Ranking Objects by Following Paths in Entity-Relationship Graphs

Minsuk Kahng, Sangkeun Lee and Sang-goo Lee
School of Computer Science and Engineering
Seoul National University
Seoul, South Korea
{minsuk,liza183,sglee}@europa.snu.ac.kr

ABSTRACT

In this paper, we propose an object ranking method for search and recommendation. By selecting schema-level paths and following them in an entity-relationship graph, it can incorporate diverse semantics existing in the graph. Utilizing this kind of graph-based data models has been recognized as a reasonable way for dealing with heterogeneous data. However, previous work on ranking models using graphs has some limitations. In order to utilize a variety of semantics in multiple types of data, we define a schema path as a basic component of the ranking model. By following the path or a combination of paths, relevant objects could be retrieved or recommended. We present some preliminary experiments to evaluate our method. In addition, we discuss several interesting challenges that can be considered in future work.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models, Information filtering*; H.2.8 [**Database Management**]: Database applications

General Terms

Algorithms, Experimentation

Keywords

object retrieval, graph, ranking model, recommender systems, information retrieval, entity-relationship graph, similarity search

1. INTRODUCTION

With the growing amount of data available on the Web and enterprises, retrieval systems and recommender systems have gained popularity by analyzing the data and providing preferrable objects to the users. In addition to the growing amount of data, the types of data have become more diverse. For example, although traditional retrieval systems only need to consider webpages and textual data in those webpages, current systems consider various types of data, such as products, multimedia objects, information on the map, the user, clickthrough log, and even some contextual information (*e.g.* time and location). Thus processing these heterogeneous data has become an important issue for ranking models of retrieval and recommender systems.

Graph-based data models have recently gained popularity for dealing with these heterogeneous data. By using a graph, various types of data can be easily expressed. In particular, objects in the real world and relationships between them can be represented in the form of an *entity-relationship graph*. Examples include social networks among people, structured knowledge in Wikipedia, hyperlinked Web, and relational data stored in enterprises' databases. A variety of real-world objects and relationships can be easily represented by nodes and edges in the graph. Thus, using graph-based data models has become a reasonable solution for dealing with this heterogeneity.

However, existing work on ranking models using graphs has some limitations in fully capturing the semantics of different types of data. Since different edge types (*e.g.* author-paper, paper-publisher) convey different semantics, we need to deal with them differently. However, much of the work considered only the structure of the graph without utilizing the types of nodes or edges [7, 9]. In order to differentiate the types of data, *ObjectRank* [5] assigned different weights to each type of edge. However, the *edge-level feature* is still insufficient in expressing the delicate meanings embedded in various types of data [20]. While some edge types (*e.g.* author-paper) are not needed for some specific tasks (*e.g.* recommending related conferences given some keywords), they could play an important role in other tasks (*e.g.* finding papers whose authors are the same). Recently, the *path-level feature* has been mentioned by several researchers as a replacement for the edge-level feature [20, 25]. A schema-level path, which is a sequence of edge types, makes it possible for an edge to have a different role depending on the tasks. Therefore, it enables us to create more semantically-enriched applications.

In this paper, we propose an object ranking method that focuses on paths in an entity-relationship graph for the task of search and recommendation. We define a *schema path* in the heterogeneous graph, which helps to capture diverse semantics in the data. The schema path represents the existing workflows of search or recommendation, and beyond that, it can express a new way of ranking by discovering

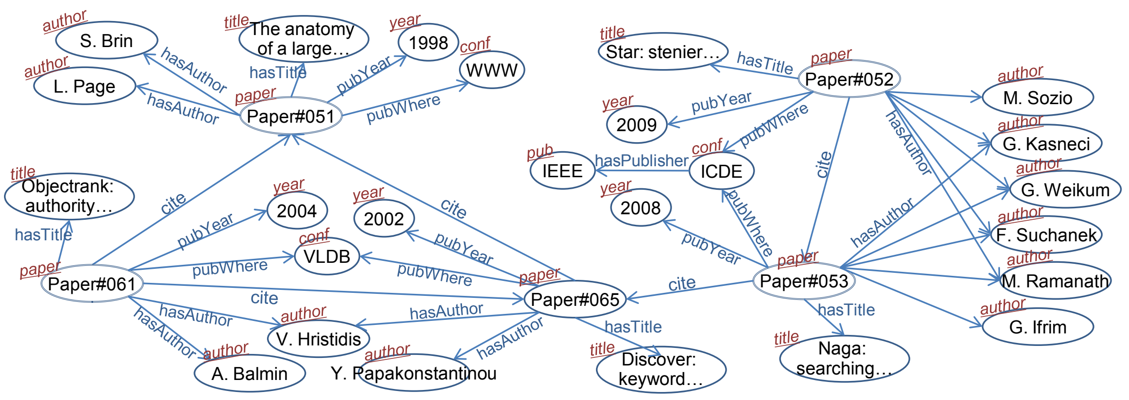

Figure 1: Example of heterogeneous data graph. Using the DBLP publication dataset, we build a data graph which has types on both nodes and edges.

novel paths. Through traversing these paths in the graph, the relevant or preferrable objects are retrieved or recommended to the query users. As on-going work, we present some preliminary experiments, and discuss several challenging issues that can be considered in future work.

