Index for *almost* DAGs

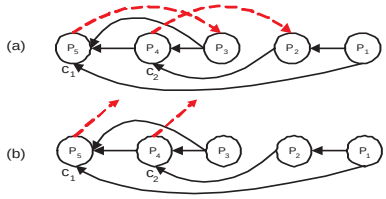

Figure 12: Almost DAG.

Example 1 *The graph D^A is shown in Figure 12 (a). Assume $d = 0.5$ and all edges are of the same type t with authority transfer rate $\alpha(t) = 1$. First we topologically sort the graph and identify the backnodes $c_1 \equiv P_5, c_2 \equiv P_4$. Then we create the coefficients table C (line 5), as follows:*

$$\begin{aligned}
r(P_1) &= 0 \\
r(P_2) &= 0.5 \cdot 0.5 \cdot c_2 = 0.25 \cdot c_2 \\
r(P_3) &= 0.5 \cdot c_1 \\
r(P_4) &= 0.5 \cdot r(P_2) + 0.5 \cdot 0.5 \cdot r(P_3) = \\
&\quad 0.125 \cdot c_1 + 0.125 \cdot c_2 \\
r(P_5) &= 0.5 \cdot 0.5 \cdot r(P_3) + 0.5 \cdot 0.5 \cdot r(P_4) = \\
&\quad 0.156 \cdot c_1 + 0.031 \cdot c_2
\end{aligned}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & 0.25 \\ 0.5 & 0 \\ 0.125 & 0.125 \\ 0.156 & 0.031 \end{bmatrix}$$

Assume we build the index for one keyword w contained in nodes P_1, P_3. We calculate (line 7) ObjectRanks for D'^A (taken by removing the backedges (dotted lines) from D^A).

$$
\begin{aligned}
r(P_1) &= 0.5 \\
r(P_2) &= 0.5 \cdot 0.5 \cdot r(P_1) = 0.125 \\
r(P_3) &= 0.5 \\
r(P_4) &= 0.5 \cdot 0.5 \cdot r(P_3) + 0.5 \cdot r(P_2) = \\
&\quad 0.188 \\
r(P_5) &= 0.5 \cdot 0.5 \cdot r(P_4) + 0.5 \cdot 0.5 \cdot r(P_3) + \\
&\quad 0.5 \cdot 0.5 \cdot r(P_1) = 0.297
\end{aligned}
$$

$\mathbf{r}' = [0.5\ 0.125\ 0.5\ 0.188\ 0.297]^T$

Solving the equation of line 8:

$$
\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0.156 & 0.031 \\ 0.125 & 0.125 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} 0.297 \\ 0.188 \end{bmatrix}
$$

we get: $\mathbf{c} = [0.361\ 0.263]^T$*, where the overline-notation selects from the matrices the 5-th and the 4-th lines, which correspond to the backnodes* c_1 *and* c_2 *respectively. The final ObjectRanks are (line 10):* $\mathbf{r} = [0.5\ 0.190\ 0.680\ 0.266\ 0.361]^T$.

This algorithm can be viewed as a way to reduce the $n \times n$ ObjectRank calculation system of Equation 6, where n is the size of the graph, to the much smaller $|L| \times |L|$ equations system of line 8 of Figure 11. Interestingly, the two equations systems have the same format $\mathbf{r} = \mathbf{A}\mathbf{r} + \mathbf{b}$, only with different coefficient tables $\mathbf{A}, \mathbf{b}$. The degree of reduction achieved is inversely proportional to the number of backnodes.

The linear, first-degree equations system of line 8 can be solved using any of the well-studied arithmetic methods like Jacobi and Gauss-Seidel [14], or even using the PageRank iterative approach which is simpler because we do not have to solve each equation with respect to a variable. The latter is shown to perform better in Section 7.

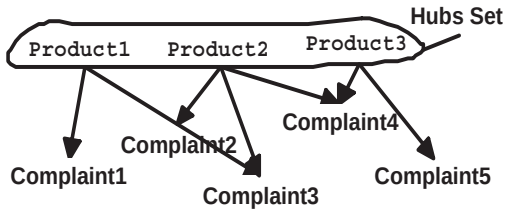

Figure 13: Hierarchical-graph.

