Chang LJ, Yu J, Qin L. Context-sensitive document ranking. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(3): 444–457 May 2010

Context-Sensitive Document Ranking

Li-Jun Chang (常利军), Jeffrey Xu Yu (于　旭), and Lu Qin (秦　璐)

Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong Hong Kong, China

E-mail: {ljchang, yu, lqin}@se.cuhk.edu.hk

Received November 3, 2009; revised December 12, 2009.

Abstract Ranking is a main research issue in IR-styled keyword search over a set of documents. In this paper, we study a new keyword search problem, called context-sensitive document ranking, which is to rank documents with an additional context that provides additional information about the application domain where the documents are to be searched and ranked. The work is motivated by the fact that additional information associated with the documents can possibly assist users to find more relevant documents when they are unable to find the needed documents from the documents alone. In this paper, a context is a multi-attribute graph, which can represent any information maintained in a relational database, where multi-attribute nodes represent tuples, and edges represent primary key and foreign key references among nodes. The context-sensitive ranking is related to several research issues, how to score documents, how to evaluate the additional information obtained in the context that may contribute to the document ranking, how to rank the documents by combining the scores/costs from the documents and the context. More importantly, the relationships between documents and the information stored in a relational database may be uncertain, because they are from different data sources and the relationships are determined systematically using similarity match which causes uncertainty. In this paper, we concentrate ourselves on these research issues, and provide our solution on how to rank the documents in a context where there exist uncertainty between the documents and the context. We confirm the effectiveness of our approaches by conducting extensive experimental studies using real datasets. We present our findings in this paper.

Keywords document ranking, uncertain ranking, structure cost, similarity

1 Introduction

Keyword search is an important issue in the information retrieval area[1] and has been widely and extensively studied over decades. Most of the existing keyword search approaches return a set of documents with ranking that is most important and relevant to a set of user-specified keywords. However, in many real applications, the documents may not be the only data source, and there are other forms of interrelated data for the same application domain. In this paper, we study a new ranking problem that is to rank documents, in particular, when users cannot find any documents that contain all the given keywords from the set of documents itself, or when users can possibly find more relevant documents by exploring the documents as well as the interrelated data. In other words, we consider how to effectively use the additional relevant information, that may be maintained in a database, to rank documents. We explain our motivation using an example.

Fig.1 shows a collection of reviews (documents) on the left side about movies. There are three movie reviews with review titles, namely, "Ocean's Twelve", "Seven", and "Twelve Monkeys". In the movie review entitled "Ocean's Twelve", it wrote "···those things make the film truly great···". These movie reviews are about movies which can be maintained as tuples in a multi-attribute table, consisting of the movie title and other possible attribute values as illustrated in Fig.1 on

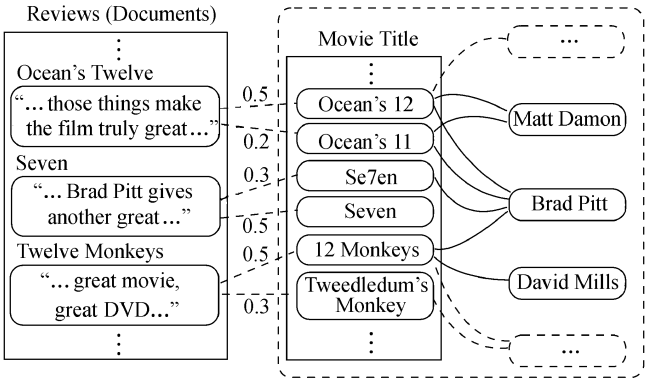

Fig.1. Movie reviews and movie records.

Regular Paper

This work was supported by the Research Grants Council of the Hong Kong SAR, China, under Grant Nos. 419008 and 419109.

the right side (refer to IMDB `http://www.imdb.com/interfaces`). The names of directors or actors or actresses may not appear in the movie reviews or even in the movie title, but they appear in some tuples in some tables in the IMDB movie database that are linked to the movie titles via relationships such as movies directed by directors, and movies played by actors/actresses, based on the primary keys and foreign key references. We call such interrelated information maintained in multi-attribute tables a context to which the movie review is directly/indirectly related. The relationship between a movie review (document) and a movie title (or simply movie) maintained in a multi-attribute table is with uncertainty. Such uncertainty may be caused by missing titles, or mismatching of the discussions in the body of movie review, or different movies with the same title. For example, the movie review entitled "Ocean's Twelve" may not exactly match the movie title "Ocean's 12" maintained in the multi-attribute table. In Fig.1, there is a link between a movie review and a movie in a table, where such uncertainty is indicated as a probability.