The rest of the paper is organized as follows. After reviewing the literature in Section 2, we introduce the data model in Section 3, and formalize the problem in Section 4. Then we describe the ranking method in Section 5. Section 6 provides preliminary experiments on real datasets, and Section 7 discusses several possible directions for future research. Finally, we conclude the paper in Section 8.

2. RELATED WORK

In this section, we review several related areas. We introduce *keyword search in database* and *recommender systems*. Then we present some work on ranking models using graphs for handling heterogeneous data.

The aim of *keyword search in database* is similar to our problem; its goal is to search entities from structured data [22, 9]. Instead of using complex queries like SQL for retrieving structured data in relational databases, *keyword queries* are used to specify the user's information needs. Researchers often express relational data as a graph [14], then try to find nodes that are close to the query nodes. However, most of these methods focus more on the structure of the graph [22] (*e.g.* finding minimal subgraphs) than the meaning of relations. In other words, they do not differentiate the type of edges, which makes it hard for ranking models to capture semantics in the data. Moreover, they usually do not take the task of recommendation into account. The recommendation task usually requires traversing the same edges on the schema graph more than once. For example, *collaborative filtering* [2] finds similar users using user-item relation, and use this relation again for recommending items (e.g. $\texttt{USER} \rightarrow \texttt{ITEM} \rightarrow \texttt{USER} \rightarrow \texttt{ITEM}$). However, they do not assume this scenario because most search tasks do not need to traverse the same edge multiple times.

On the contrary, *recommender systems* infer the query user's unknown preference by finding similar users in order to recommend objects [2], while *keyword search* retrieves entities that exactly match the query. If we take a music recommendation as an example, instead of recommending songs that have been played many times before by the query user, they recommend new songs that have not been listened before by him, but he may likely to listen to. However, since it primarily focuses on only two types of entities, user and item, most well-known methods cannot be easily extended to incorporate multiple types of data. Some researchers try to incorporate various contextual information into the recommender systems [3, 15, 23] and build flexible querying framework [1, 18], but most of them do not deal with heterogeneous data. In order to process multiple types of data and utilize the advantages of finding similar objects, we define schema paths that enable us to find similar objects in a heterogeneous graph.

There are many papers that recognize the power of graphs, and use graph data models for ranking. The earlier work, such as *PageRank* [7], only focused on homogeneous data. After that, *ObjectRank* [5] extended *PageRank* by assigning different edge weights depending on the type of edges, then ranks objects using random walks. Recently, some researchers have sporadically reported that path instead of edge in a graph is a key feature in this kind of heterogeneous graph. Sun *et al.* proposed meta path-based framework, *PathSim* [25], for top-k similarity search. Lao *et al.* proposed *path-constrained random walks* model [20]. They built a path tree and combined the result of several paths. Varadarajan *et al.* [29] proposed a query language for flexible querying on graph by using the path. The *Semantic Web* commnity has been trying to rank objects in the RDF graph [16, 27, 12]. The user writes SPARQL query language to inform the system how to leverage the typed RDF graph [12]. Instead of using SPARQL, Dritsou *et al.* used path query for querying RDF databases [11]. We think that our contribution lies in not only introducing a ranking method that can be used for a variety of search and recommendation tasks, but also presenting research directions for the path-based ranking based on the previous work focusing on the path in the graph.

3. DATA MODEL

The data are represented as a heterogeneous *data graph*, and we build *schema graph* which contains the type information of nodes and edges in the data graph. We assume the data conveys several different types of objects (*e.g.* documents, movies, users, tags, and textual description) and relationships between these objects. After transforming these data into the graph, multiple types of nodes and edges are created.

Definition 1. ***Data Graph.*** *A data graph is a typed directed graph $G_D = (V, E, T_V, T_E)$, where V is a set of nodes, $E \subseteq V \times V$ is a multi-set of edges, T_V is a set of node types, and T_E is a set of edge types. There are a type mapping function of the nodes $\phi_V : V \rightarrow T_V$, and that of the edges $\phi_E : E \rightarrow T_E$. Each node $v \in V$ is uniquely assigned to a node type $\phi(v) \in T_V$, and each edge $e(v_i \rightarrow v_j) \in E$ is also assigned to a single edge type $\phi(e) \in T_E$.*

Figure 1 shows the example of heterogeneous data graph. The dataset is collected from the DBLP publication database[1], and the node type of the graph consists of paper, title, author, conference, year, and publisher.

A schema graph is defined by expressing type information of the above graph.

Definition 2. ***Schema Graph for Data Graph.*** *A schema graph is a directed graph $G_S = (T_V, T_E)$, where T_V is a set of nodes that are the same as the set of node types in the data graph G_D, and similarly $T_E \subseteq T_V \times T_V$ is a set of edges that represent the edge types in G_D.*

There are some notable differences between other graph-based data models and our model. The data model used in *keyword search in database* often has attributes on nodes [9, 22]. Thus, in order to analyze the graph, both the structure of graph and attributes of nodes need to be considered [26]. On the other hand, the RDF data model does not have attributes and types on the nodes [16, 27]. The data model looks more simple, but it is little more difficult to deal with the type of nodes because the node type is not treated as a special feature of the data model. Our model is a hybrid of these two models, which means our nodes do not contain attributes, but have types. Therefore, we only need to analyze the structure of graph, and it is easier to capture semantics of both nodes and edges. We note that our data model is similar to *heterogeneous information networks* from Han *et al.* [13].