5.4 Algorithm for graphs with small vertex cover

Similarly to the almost-DAG case, we can reduce the ObjectRank calculation to a much smaller system (than the one of Equation 6) if authority transfer data graph D^A contains a relatively small vertex cover H. For example, consider a subset of the complaints database (Figure 21) consisting of the products and the complaints (without the reference edge to other complaints). Then H is the set of the products (Figure 13).[6] We call the nodes of H hub-nodes.

[6] A complaint can refer to more than one products.

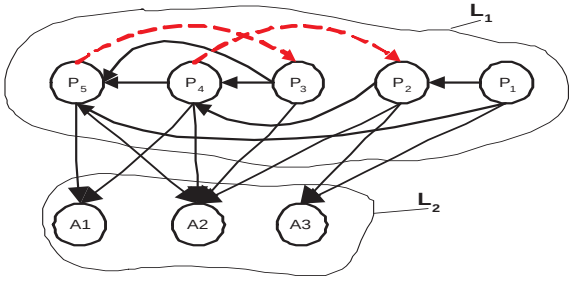

Figure 14: Serializable Graph.

The intuition of the algorithm is the following: Let $r(h_i)$ be the ObjectRank of hub-node h_i. First, the ObjectRank of every non-hub-node i is expressed as a function of the ObjectRanks of the hub-nodes pointing to i. Then the $r(h_i)$ is expressed as a function of the non-hub-nodes pointing to h_i. This expression is equal to $r(h_i)$, so we get $|H|$ such equations for the $|H|$ hub-nodes. Hence we reduce the computation to a $|H| \times |H|$ linear, first-degree system. Notice that we omit the details of the optimizations of Sections 5.4 and 5.5 due to lack of space.

5.5 Serializing ObjectRank Calculation

This section shows when and how we can *serialize* the ObjectRank calculation of the whole graph $D^A(V_D, E_D^A)$ over ObjectRank calculations for disjoint, non-empty subsets $L_1, \ldots, L_r$ of V_D, where $L_1 \cup \ldots \cup L_r \equiv V_D$. The calculation is serializable if we first calculate the ObjectRanks for L_1, then use these ObjectRanks to calculate the ObjectRanks of L_2 and so on.

For example, consider the subset of the papers database consisting of the papers, their citations and the authors, where authority is transferred between the papers and from a paper to its authors (and not vice versa). Figure 14 shows how this authority transfer data graph can be serialized. In particular, we first calculate the ObjectRanks for the nodes in L_1 and then for the nodes in L_2, as we elaborate below.

To define when the calculation is serializable, we first define the graph $D'^A(V', E')$ with $V' = \{L_1 \cup \ldots \cup L_r\}$ and $E' = \{(L_i, L_j) | \exists (v_i, v_j) \in E_D^A \wedge v_i \in L_i \wedge v_j \in L_j\}$. That is, there is an edge (L_i, L_j) in D'^A if there is an edge between two nodes $v_i \in L_i, v_j \in L_j$ of D^A. The following theorem defines when the ObjectRank calculation is serializable.

Theorem 5.2 *The ObjectRank calculation for* D^A *is serializable iff* D'^A *is a DAG.*

The algorithm works as follows: Let $L_1, \ldots, L_r$ be topologically ordered. First, the ObjectRanks of the nodes in L_1 are computed ignoring the rest of D^A. Then we do the same for L_2, including in the computation the set I of nodes (and the corresponding connecting edges) of L_1 connected to nodes in L_2. Notice that the ObjectRanks of the nodes in I are not changed since there is no incoming edge from any node of L_2 to any node in I. Notice that any of

the ObjectRank calculations methods described above can be used in each subset L_i.

5.6 Manipulating Initial ObjectRank values

All algorithms so far assume that we do a fresh execution of the algorithm for every keyword. However, intuitively we expect nodes with high global ObjectRank to also have high ObjectRank with respect to many keywords. We exploit this observation by assigning the global ObjectRanks as initial values for each keyword specific calculation.