Suppose a user wants to know something "great" about the actor "Brad Pitt", by issuing a 2-keyword query, "great" and "Brad Pitt", against the collection of movie reviews. The movie review entitled "Seven" will have the highest score, followed by movie review on "Twelve Monkeys" and "Ocean's Twelve". Suppose that the user issues a keyword query with two additional keywords, "Matt Damon" and "David Mills", the movie reviews and their rank will remain the same, since none of the reviews contain the additional two keywords. As observed from Fig.1, there is information about "Matt Damon" and/or "David Mills" indirectly linked to the movies and the movie reviews, as a part of structural information, which are not effectively used to answer users' keyword queries. Furthermore, assume a 2-keyword query is "great" and "David Mills". Without the context, there are no documents containing the two given keywords. With the context shown in Fig.1, it is highly possible to find movie reviews on movies the two keywords are most related.

The main issue we study in this paper is how to rank documents using a set of keywords, given a context that is associated with the documents. The main contributions of this paper are summarized below. First, we study a new ranking problem to rank documents for a given set of keywords. The uniqueness of the problem is that the documents to be ranked are associated with sets of interrelated multi-attribute tuples called context, which contains additional information that assist users to rank the relevant documents with some uncertainty. Second, we model the problem using a graph $\mathcal{G}$ with two different kinds of nodes (document nodes and multi-attribute nodes), and discuss its score function, cost function, and ranking with uncertainty. Third, we propose new algorithms to rank documents that are most related to the user-given keywords by integrating the context information. We also conducted extensive experimental studies to show the effectiveness of our approaches using a real dataset.

The remainder of this paper is organized as follows. Section 2 gives the problem statement. We discuss score functions and ranking in Section 3 and Section 4, respectively. In Section 5, we give an overall system architecture, and discuss how to rank with uncertainty. Experimental studies are given in Section 6, followed by discussions on related work in Section 7. Finally, we conclude the paper in Section 8.

2 Problem Statement

We consider a set of documents, $D_A = \{d_1, d_2, \ldots\}$. In addition, we assume there exists a context, $\mathcal{C}$, which is considered as a multi-attribute graph $G_R(V, E)$ that specifies the knowledge about documents. A multi-attribute graph G_R is capable of representing all the tuples in a relational database, where a node represents a tuple, and an edge between two nodes represents a foreign-key reference between the two corresponding tuples. All nodes (tuples) in the same relation are said to have the same type. There exists a set of specific A-typed nodes in $G_R(V, E)$, V_A ($\subseteq V$) which are explicitly linked to the documents in D_A. A document may be linked to several A-typed nodes, and an A-typed node may be related to several documents. To integrate the set of documents, D_A, and the multi-attribute graph, $G_R(V, E)$, we consider a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V} = V \cup D_A$, and $\mathcal{E} = E \cup E_D$ where E_D is a set of pairs, (d_i, v_j), if d_i is related to an A-typed node v_j. We call a node in D_A a document node, and a node in G_R a multi-attribute node. We assign edge-weights to edges in the subgraph $G_R(V, E)$ of $\mathcal{G}$ based on [2-3]. In detail, for a foreign key reference from u to v, the edge weight for (u, v) is given as (1), and the edge weight for (v, u) is given as (2).

$$w_e((u, v)) = 1 \tag{1}$$

$$w_e((v, u)) = \log_2(1 + N_{in}(v)) \tag{2}$$

where $N_{in}(v)$ is the number of nodes that refer to v. We assign the edges in E_D with a non-zero weight which is a normalized similarity score computed as given in [4]. We consider the weight assigned to $(d_i, v_j) \in E_D$ as the probability that d_i is related to v_j. A document node, d_i, may be related to

several different multi-attribute nodes, v_j and v_k, with probability distribution, $\mathsf{prob}(d_i, v_j)$ and $\mathsf{prob}(d_i, v_k)$, respectively. However, the probability must satisfy $\sum_{v_j} \mathsf{prob}(d_i, v_j) \leqslant 1$ in order to form a distribution.