3.1 Example: Building Graphs from Relational Databases

The data storage models that express objects and relationships, such as relational database (RDB) and XML, could be transformed into our data model. As a representative example, we explain how to convert relational database into our model.

3.1.1 Data Transformation

The data stored in the relational database should be converted with minimal loss of information. The method we explain here is similar to the way of transforming RDB to RDF discussed in the *Semantic Web* community [24].

The procedure is as follows. First, we build a schema graph. A node is created for each relation. If there exist foreign-key relationships between relations, edges are connected to them. Then we also create a node for each attribute in the relation, and connect the relation node and the attribute node with an edge. Here, not all attributes need to be included. Similarly, a data graph is constructed. While the schema graph consists of relations and attributes, the data graph consists of tuples and values. Figure 3 shows

[1]http://dblp.uni-trier.de/xml/

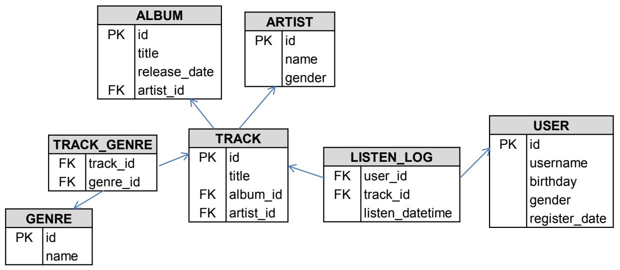

Figure 2: Example of relational database collected from a music streaming service

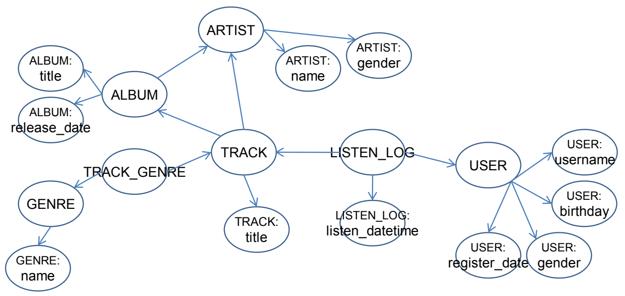

Figure 3: Schema graph built from music database. The edge types are omitted.

the schema graph constructed from the database of a music streaming service depicted in Figure 2[2].

3.1.2 Feature Node Extraction

After transforming the original data, extracting features from value nodes helps to capture diverse semantics. Instead of using the raw data values in the database, we extract several features from the values based on their data type (*e.g.* long text, timestamp). Firstly, terms can be extracted from textual values based on the idea of *bag-of-words* [10]. For instance, from the title 'DISCOVER: Keyword Search ...' [14], we extract term nodes like 'discover', 'keyword', 'search', etc. We also extract several features from the values whose type is timestamp [4]. For example, from the timestamp '2 August 2011, 11:20 AM', the feature nodes '2011' (year), '8' (month), '11' (hour), 'Tuesday' (day of week) could be generated. After defining these kinds of rules based on their data types, feature nodes and edges are added to the data graph. Figure 4 shows how we add the nodes and edges.

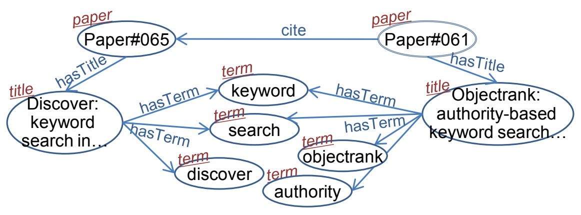

Figure 4: Subgraph of the data graph with feature nodes and edges. Several term nodes are added, then edges between title nodes and term nodes are formed.

[2]The edge types are omitted since it is straightforward to note the meaning of the relations.

4. PROBLEM

In this section, we formalize the problem. Given a query and target type, the system returns top-k objects (nodes).

Definition 3. ***Object Ranking given Query.*** *Given a schema graph $G_S = (T_V, T_E)$ and data graph $G_D = (V, E, T_V, T_E)$, the user specifies a query $\mathcal{Q}$ and target type of objects $d_r \in T_V$. A query $\mathcal{Q}$ consists of a set of nodes $\mathcal{Q} = \{v_1, \ldots, v_n\}$, where each node $v_k \in \mathcal{Q}$ is in the data graph $\mathcal{Q} \subset V$. A target type of objects d_r is defined by selecting a node in the schema graph G_S. Given a query $\mathcal{Q}$ and target type of objects d_r, the system scores all candidate nodes $v \in V$ of target type $\phi(v) = d_r$, then returns top-k nodes whose scores are high.*

The following examples show how some search and recommendation problems can be defined. To find papers using some keywords, the target type would be a paper, and the query would be several terms. If we want to personalize the result, the query user can be added to the query. Another example is *context-aware recommendation*. It aims to suggest objects (items) by considering a user and his context, such as time and location [3, 15]. The query could be a specific user, timestamp, and location, and the target type could be a song if its domain is music.