Furthermore, we investigate a space vs. time tradeoff. In particular, assume we have limitations on the index size. Then we only store a prefix (the first p nodes) of the nodes' list (recall that the lists are ordered by ObjectRank) for each keyword. During the query stage, we use these values as initial values for the p nodes and a constant (we experimentally found 0.03 to be the most efficient for our datasets) for the rest[7]. Both ideas are experimentally evaluated in Section 7.1.

6 Relevance Feedback Survey

To evaluate the quality of the results of ObjectRank, we conducted two surveys. The first was performed on the DBLP database, with eight professors and Ph.D. students from the UC, San Diego database lab, who were not involved with the project. The second survey used the publications database of the IEEE Communications Society (COMSOC) [8] and involved five senior Ph.D. students from the Electrical Engineering Department.

Each participant was asked to compare and rank two to five lists of top-10 results for a set of keyword queries, assigning a score of 1 to 10, according to the relevance of the results list to the query. Each result list was generated by a different variation of the ObjectRank algorithm. One of the results lists in each set was generated by the "default" ObjectRank configuration which used the authority transfer schema graph of Figure 4 and $d = 0.85$. The users knew nothing about the algorithms that produced each result list. The survey was designed to investigate the quality of ObjectRank when compared to other approaches or when changing the adjusting parameters.

Effect of keyword-specific ranking. First, we assess the basic principle of ObjectRank, which is the keyword-specific scores. In particular, we compared the default (that is, with the parameters set to the values discussed in Section 1) ObjectRank with the global ranking algorithm that sorts objects that contain the keywords according to their global ObjectRank (where the base-set contains all nodes). Notice that this is equivalent to what Google used to[9] do for Web pages, modulo some minor difference on the calculation of the relevance score by Google. The DBLP survey included results for two keyword queries: "OLAP" and "XML". The score was 7:1 and 5:3 in favor of the keyword-specific ObjectRank for the first and second keyword query respectively. The COMSOC survey used the keywords "CDMA" and "UWB (ultra wideband)" and the scores were 4:1 and 5:0 in favor of the keyword-specific approach respectively.

Effect of authority transfer rates. We compared results of the default ObjectRank with a simpler version of the algorithm that did not use different authority transfer rates for different edge types, i.e., all edge types were treated equally. In the DBLP survey, for both keyword queries, "OLAP" and "XML", the default ObjectRank won with scores 5:3 and 6.5:1.5 (the half point means that a user thought that both rankings were equally good) respectively. In the COMSOC survey, the scores for "CDMA" and "UWB" were 3.5:1.5 and 5:0 respectively.

Effect of the damping factor d. We tested three different values of the damping factor d: 0.1, 0.85, and 0.99, for the keyword queries "XML" and "XML AND Index" on the DBLP dataset. Two points were given to the first choice of a user and one point to the second. The scores were $2.5 : 8 : 13.5$ and $10.5 : 11.5 : 2$ (the sum is 24 since there are 8 users times 3 points per query) respectively for the three d values. We see that higher d values are preferred for the "XML", because "XML" is a very large area. In contrast, small d are preferable for "XML AND Index", because few papers are closely related to both keywords, and these papers typically contain both of them. The results were also mixed in the COMSOC survey. In particular, the damping factors 0.1, 0.85, and 0.99 received scores of 5:6:4 and 4.5:3.5:7 for the queries "CDMA" and "UWB" respectively.

Effect of changing the weights of the keywords. We compared the combining functions for AND semantics of Equations 7 and 9 for the two-keyword queries "XML AND Index" and "XML AND Query", in the DBLP survey. The use of the normalizing exponents proposed in Section 3.3 was preferred over the simple product function with ratios of 6:2 and 6.5:1.5 respectively. In the COMSOC survey, the same experiment was repeated for the keyword query "diversity combining". The use of normalizing exponents was preferred at a ratio of 3.5:1.5.