Example 2.1. Fig.2(a) shows a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. There are two documents in $D_O = \{d_1, d_2\}$, and a multi-attribute graph $G_R(V, E)$ illustrated in the dashed rectangle. There are two types of nodes in V, namely `O` and `V`, where the former consists of $\{o_1, o_2, o_3\}$, and the latter consists of $\{v_1, v_2, v_3, v_4\}$. The edges between two nodes in G_R show how the multi-attribute nodes are interrelated. The edge (d_1, o_1) and (d_1, o_2) show that d_1 is related to o_1 and o_2 with probability distribution, 0.5 and 0.4, respectively. The edges (d_2, o_2) and (d_2, o_3) show that d_2 is related to o_2 and o_3 with probability distribution, 0.6 and 0.3, respectively. There are three distinct keywords appearing in $\mathcal{G}$: `a`, `b`, and `c`. The keyword `a` appears in multi-attribute nodes v_1 and v_2, the keyword `b` appears in the multi-attribute node v_4, and the keyword `c` appears in document nodes d_1 and d_2.

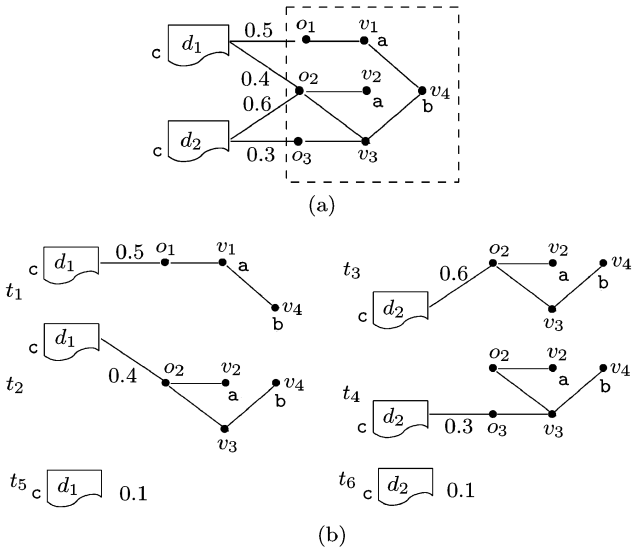

Fig.2. Example. (a) Sample graph. (b) Local contexts.

Problem Statement. Finding top-k documents for an l-keyword query, $Q = \{w_1, w_2, \ldots, w_l\}$, against graph $\mathcal{G}$.

The ranking is based on the following consideration. First, if a document node itself contains all the l-keywords, it should be ranked higher. If a document node, d_i, does not contain all the l-keywords, it is ranked based on d_i and its associated local context (which will be introduced next) that together contain all the l-keywords. Also, because a document d_i may be related to several local contexts that contain all the keywords individually, it needs to identify a local context that will make the document d_i ranked higher. The selection of such a local context is done by considering all possibilities. Below, we discuss the score/cost functions in Section 3 and ranking strategies in Section 4.

3 Score Function

Given an l-keyword query $Q = \{w_1, w_2, \ldots, w_l\}$, we first discuss a concept, called a local context, for a document node d_i in $\mathcal{G}$ to be possibly ranked. A local context is a connected tree in $\mathcal{G}$, denoted as $T(V, E)$. The local context contains one document node, d_i, that can also link with a connected rooted tree, $\Delta(V, E)$, as a subgraph in G_R. Note that $G_R \subset \mathcal{G}$. We say that the local context T supports the document d_i. A connected tree $\Delta(V, E)$ specifies an interrelated tuple structure among tuples that is related to the document node d_i. The root of $\Delta_j(V, E)$ is an A-typed node, say v_j, and the probability for d_i to link with v_j is $\mathsf{prob}(d_i, v_j)$. A keyword w is contained in $T(V, E)$, if it appears in the document node d_i or in any multi-attribute node in the corresponding $\Delta(V, E)$. A local context, $T(V, E)$, contains l-keywords based on an "or" semantics. In other words, a local context for a specific document node d_i may not contain all the required l-keywords. For example, a document node d_i may not link with any A-typed node in $\mathcal{G}$, or a document node d_i may link with an A-typed node, v_j, but there does not exist a connected tree $\Delta(V, E)$, rooted at v_j, containing the required keywords that do not appear in d_i. In such cases, the corresponding $T(V, E)$ does not contain all the required keywords. On the other hand, for a given document node d_i, there may exist more than one connected trees, for example, two trees Δ_j and Δ_k rooted at v_j and v_k. Each of the trees contains all or some of the required keywords that do not appear in d_i. We treat it as two local contexts. In order to select one of them as the local context for d_i, we define a score function, denoted as $score(T, Q)$.