5. RANKING OBJECTS WITH PATHS

In this section, we describe how to rank objects using paths. We first describe how to select paths from the schema graph. By following these paths from a query to target, objects are ranked.

5.1 Selecting Paths

We first define a *schema path*, then explain how to find these paths.

5.1.1 Definition of Schema Path

A *schema path* represents a workflow for analyzing the data in order to rank objects. We define it as a sequence of edge types extracted from the schema graph.

Definition 4. ***Schema Path.*** *A schema path p is a sequence of edge types $p = \langle t_1, \ldots, t_m \rangle$, where the edge type $t_k \in T_E \cup \textbf{\textit{Inv}}(T_E)$ comes from the schema graph G_S. Here $\textbf{\textit{Inv}}(T_E)$ returns a set of edges whose direction of the edges in T_E are reversed. The edge types are connected in a line, $\texttt{tail}(t_1) = \phi(v_q)$ where $v_q \in \mathcal{Q}$, $\texttt{head}(t_k) = \texttt{tail}(t_{k+1})$, and $\texttt{head}(t_m) = d_r$*[3]*.*

We present some examples of schema paths. The schema path '$\texttt{(term)} \xrightarrow{hasTerm^{-1}} \texttt{(ARTIST:name)} \xrightarrow{hasName^{-1}} \texttt{(ARTIST)} \xrightarrow{hasArtist^{-1}} \texttt{(TRACK)}$'[4] represents a workflow that first finds artists whose names contain terms in a query, then returns their songs. Another path '$\texttt{(TRACK)} \xrightarrow{hasTrack^{-1}} \texttt{(LISTENLOG)} \xrightarrow{hasUser} \texttt{(USER)} \xrightarrow{hasUser^{-1}} \texttt{(LISTENLOG)} \xrightarrow{hasTrack} \texttt{(TRACK)} \xrightarrow{hasAlbum} \texttt{(ALBUM)}$' expresses a workflow that given a song, finds similar songs based on the fact that two songs are similar if the same user listened to both of them. Then it recommends albums that contain these similar songs.

[3]If there is an edge $e(v_i \to v_j)$ from v_i to v_j, it satisfies $\texttt{head}(e) = v_j$ and $\texttt{tail}(e) = v_i$.

[4]The edge type $\xrightarrow{hasTerm^{-1}}$ is generated by changing the direction of the edge type $\xrightarrow{hasTerm}$.

5.1.2 Finding Candidate Schema Paths

The next step is to find the schema paths for the specific search or recommendation task. The problem lies in that there could be too many candidate paths. When the query and target type of objects are given, the leftmost and rightmost node are determined. Then it becomes possible to generate an infinite number of paths from the graph between these two nodes. In order to restrict the number of candidate paths, we impose some constraints to the schema paths based on the characteristics of the tasks.

We introduce three types of schema paths depending on the task. The first type is used for typical search tasks. The second type can be used for finding similar objects. The last type, which combines these two types, is targeted at the recommendation tasks that include similarity search.

Firstly, typical search tasks do not require the process of finding similar objects. In other words, we only need to find simple connecting paths between the query and target instead of including certain nodes multiple times as in '$\texttt{(PAPER)} \to \texttt{(AUTHOR)} \to \texttt{(PAPER)} \to \texttt{(BOOKTITLE)}$'[5].

Definition 5. ***Simple Connecting Path for Typical Search.*** *This type of schema path p satisfies the condition that all the nodes (i.e. $\texttt{tail}(t_1)$, $\texttt{head}(t_1)$, ..., $\texttt{head}(t_k)$, ..., $\texttt{head}(t_m)$) in the path p are different from each other.*

The next category of the path is targeted at finding similar objects. It represents a workflow that returns similar nodes whose type is the same as the type of the query node $\texttt{tail}(t_1) = \texttt{head}(t_m)$. This type of path can be used to find similar objects for expanding the current state. For example, when searching documents, the process of finding synonyms, called *query expansion*, would be helpful. The path would be $\texttt{(term)} \xrightarrow{hasTerm^{-1}} \texttt{(DOCUMENT)} \xrightarrow{hasTerm} \texttt{(term)}$. We assume the shape of the path is symmetric, that is, the right half of the path is generated by copying the left half.

Definition 6. ***Symmetric Path for Finding Similar Objects.*** *The shape of this type of schema path p looks symmetric. The number of edges m in the path $p = \langle t_1, \ldots, t_m \rangle$ is even $m = 2l$, where l is an integer. After the left half of the schema path is determined, the right half of the path is automatically generated by copying the left half with $\textbf{\textit{inv}}$ function $\textbf{\textit{inv}}(t_k) = t_{m-k+1}$, where $1 \leq k \leq l$ and $\textbf{\textit{inv}}(t)$ returns the edge whose direction is reversed.*

By combining these two types of paths, we can create a path for several recommendation tasks. Some tasks could be decomposed into typical search part and similarity search part. For example, recommendation for songs are done by searching songs that have been listened to by the user, then suggesting songs whose artists are the same. The path would be $\texttt{(USER)} \xrightarrow{hasUser^{-1}} \texttt{(LISTENLOG)} \xrightarrow{hasTrack} \texttt{(TRACK)} \xrightarrow{hasArtist} \texttt{(ARTIST)} \xrightarrow{hasArtist^{-1}} \texttt{(TRACK)}$. This kind of path $p = \langle t_1, \ldots, t_m \rangle$ includes the subset of paths $p = \langle t_a, \ldots, t_b \rangle$ that satisfies the conditions of the symmetric path, where $1 \leq a < b \leq m$.