7 Experiments

In this section we experimentally evaluate the system and show that calculating the ObjectRank is feasible, both in the preprocessing and in the query execution stage. For the evaluation we use two real and a set of synthetic datasets: COMSOC is the dataset of the publications of the IEEE Communications Society [10], which consists of $55,000$ nodes and $165,000$ edges. DBLPreal is a subset of the DBLP dataset, consisting of the publications in twelve database conferences. This dataset contains $13,700$ nodes and $101,500$ edges. However, these datasets are too small to evaluate the index creation algorithms. Hence, we also

[7] Notice that, as we experimentally found, using the global ObjectRanks instead of a constant for the rest nodes is less efficient. The reason is that if a node u is not in the top-p nodes for keyword k, u probably has a very small ObjectRank with respect to k. However u may have a great global ObjectRank.

[8] http://www.comsoc.org

[9] Google's current ranking algorithm is not disclosed.

[10] http://www.comsoc.org

created a set of artificial datasets shown in Table 2, using the words of the DBLP dataset. The outgoing edges are distributed uniformly among papers, that is, each paper cites on average 10 other papers. The incoming edges are assigned by a non-uniform random function, similar to the one used in the TPC-C benchmark [11], such that the top-10% of the most cited papers receive 70% of all the citations.

name	*#nodes*	*#edges*
DBLP30	3,000	30,000
DBLP100	10,000	100,000
DBLP300	30,000	300,000
DBLP1000	100,000	1,000,000
DBLP3000	300,000	3,000,000

Table 2: Synthetic Datasets.

To store the databases in a RDBMS, we decomposed them into relations according to the relational schema shown in Figure 15. Y is an instance of a conference in a particular year. PP is a relation that describes each paper *pid2* cited by a paper *pid1*, while PA lists the authors *aid* of each paper *pid*. Notice that the two arrows from P to PP denote primary-to-foreign-key connections from *pid* to *pid1* and from *pid* to *pid2*. We ran our experiments using the Oracle 9i RDBMS on a Xeon 2.2-GHz PC with 1 GB of RAM. We implemented the preprocessing and query-processing algorithms in Java, and connect to the RDBMS through JDBC.

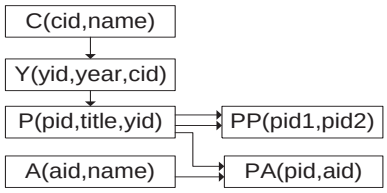

Figure 15: Relational schema.

The experiments are divided into two classes. First, we measure how fast the ObjectRank Execution module (Figure 8) calculates the ObjectRanks for all keywords and stores them into the ObjectRank Index, using the *CreateIndex* algorithm of Figure 9. The size of the ObjectRank Index is also measured. This experiment is repeated for various values of *epsilon* and *threshold*, and various dataset sizes. Furthermore, the General ObjectRank algorithm is compared to the almost-DAG algorithm, and the effect of using various initial ObjectRank values is evaluated. Second, in Section 8 the Query module (Figure 8) is evaluated. In particular, we measure the execution times of the combining algorithm (Section 4) to produce the top-k results, for various values of k, various numbers of keywords m, and OR and AND semantics.

7.1 Preprocessing stage

General ObjectRank algorithm. Tables 3 and 4 show how the storage space for the ObjectRank index decreases as the ObjectRank *threshold* of the stored objects increases,

[11] http://www.tpc.org/tpcc/

threshold	*time (sec)*	*nodes/keyword*	*size (MB)*
0.3	3702	84	2.20
0.5	3702	67	1.77
1.0	3702	46	1.26

Table 3: Index Creation for DBLPreal for $epsilon = 0.1$

threshold	*time (sec)*	*nodes/keyword*	*size (MB)*
0.05	80829	9.4	1.17
0.07	80829	8.3	1.08
0.1	80829	7.7	1.03

Table 4: Index Creation for COMSOC for $epsilon = 0.05$

for the real datasets. Notice that DBLPreal and COMSOC have $12,341$ and $40,577$ keywords respectively. Also notice that much fewer nodes per keyword have ObjectRank above the $threshold$ in COMSOC, since this dataset is more sparse and has more keywords. The time to create the index does not change with *threshold* since *threshold* is not used during the main execution loop of the CreateIndex algorithm. Tables 5 and 6 show how the index build time decreases as *epsilon* increases. The reason is that fewer iterations are needed for the algorithm to converge, on the cost of lower accuracy of the calculated ObjectRanks. Notice that the storage space does not change with *epsilon*, as long as $epsilon < threshold$.