In the following we use T and t_i as local contexts interchangeably. Here, T means a general local context, whereas t_i means a local context for a specific document node.

Example 3.1. Reconsider Fig.2(a). Given a 3-keyword query, $Q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, we show the six local contexts, t_i, for $1 \leqslant i \leqslant 6$, in Fig.2(b). Among them, the local contexts, t_1, t_2, and t_5, support document d_1, and the local contexts, t_3, t_4, and t_6, support document d_2. Assume $score(t_i, Q)$, for $1 \leqslant i \leqslant 6$, are the scores, which we will discuss in detail. For document node d_1, with probability 0.5 the score is $score(t_1, Q)$, with probability 0.4 the score is $score(t_2, Q)$, and with probability 0.1 the score is $score(t_5, Q)$.

As shown in Example 3.1, there are local contexts with only a document node d_i if $1 - \sum_{v_j} \mathsf{prob}(d_i, v_j) >$

0. For example, for document d_1, the local context t_5 is with probability $1 - 0.5 - 0.4 = 0.1$; for document d_2, the local context t_6 is with probability $1 - 0.6 - 0.3 = 0.1$. In the following, we denote the probability of a local context, t' for a document d_i as $\mathsf{prob}(t')$. It is equal to $\mathsf{prob}(d_i, v_j)$ if the edge (d_i, v_j) appears in the local context t', and it is $\mathsf{prob}(d_i) = 1 - \sum_{v_j} \mathsf{prob}(d_i, v_j)$ otherwise.

Score Function for Local Context. We discuss score functions to score a local context for a document below, and will discuss ranking documents in local contexts, in the next section, which is based on the scores and probabilities of having such scores.

Recall that a local context t' consists of a document node d_i which may link with a connected tree Δ_j rooted at a multi-attribute node v_j. There are two main components in our score function to score a local context t'. One is related to the keywords it contains. The other is how to evaluate the cost of using a connected tree Δ_j. The connected tree Δ_j is needed if it contains some keywords that are missing in the document node d_i. However, there is a cost to use such a connected tree Δ_j to include more keywords. It is because that there may be a Δ_j which is very relevant to d_i, and there may be a Δ_j which is not very relevant to d_i.

First, for the first component, we consider a local context T as a virtual document[5-6], and treat all local contexts as a supporting-set, where each local context can be scored using an IR-style relevance score, with respect to the l-keyword query $Q = \{w_1, w_2, \ldots, w_l\}$. It is defined below[6].

$$score_{\mathrm{IR}}(T, Q) = \sum_{w \in T \cap Q} \frac{1 + \ln(1 + \ln(tf_w(T)))}{(1 - s) + s \cdot \frac{dl_T}{avdl}} \cdot \ln(idf_w). \tag{3}$$

Here, $w \in T \cap Q$ indicates the subset of keywords of Q appearing in T. $tf_w(T)$ is the term frequency, e.g., the number of occurrences of the keyword w in the local context T; $idf_w = \frac{N+1}{df_w(Doc)}$, where $df_w(Doc)$ is the document frequency, e.g., the number of virtual documents that contain the keyword w and N is the total number of the virtual documents; $(1 - s) + s \cdot \frac{dl_T}{avdl}$ is the length normalization, where dl_T is the document length of the local context T, $avdl$ is the average document length among the whole virtual document collection, and s is a parameter. We also consider a completeness coefficient for $score_{\mathrm{IR}}(T, Q)$, denoted as $complete(T, Q)$, which specifies how important for a local context to contain a keyword. The more keywords contained the better.

$$complete(T, Q) = \frac{\sqrt[p]{\sum_{w_i \in Q} (I(w_i, T) \cdot \ln(idf_{w_i}))^p}}{\sqrt[p]{\sum_{w_i \in Q} (\ln(idf_{w_i}))^p}}. \tag{4}$$

Here, $I(w_i, T)$ is an indicator function, it equals to 1 if and only if local context T contains keyword w_i and 0 otherwise. The value of $complete(T, Q)$ is in the range $[0, 1]$. $complete(T, Q)$ is equal to 1 if all the keywords appear in T. The intuition behind the completeness function is that, if more (distinct) keywords appear in a local context, then it should be ranked higher; if the number of (distinct) keywords appeared is the same, then the more discriminative (larger $\ln(idf_w)$) the keyword it contains the higher it ranks.