[5]The edge types are omitted

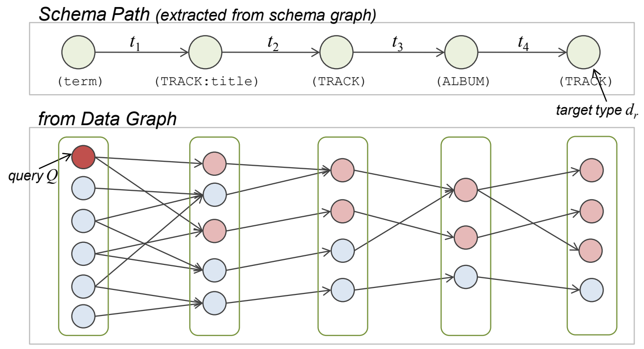

Figure 5: The schema path gives a guideline to follow the edges on the data graph.

To sum up, the number of possible candidate paths decreases by imposing several constraints on the paths. The process of finding candidates can be done by using simple graph operations. The *simple connecting paths* could be found by simple modification of graph traversal algorithms. The *symmetric paths* could be produced by constructing a tree from the node.

5.2 Following a Single Path

By following a single schema path which represents a single workflow, we can obtain a ranked result of target objects. A schema path gives us a guideline to follow the data graph from the query to target.

As we defined, a schema path is a sequence of edge types. Starting from the query node, we follow the edges on the path one by one. Figure 5 shows how we traverse the data graph. The query $\mathcal{Q}$ determines the initial state. Then we consider only a bipartite graph G'_D whose edge type is t_1, which is a subgraph of the data graph G_D. Based on the scoring function $f(v_y; v_x, t_k)$, where $\phi(v_x) = \texttt{tail}(t_k)$ and $\phi(v_y) = \texttt{head}(t_k)$, we obtain the scores of nodes v_y on the right side of the bipartite graph. This propagation process is done iteratively through all edges on the path. After the iterations, the final scores of nodes whose type is target type are obtained.

We formalize this process using a matrix equation [29] as follows:

$$\mathbf{r} = \mathbf{M}^{(m)} \cdots \mathbf{M}^{(1)}\mathbf{q} = \mathbf{A}\mathbf{q}. \tag{1}$$

The query $\mathbf{q}$ represents the initial state by assigning $q_i = 1$ if $q_i \in \mathcal{Q}$, otherwise $q_i = 0$. The matrix $\mathbf{M}$ represents the scoring function for the bipartite graph. The value of i-th row and j-th column of the matrix m_{ij} represents a score of node v_i determined by node v_j, that is $f(v_i; v_j, t_k)$. If we multiply the matrices $\mathbf{M}^{(k)}$ for each edge type t_k on the path, we obtain $\mathbf{A}$. Then by multiplying $\mathbf{A}$ by $\mathbf{q}$, we obtain $\mathbf{r}$ which informs the final score of nodes whose type is the target type d_r.

Then the question arises as to how to define a scoring function $f(v_y; v_x, t_k)$ for the bipartite graph in order to assign the values of the matrices. We introduce three of the possibilities.

The first method is based on simple propagation. A score of node v_y becomes positive if it is connected to the query nodes, otherwise the score is 0. The equation is written as:

$$f(v_y; v_x, t_k) \propto |\{e(v_x \to v_y; t_k)\}|. \tag{2}$$

The numerator $|\{e(v_x \to v_y; t_k)\}|$ denotes the number of edges from v_x to v_y whose edge type is t_k. Since the data graph G_D has a multiset of edges, it is possible to have multiple edges between these two nodes. However, it is hard to differentiate the relatedness of the nodes v_y to the query because this method only considers the semantics of the number of paths from the query node. In order to reflect various semantics of the data, we need scoring functions that utilize additional graph properties such as the degree of the nodes.

The second scoring method is a random-walk based one that captures authority. This method is similar to the random walk with restart model [5, 20]. The equation is written as:

$$f(v_y; v_x, t_k) \propto \frac{|\{e(v_x \to v_y; t_k)\}|}{|\{e(v_x \to v_\cdot; t_k)\}|} = \frac{|\{e(v_x \to v_y; t_k)\}|}{\texttt{deg}^+(v_x; t_k)}. \tag{3}$$

The denominator $\texttt{deg}^+(v_x; t_k)$ refers to the number of edges from the node v_x where the edges' type is t_k. This function is likely to assign higher scores to popular nodes [6]. In other words, the nodes with higher degree tends to get higher scores. However, this kind of authority flow has some disadvantages in that our goal is not always to find authoritative objects, but to find relevant objects with respect to the query.