Table 7 shows how the execution times and the storage requirements for the ObjectRank index scale with the database size for the DBLP synthetic datasets for $epsilon = 0.05$ and $threshold = 0.1$. Notice that the average number of nodes having ObjectRank higher than the *threshold* increases considerably with the dataset size, because the same keywords appear multiple times.

General ObjectRank vs. almost-DAG algorithm. Figure 16 compares the index creation time of the General ObjectRank algorithm (*Gen-OR*) and two versions of the almost-DAG algorithm, on the DBLP1000 dataset, for various number of backnodes. The *algebraic* version (*Alg-A-DAG*) precisely solves the $\mathbf{c} = \overline{\mathbf{C}} \cdot \mathbf{c} + \overline{\mathbf{r}'}$ system using an off the self algebraic solver. The *PageRank* version (*PR-A-DAG*) solves this system using the PageRank [8] iterative method. The measured times are the average processing time for a single keyword and do not include the time to retrieve the base-set from the inverted text index, which is common to all methods. Also, the time to calculate $\mathbf{C}$ is omitted, since it $\mathbf{C}$ is calculated once for all keywords, and it requires a single pass over the graph. The *Iterative part* of the execution times corresponds to the one pass we perform on the DAG subgraph to calculate $\mathbf{r}'$ for almost-DAG algorithms, and to the multiple passes which consist the whole computation for the General ObjectRank algorithm.

Also, notice that $epsilon = 0.1$ for this experiment (the

epsilon	*time (sec)*	*nodes/keyword*	*size (MB)*
0.05	3875	67	1.77
0.1	3702	67	1.77
0.3	3517	67	1.77

Table 5: Index Creation for DBLPreal for $threshold = 0.5$

epsilon	*time (sec)*	*nodes/keyword*	*size (MB)*
0.05	80829	7.7	1.03
0.07	77056	7.7	1.03
0.1	74337	7.7	1.03

Table 6: Index Creation for COMSOC for $threshold = 0.1$

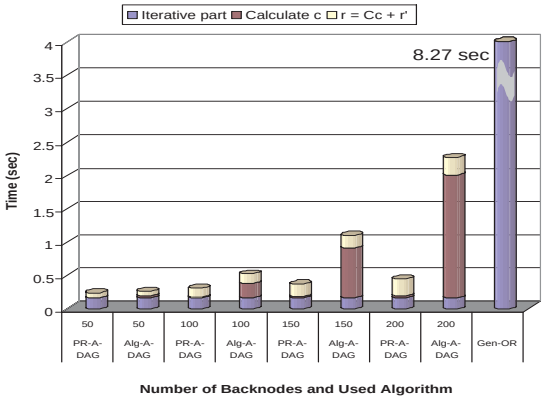

Figure 16: Evaluate almost-DAG algorithm.

threshold value is irrelevant since it does not affect the processing time, but only the storage space). The time to do the topological sorting is about 20 sec which is negligible compared to the time to calculate the ObjectRanks for all keywords.

Initial ObjectRanks. This experiment shows how the convergence of the General ObjectRank algorithm is accelerated when various values are set as initial ObjectRanks. In particular, we compare the naive approach, where we assign an equal initial ObjectRank to all nodes, to the global-as-initial approach, where the global ObjectRanks are used as initial values for the keyword-specific ObjectRank calculations. We found that on DBLPreal (COMSOC), for $epsilon = 0.1$, the naive and global-as-initial approaches take 16.3 (15.8) and 12.8 (13.7) iterations respectively.

Furthermore, we evaluate the space vs. time tradeoff described in Section 5.6. Tables 8 and 9 show the average number of iterations for $epsilon = 0.1$ on DBLPreal and COMSOC respectively for various values of the precomputed list length p.