Second, for the second component, we define it to be the total distance between the document node d_i and the nodes in Δ_j that contain the keywords in a local context T[7].

$$cost_{\mathrm{st}}(T, Q) = \sum_{w \in V(\Delta_j) \cap Q} dis(d_i, node(w)) \tag{5}$$

where $V(\Delta_j)$ denotes the set of nodes in T that appear in Δ_j, $node(w)$ denotes the node in Δ_j that contains the keyword w, and $dis(a, b)$ is the distance between two nodes in the local context based on the edge weights.

Combining the two components, the score of a local context, T, with respect to a query $Q = \{w_1, w_2, \ldots, w_l\}$ is given as follows.

$$score(T, Q) = \alpha \cdot complete(T, Q) \cdot score_{\mathrm{IR}}(T, Q) - (1 - \alpha) \cdot cost_{\mathrm{st}}(T, Q) \tag{6}$$

where α is a parameter in the range of $[0, 1]$, that specifies the relative importance of the two factors. A local context, T, is ranked higher if its score $score(T, Q)$ is larger, which implies that either its textual component score $score_{\mathrm{IR}}(T, Q)$ is larger or its structural component cost $cost_{\mathrm{st}}(T, Q)$ is smaller. In our formulation, a local context does not necessarily include all the l-keywords, even though more keywords is demanded to increase the completeness factor $complete(T, Q)$. As indicated in (6), if we include one more node in a local context that contains a keyword, it increases the IR-styled score (see (3)) and completeness factor (see (4)), but it also increases the cost (see (5)). Increasing the IR-styled score or completeness factor implies that the local context is more related to the query Q, which makes the final score (see (6)) larger (higher rank). Increasing the cost implies that the local context (connected tree) becomes larger to include more multi-attribute nodes (or keywords), which makes the final score (see (6)) smaller (lower rank).

4 Ranking with Uncertainty

In the previous section, we discuss our score function to score a local context. In this section, we concentrate on ranking. We adopt ranking with uncertainty

probability as discussed in [8-10], where x-Relation model is used. In the x-Relation model[9,11], there is a set of independent x-tuples (called generation rules in [8, 10]), where an x-tuple consists of a set of mutually exclusive tuples (called alternatives), represents a discrete probability distribution of the possible tuples it may take in a randomly instantiated data. Each alternative t has a score and a probability, where $Pr(t)$ represents its existence probability over possible instances. In the x-Relation model, the alternatives of x-tuples are assumed to be disjoint.

d_i	o_j	$\mathsf{prob}(d_i, o_j)$	t_x
d_1	o_1	0.5	t_1
	o_2	0.4	t_2
	–	0.1	t_5
d_2	o_2	0.6	t_3
	o_3	0.3	t_4
	–	0.1	t_6

(a)

Possible World (I)	$Pr(I)$
$\{t_1, t_3\}$	0.30
$\{t_1, t_4\}$	0.15
$\{t_1, t_6\}$	0.05
$\{t_2, t_3\}$	0.24
$\{t_2, t_4\}$	0.12
$\{t_2, t_6\}$	0.04
$\{t_3, t_5\}$	0.06
$\{t_4, t_5\}$	0.03
$\{t_5, t_6\}$	0.01

(b)

Fig.3. x-Relation model. (a) x-Relation. (b) Possible worlds.

Example 4.1. Fig.3(a) is an x-Relation, consists of two x-tuples, $\tau_1 = \{t_1(0.5), t_2(0.4), t_5(0.1)\}$, and $\tau_2 = \{t_3(0.6), t_4(0.3), t_6(0.1)\}$. The x-tuple τ_1 denotes a probability distribution over t_1, t_2 and t_5, with probability 0.5 it takes t_1, with probability 0.4 takes t_2, and with probability 0.1 takes t_5.

The x-Relation, $\mathcal{X}$, is a probability distribution over a set of possible instances $\{I_1, \ldots, I_n\}$. A possible instance, I_j, maintains zero or one alternative for each x-tuple $\tau \in \mathcal{X}$. The probability of an instance I_j, is the probability that x-tuples makes exactly the choice of alternatives as in I_j, specifically, $Pr(I_j) = \prod_{t \in I_j} Pr(t) \times \prod_{\tau \notin I_j} (1 - Pr(\tau))$, where $\tau \notin I_j$ means x-tuple τ takes no alternative in I_j, $Pr(\tau) = \sum_{t \in \tau} Pr(t)$. The entire possible worlds of an x-Relation, $\mathcal{X}$, denoted as $pwd(\mathcal{X})$, is the set of all the subset of $I \subseteq \mathcal{X}$ with probability greater than 0 ($Pr(I) > 0$).