To tackle this issue, we introduce third scoring function. The equation is written as:

$$f(v_y; v_x, t_k) \propto \frac{|\{e(v_x \to v_\cdot; t_k)\}| \cap |\{e(v_\cdot \to v_y; t_k)\}|}{|\{e(v_x \to v_\cdot; t_k)\}| \cup |\{e(v_\cdot \to v_y; t_k)\}|} \tag{4}$$
$$= \frac{|\{e(v_x \to v_y; t_k)\}|}{\texttt{deg}^+(v_x; t_k) + \texttt{deg}^-(v_y; t_k) - |\{e(v_x \to v_y; t_k)\}|}.$$

By placing $\texttt{deg}^-(v_y; t_k)$, we can diminish the effect of authority flow. It looks similar to the *jaccard similarity coefficient* used for set similarity [8]. We can give some variations to this formula based on properties like the number of edges and degree of the nodes. If we eliminate the term $|\{e(v_x \to v_y; t_k)\}|$ in the denominator, it looks similar to the *dice coefficient* [25]. A detailed study would be helpful in finding more appropriate measures.

5.3 Combining Results from Paths

The results obtained from several paths could be combined. As the search engines incorporate features to improve the quality of ranking [21], aggregating the results from several paths helps to improve the user's satisfaction. The process of aggregating can be done with a weighted combination of results. The weights can be determined by service providers [5], users, or some learning models (e.g. learning-to-rank) [21, 20]. Details about the aggregation process are not the scope of this paper.

6. PRELIMINARY EXPERIMENTS

In this section, we provide preliminary experimental results. We define four retrieval or recommendation tasks using real-world datasets, and show that our method works with some example queries. This paper leaves detailed evaluation including comparison to other methods as future work.

6.1 Datasets

We used two real-world datasets: a music dataset, and DBLP publication dataset.

Table 4: Results for finding similar papers with two different paths.

Path 1	Path 2
1 DISCOVER: Keyword Search in Relational Datab.	1 DISCOVER: Keyword Search in Relational Datab.
2 Discover Relaxed Periodicity in Temporal Datab.	2 ObjectRank: Authority-Based Keyword Search in Datab.
3 Discover Relev. Env. Feature Using Concurrent Reinf. Learning	3 ObjectRank: A System for Auth.-based Search on Datab.
4 Mining Propositional Knowl. Bases to Discover Multi-level Rules	4 Keyword Proximity Search on XML Graphs
5 Mining Multivar. Time-Ser. Sensor Data to Discover Behav. Envel.	5 Efficient IR-Style Keyword Search over Relational Datab.

Table 1: Results for searching songs with keyword query.

Query #1: "snow"		Query #2: "love story"	
1	Snow Snow Snow	1	Love Story
2	Snow	2	White Love Story
3	Snow Dream	3	Theme From Love Story
3	April Snow	4	Story
	. . .	5	Another Story
10	Let It Snow		. . .
10	Melt The Snow	15	Never Ending Story

Table 2: Queries for finding related conferences.

Query #1	"twitter"
Query #2	"graph clustering"
Query #3	"transaction lock protocol optimizing"
Query #4	"language model smoothing"

Table 3: Results for finding related conferences.

Rank	Query #1	Query #2	Query #3	Query #4
1	WWW	KDD	ICDE	SIGIR
2	ICWSM	CIKM	SIGMOD	ECIR
3	CHI	ICDE	VLDB	CIKM
4	ECIR	ICDM	DASFAA	AAAI
5	WSDM	SDM	CIKM	WWW

The music dataset is provided by Bugs[6], one of the largest music streaming service in South Korea. It contains meta-data of songs, albums, and artists, and listening log by the user. The form of raw dataset is relational data, so we convert these data into the graph by following the explanation in Section 3. There are about 902,000 nodes and 2,960,000 edges. The database schema is almost same as that in Figure 2, but there are small differences in that we know the album's genre instead of the track's genre.

Another dataset is collected from DBLP[7]. We collect the publication information of conference papers from 20 major conferences in database and information retrieval area. From its original form XML, we transform it to the relational database [22], then convert it to the graph. There are about 407,000 nodes and 911,000 edges.

6.2 Tasks

We present several scenarios of search and recommendation tasks.

Basic Keyword Search.

The first task is the simple search task. The user writes a keyword query into the music retrieval system, and it retrieves several songs whose titles contain terms in the query. The schema path we define is `(term)` $\xrightarrow{hasTerm^{-1}}$ `(TRACK:title)`, and the third scoring function is used. Through this experiment, we aim to check whether our scoring function works. We use two different keywords queries, and the results are shown in Table 1. First of all, the songs ranked higher if the *term frequency (TF)* is high as the rank of 'Snow Snow Snow' is higher than that of 'Snow'. The length of document also affects the result since 'Snow Dream' is ranked higher than 'Melt The Snow'. Moreover, the *inverse document frequency (IDF)* also affects the result because 'Story' is ranked higher than 'Love' while the term 'Love' appeared more frequently than 'Story' in the dataset.

[6]`http://www.bugs.co.kr/`

[7]`http://dblp.uni-trier.de/xml/`

Table 5: Results for recommending artist by combining two paths.