8 Query Stage Experiments

Figures 17 and 18 show how the average execution time changes for varying number of requested results k, for two-keyword queries on DBLPreal and COMSOC respectively. We used the index table created with $epsilon = 0.1$ (0.05)

dataset	*time (sec)*	*nodes/keyword*	*size (MB)*
DBLP30	2933	6	0.3
DBLP100	11513	21	0.7
DBLP300	45764	65	1.7
DBLP1000	206034	316	7.9
DBLP3000	6398043	1763	43.6

Table 7: Index Creation for Synthetic Datasets.

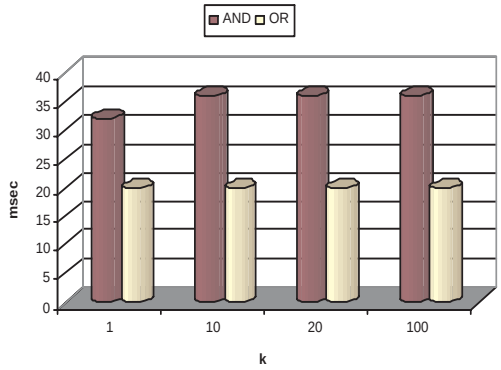

Figure 17: Varying k in DBLPreal.

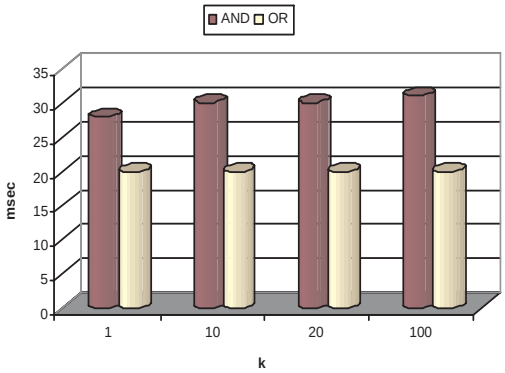

Figure 18: Varying k in COMSOC.

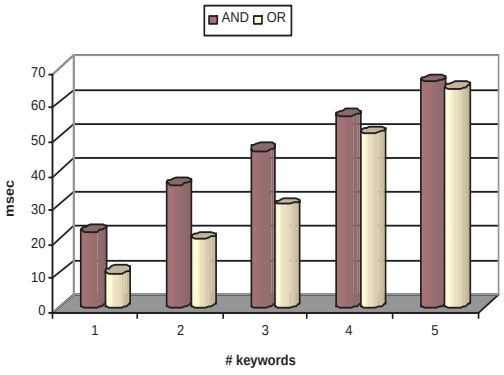

Figure 19: Varying # keywords in DBLPreal.

List length p	*iterations*
13700	1
13000	1.2
8000	1.8
2500	3
800	8.7
100	13.3
0	16.3

Table 8: Number of iterations for various lengths of precomputed lists for DBLPreal

List length p	*iterations*
55000	1
54000	2.9
30000	5.3
13000	6.5
1600	7.8
400	10.7
25	13
0	15.8

Table 9: Number of iterations for various lengths of precomputed lists for COMSOC

and $threshold = 0.3$ (0.1) for DBLPreal (COMSOC). The times are averaged over 100 repetitions of the experiment. Notice that the time does not increase considerably with k, due to the fact that about the same number of random accesses are needed for small k values, and the processing time using the Threshold Algorithm is too small. Also notice that the times for COMSOC are slightly smaller than DBLPreal, because the inverted lists are shorter. Figures 19 and 20 show that the execution time increases almost linearly with the number of keywords, which again is due to the fact that the disk access time to the ObjectRank lists is the dominant factor, since the processing time is too small. Finally, notice that the execution times are shorter for OR semantics, because there are more results, which leads to a smaller prefix of the lists being read, in order to get the top-k results.