Rank	Artist's name	Rank for Indiv. Path	
		Path 1	Path 2
1	Black Eyed Peas	1	21
2	8Eight (local)	8	2
3	V.O.S. (local)	-	1
4	Eminem	4	37
5	Ciara	2	86

Recommendation with Longer Path.

The next task uses longer paths for recommending conferences. Given some keyword queries, we want to find some conferences that are highly related to the keywords in the query. We use the second scoring function in order to assign higher scores to authoritative conferences. The schema path is `(term)` $\xrightarrow{hasTerm^{-1}}$ `(PAPER:title)` $\xrightarrow{hasTitle^{-1}}$ `(PAPER)` $\xrightarrow{hasBooktitle}$ `(PAPER:booktitle)`. We use four different keyword queries shown in Table 2. Each of them is related to the areas of Web and social network, data mining, databases, information retrieval, respectively. The results are shown in Table 3. and they are just as we expected. As can be seen, the conferences that are ranked at the top are one of the most authoritative conferences in each area.

Different Paths.

Next, we want to explore how two different paths can produce different results. The task is to find papers that are similar to a given paper. We select two different schema paths. The first path helps to find the papers whose titles are similar to the query paper: `(PAPER)` → `(PAPER:title)` → `(term)` → `(PAPER:title)` → `(PAPER)`[8]. The other path allows the ranking model to find the papers that share the authors with the query paper: `(PAPER)` → `(PAPERAUTHOR)` → `(PAPERAUTHOR:author)` → `(PAPERAUTHOR)` → `(PAPER)`. The paper titled 'DISCOVER: Keyword Search in Relational . . . ' [14] is used as a query. The result is shown in Table 4. As we can see, the semantic meanings of these two results are different. If we compare the papers at rank 2, the result

[8]The edge types are omitted.

from the second path 'ObjectRank...' seems more similar in terms of the content of the paper. However, we would not be able to say which result is always better. It depends on the application and the query user's information needs.

Combining Paths for Recommendation.

Lastly, we combine the results from two different paths for personalized recommendation. The scenario is that, given a specific user and current date, the system recommends the musical artists based on the user's listening log, and daily popularity inferred from the log. The schema paths we used are '`(USER)` → `(LISTENLOG)` → `(TRACK)` → `(LISTENLOG)` → `(USER)` → `(LISTENLOG)` → `(TRACK)` → `(ARTIST)`' and '`(date)` → `(LISTENLOG:listendatetime)` → `(LISTENLOG)` → `(TRACK)` → `(ARTIST)`'. The first path finds similar users using the log as in *collaborative filtering* [2], then recommends artists. The second path guides the system to recommend artists who are popular on that particular date, which can be thought of as a simple version of context-aware recommendation [3, 15]. The results are shown in Table 5. Since the query user likes hip-hop, some hip-hop artists (rank 1, 4, 5) are recommended. In addition, some local artists (rank 2, 3) are recommended based on the daily popularity. Although it is very simple combination of the results, it is possible to combine the paths in more various ways.

7. DISCUSSION

As on-going research, we present several possible directions for future research.

7.1 Learning

The ranking model could be improved by adopting some learning techniques. Firstly, instead of assigning the weights of candidate paths manually, the system can learn the importance of each path for the specific task [20].

If we have a gold standard for the task, the weights could be automatically found by analyzing the relationships between features. A variety of learning-to-rank algorithms [21] can be adopted as a learning model. In addition, the type of target objects could be determined automatically without being specified by the user. Adopting the techniques used in other related areas would be helpful. The work in *keyword search in database* determines the target objects by analyzing the candidate subgraphs [9]. In desktop search, Kim *et al.* [17] studied on type prediction; given a query, it determines which type of objects (*e.g.* email, presentations) should be retrieved.

7.2 Efficiency

Although we have not discussed any efficiency issues, several performance problems will arise when the size of data becomes large. *Materializing* the results for some paths that are queried frequently could be one of the solutions for this issue. Instead of propagating the sequence of edges every time the user inquires, we can save the results in advance [25]. *Filtering* is another technique that can be used to improve the performance. When we propagate through edges, the nodes whose scores are too low could be removed based on threshold [19]. Since we only have interest in the top part of the ranked result, filtering does not largely affect the final result. We also used this filtering strategy for our experiments.

7.3 Flexible Querying Framework

To increase the expressiveness of the query, we can provide a variety options to the users by developing a flexible query language or querying framework [29]. For example, the framework may allow users to use operators or conditional clauses in the query, choose the scoring functions, and define more complex schema paths that are not just the sequence of edges as we defined. Moreover, the model could be improved by allowing users to interact with results. For instance, the users could provide feedback to the system. As search engines use the technique called *relevance feedback*, our path-based ranking method would allow the users to change the weights of paths if they think that the weights are inappropriate.

7.4 Result Presentation

The path-based ranking method enables the creation of an explanation about why the object is included in the ranked result. As hundreds of features are utilized in the ranking model of the search engines and recommender systems, users are more likely to have doubts why some objects are ranked higher. Explanation has been recognized as a solution for this problem [30, 28]. For example, a music recommender system gives an explanation of why the song is recommended, such as "Your friend likes this song, but since you have not listened to it, would you like to try it?". We could generate this kind of explanation automatically by using the information of the schema paths, such as the node types included in the paths and the weight of each path.

8. CONCLUSIONS

In this paper, we propose a method for ranking objects in an entity-relationship graph. We define a schema path which implies the semantics of the heterogeneous data. It gives a guideline to follow the graph. By walking through the graph with the schema paths, the system ranks objects for search or recommendation tasks.

It has recently been recognized that the path in the graph is a useful feature for analyzing complex data [20, 25, 29, 11]. However, there is still much room for improvement, and we hope that more research will be conducted on the path-based ranking model for search and recommendation.

9. ACKNOWLEDGMENTS

We would like to thank Neowiz Bugs Corporation who provided us with the dataset of their music streaming service for this research.

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20110017480).

10. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In *Electronic Commerce*, volume 2232 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2001.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(6):734 – 749, 2005.

[3] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*. Springer, 2011.

[4] O. Alonso, M. Gertz, and R. Baeza-Yates. Clustering and exploring search results using timeline constructions. In *CIKM '09: Proceeding of the 18th ACM Conf. on Information and Knowledge Management*, pages 97–106. ACM, 2009.

[5] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: authority-based keyword search in databases. In *VLDB '04: Proceedings of the 30th Int'l Conf. on Very Large Data Bases*, pages 564–575. VLDB Endowment, 2004.

[6] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *WWW '08: Proceedings of the 17th Int'l Conf. on World Wide Web*, pages 895–904. ACM, 2008.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW '98: Proceedings of the 7th Int'l Conf. on World Wide Web*, 1998.

[8] S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int'l Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.

[9] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD '09: Proceedings of the 35th SIGMOD Int'l Conf. on Management of Data*, pages 1005–1010. ACM, 2009.

[10] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, 2009.

[11] V. Dritsou, P. Constantopoulos, A. Deligiannakis, and Y. Kotidis. Optimizing query shortcuts in rdf databases. In *ESWC '11: Proceedings of the 8th Extended Semantic Web Conf.*, volume 6644 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2011.

[12] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. Language-model-based ranking for queries on rdf-graphs. In *CIKM '09: Proceeding of the 18th ACM Conf. on Information and Knowledge Management*, pages 977–986. ACM, 2009.

[13] J. Han, Y. Sun, X. Yan, and P. Yu. Mining knowledge from databases: an information network analysis approach. In *SIGMOD '10: Proceedings of the 36th ACM SIGMOD Int'l Conf. on Management of Data*, pages 1251–1252. ACM, 2010.

[14] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB '02: Proceedings of the 28th Int'l Conf. on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.

[15] M. Kahng, S. Lee, and S.-g. Lee. Ranking in context-aware recommender systems. In *WWW '11: Proceedings of the 20th Int'l Conf. Companion on World Wide Web*, pages 65–66. ACM, 2011.

[16] G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: searching and ranking knowledge. In *ICDE '08: Proceedings of the 24th IEEE Int'l Conf. on Data Engineering*, pages 953–962. IEEE, 2008.

[17] J. Kim and W. B. Croft. Ranking using multiple document types in desktop search. In *SIGIR '10: Proceeding of the 33rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 50–57. ACM, 2010.

[18] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. Flexrecs: expressing and combining flexible recommendations. In *SIGMOD '09: Proceedings of the 35th SIGMOD Int'l Conf. on Management of Data*, pages 745–758. ACM, 2009.

[19] N. Lao and W. W. Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *KDD '10: Proceedings of the 16th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 881–888. ACM, 2010.

[20] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81:53–67, 2010.

[21] T.-Y. Liu. Learning to rank for information retrieval. *Founddations and Trends in Infomation Retrieval*, 3:225–331, 2009.

[22] J. Park and S.-g. Lee. Keyword search in relational databases. *Knowledge and Information Systems*, 26:175–193, 2011.

[23] S. Rendle. *Context-Aware Ranking with Factorization Models*, volume 330 of *Studies in Computational Intelligence*. Springer, 2010.

[24] J. F. Sequeda, M. Arenas, and D. Miranker. On directly mapping relational databases to rdf and owl (extended version). *Technical Report TR-11-28*, 2011.

[25] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(7), 2011.

[26] Y. Tian, R. Hankins, and J. Patel. Efficient aggregation for graph summarization. In *SIGMOD '08: Proceedings of the 34th ACM SIGMOD Int'l Conf. on Management of Data*, pages 567–580. ACM, 2008.

[27] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE '09: Proceedings of the 25th IEEE Int'l Conf. on Data Engineering*, pages 405–416. IEEE, 2009.

[28] R. Varadarajan, V. Hristidis, and L. Raschid. Explaining and reformulating authority flow queries. In *ICDE '08: Proceedings of the 24th IEEE Int'l Conf. on Data Engineering*, pages 883–892. IEEE, 2008.

[29] R. Varadarajan, V. Hristidis, L. Raschid, M.-E. Vidal, L. Ibáñez, and H. Rodríguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT '09: Proceedings of the 12th Int'l Conf. on Extending Database Technology: Advances in Database Technology*, pages 553–564. ACM, 2009.

[30] C. Yu, L. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *ICDE '09: Proceedings of the 25th IEEE Int'l Conf. on Data Engineering*, pages 1299–1302. IEEE, 2009.