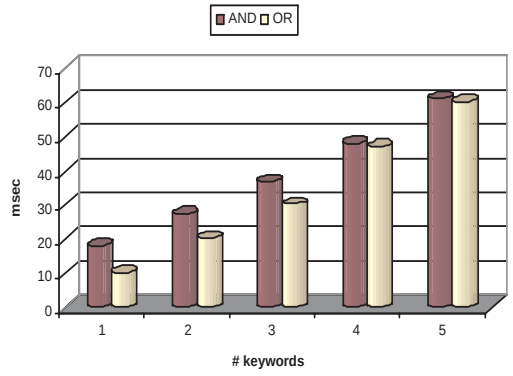

Figure 20: Varying # keywords in COMSOC.

9 Related Work

We first present how state-of-the-art works rank the results of a keyword query, using traditional IR techniques and exploiting the link structure of the data graph. Then we discuss about related work on the performance of link-based algorithms.

Traditional IR ranking. Currently, all major database vendors offer tools [2, 3, 1] for keyword search in single attributes of the database. That is, they assign a score to an attribute value according to its relevance to the keyword query. The score is calculated using well known ranking functions from the IR community [27], although their precise formula is not disclosed. Recent works [6, 19, 20, 5] on keyword search on databases, where the result is a tree of objects, either use similar IR techniques [6], or use the simpler boolean semantics [19, 20, 5], where the score of an attribute is 1 (0) if it contains (does not contain) the keywords.

The first shortcoming of these semantics is that they miss objects that are very related to the keywords, although they do not contain them (Section 1). The second shortcoming is that the traditional IR semantics are unable to meaningfully sort the resulting objects according to their relevance to the keywords. For example, for the query "XML", the paper [15] on Quality of Service that uses an XML-based language, would be ranked as high as a classic book on XML [4]. Again, the relevance information is hidden in the link structure of the data graph.

Link-based semantics. To the best of our knowledge, Savoy [28] was the first to use the link-structure of the Web to discover relevant pages. This idea became more popular with PageRank [8], where a global score is assigned to each Web page as we explain in Section 2. However, directly applying the PageRank approach in our problem is not suitable as we explain in Section 1. HITS [24] employs mutually dependant computation of two values for each web page: hub value and authority. In contrast to PageRank, it is able to find relevant pages that do not contain the keyword, if they are directly pointed by pages that do. However, HITS does not consider domain-specific link semantics and does not make use of schema information. The relevance between two nodes in a data graph can also be viewed as the resistance between them in the corresponding electrical network, where a resistor is added on each edge. This approach is equivalent to the random walk model [12].

Richardson et al. [26] propose an improvement to PageRank, where the random surfer takes into account the relevance of each page to the query when navigating from one page to the other. However, they require that every result contains the keyword, and ignore the case of multiple keywords. Haveliwala [18] proposes a topic-sensitive PageRank, where the topic-specific PageRanks for each page are precomputed and the PageRank value of the most relevant topic is used for each query. Both works apply to the Web and do not address the unique characteristics of structured databases, as we discuss in Section 1. Furthermore, they offer no adjusting parameters to calibrate the

system according to the specifics of an application.

Recently, the idea of PageRank has been applied to structured databases [16, 21]. XRANK [16] proposes a way to rank XML elements using the link structure of the database. Furthermore, they introduce a notion similar to our ObjectRank transfer edge bounds, to distinguish between containment and IDREF edges. Huang et al. [21] propose a way to rank the tuples of a relational database using PageRank, where connections are determined dynamically by the query workload and not statically by the schema. However, none of these works exploits the link structure to provide keyword-specific ranking. Furthermore, they ignore the schema semantics when computing the scores.

Performance. A set of works [17, 10, 22, 23] have tackled the problem of improving the performance of the original PageRank algorithm. [17, 10] present algorithms to improve the calculation of a global PageRank. Jeh and Widom [22] present a method to efficiently calculate the PageRank values for multiple base sets, by precomputing a set of *partial vectors* which are used in runtime to calculate the PageRanks. The key idea is to precompute in a compact way the PageRank values for a set of hub pages, through which most of the random walks pass. Then using these hub PageRanks, calculate in runtime the PageRanks for any base set consisting of nodes in the hub set. However, in our case it is not possible to define a set of hub nodes, since any node of the database can be part of a base set.

10 Conclusion and Future Work

We presented an adjustable framework to answer keyword queries using the authority transfer paradigm, which we believe is applicable to a significant number of domains (though obviously not meaningful for every database). We showed that our framework is efficient and semantically meaningful, with an experimental evaluation and user surveys respectively.

We investigated how this framework can be applied with small modifications to applications other than bibliographic, where the authority transfer intuition is applicable. For example, consider a complaints database (Figure 21), which stores the complaint reports of customers regarding products of the company. Assume we wish to rank the complaint reports according to their urgency, given that the goal of the company is to keep the "good" customers satisfied, and the "goodness" of a customer is the total sales associated with him/her. Then, the base set for the computation of the global ObjectRank is the set of customers, and each customer is given a base ObjectRank proportional to his/her total sales amount. A reasonable assignment of authority transfer rates is shown in Figure 21.

Finally, notice that many adjusting parameters are input during the preprocessing stage (Figure 8). We plan to investigate the opportunities of moving some of these parameters to the query stage, that is, perform on-demand ObjectRank calculation instead of storing an ObjectRank index. This would allow the user to input more parameters to calibrate his/her query.

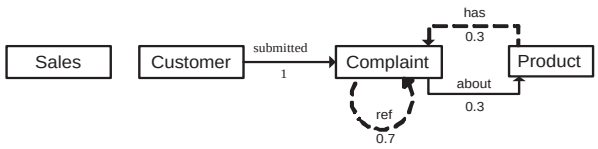

Figure 21: Authority transfer schema graph for Complaints database.

11 Acknowledgements

We thank Michael Sirivianos for creating the Web interface of the ObjectRank demo. We also thank the reviewers for their useful comments.

References

[1] http://msdn.microsoft.com/library/. 2001.

[2] http://technet.oracle.com/products/text/content.html. 2001.

[3] http://www.ibm.com/software/data/db2/extenders/textinformation/index.html. 2001.

[4] S. Abiteboul, D. Suciu, and P. Buneman. Data on the Web : From Relations to Semistructured Data and Xml. *Morgan Kaufmann Series in Data Management Systems*, 2000.

[5] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System For Keyword-Based Search Over Relational Databases. *ICDE*, 2002.

[6] G. Bhalotia, C. Nakhey, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. *ICDE*, 2002.

[7] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. *SIGIR*, 1998.

[8] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *WWW Conference*, 1998.

[9] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. *WWW Conference*, 1998.

[10] Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing PageRank. *CIKM*, 2002.

[11] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. *MIT Press*, 1989.

[12] P. G. Doyle and J. L. Snell. Random Walks and Electric Networks. *Mathematical Association of America, Washington, D. C.*, 1984.

[13] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *ACM PODS*, 2001.

[14] G. H. Golub and C. F. Loan. Matrix Computations. *Johns Hopkins*, 1996.

[15] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. *Journal of Visual Languages and Computing 13(1): 61-95*, 2002.

[16] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. *ACM SIGMOD*, 2003.

[17] T. Haveliwala. Efficient computation of PageRank. *Technical report, Stanford University (http://www.stanford.edu/ taherh/papers/efficient-pr.pdf)*, 1999.

[18] T. Haveliwala. Topic-Sensitive PageRank. *WWW Conference*, 2002.

[19] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, 2002.

[20] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. *ICDE*, 2003.

[21] A. Huang, Q. Xue, and J. Yang. TupleRank and Implicit Relationship Discovery in Relational Databases. *WAIM*, 2003.

[22] G. Jeh and J. Widom. Scaling Personalized Web Search. *WWW Conference*, 2003.

[23] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation Methods for Accelerating PageRank Computations. *WWW Conference*, 2003.

[24] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 1999.

[25] R. Motwani and P. Raghavan. Randomized Algorithms. *Cambridge University Press, United Kingdom*, 1995.

[26] M. Richardson and P. Domingos. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. *Advances in Neural Information Processing Systems 14, MIT Press*, 2002.

[27] G. Salton. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. *Addison Wesley*, 1989.

[28] J. Savoy. Bayesian inference networks and spreading activation in hypertext systems. *Information Processing and Management*, 28(3):389–406, 1992.