In our problem setting, the local contexts confirm the x-Relation model. The set of local contexts that support the same document forms an x-tuple, and an x-tuple specifies that the local contexts belong to it are mutually exclusive. Intuitively, an x-tuple supports one document with mutually exclusive evidences. And the local contexts that support different documents are independent.

Example 4.2. For the graph in Fig.2(a), there are six local contexts $\{t_1, t_2, t_3, t_4, t_5, t_6\}$, as shown in Fig.2(b). The score distribution for the document d_1 is $\{(score(t_1), 0.5), (score(t_2), 0.4), (score(t_5), 0.1)\}$, because $\{t_1, t_2, t_5\}$ is the set of local contexts that support the document d_1. The score distribution for the document d_2 is $\{(score(t_3), 0.6), (score(t_4), 0.3), (score(t_6), 0.1)\}$, because $\{t_3, t_4, t_6\}$ is the set of local contexts that support the document d_2. Here, for example, t_1 and t_2 are mutually exclusive because they are about the same document d_1, and t_1 and t_3 are independent because one is about the document d_1 and the other is about the document d_2. Fig.3(a) shows the six possible local contexts. Here, $\mathsf{prob}(d_i, o_j)$ is the probability of the local context, or in other words, the probability that d_i discusses about o_j. The 9 possible worlds, with non-zero probability, are shown in Fig.3(b), together with their probabilities. The possible world $\{t_5, t_6\}$ means that, with probability 0.01, the two document nodes, d_1 and d_2 are not related to any of the three multi-attribute nodes, o_1, o_2, and o_3. The possible world $\{t_1, t_3\}$ means that, with probability 0.3, d_1 is related to o_1, and d_2 is related to o_2. Note that the sum of the probabilities of all the possible worlds is equal to 1.

In our problem setting, an x-tuple specifies a probability distribution on the score for a document. In each possible world, it maintains exactly one local context from each x-tuple, then the size of a possible world will be exactly the size of D_A, i.e., $|I| = |D_A|$ for $\forall I \in pwd(\mathcal{X})$. And each possible world corresponds to a possible linkage between a document node in D_A and an A-typed multi-attributed node in G_R.

Several top-k definitions in the x-Relation model have been proposed recently based on the possible worlds semantics[8-10,12-14]. We will study possible world semantics based uncertain ranking in the next subsections, specifically, top-k probability in Subsection 4.1 and expected rank in Subsection 4.2.

Before we discuss the top-k probability and the expected rank, we first study a straightforward strategy of getting score for documents deterministically which is not based on the possible world semantics. As we have shown, an x-tuple specifies a probability distribution on the score for documents. It is natural to define the score of a document as the expected score of the local contexts that support it. The expected score of a document is defined as follows,

$$EScore(d_i) = \sum_{T: doc(T) = d_i} score(T, Q) \cdot \mathsf{prob}(T) \quad (7)$$

where $score(T, Q)$ is the score of local context T against query Q (refer to (6)), and $\mathsf{prob}(T)$ is the probability of local context T. Recall that, $\mathsf{prob}(T) = \mathsf{prob}(d_i, v_j)$ if (d_i, v_j) appears as an edge in T, and $\mathsf{prob}(T) = 1 - \sum_{v_j} \mathsf{prob}(d_i, v_j)$ if T consists of a single node d_i.

It is natural to use the expected score to rank documents. However, one drawback of the expected score is that the expected score is sensitive to the score values, high score value with low probability can result in high expected score. The possible worlds based uncertain ranking is insensitive to the exact score values, and it only depends on the relative order among the local contexts.

Example 4.3. For the graph $\mathcal{G}$ in Fig.2(a) and a query $Q = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, there are six local contexts as shown in Fig.3(a) which supports the two documents. The expected score of d_1 and d_2 are, $EScore(d_1) = 0.5 \cdot score(t_1, Q) + 0.4 \cdot score(t_2, Q) + 0.1 \cdot score(t_5, Q)$, $EScore(d_2) = 0.6 \cdot score(t_3, Q) + 0.3 \cdot score(t_4, Q) + 0.1 \cdot score(t_6, Q)$.

4.1 Top-k Probability

We introduce the top-k probability, as discussed in [10]. The top-k probability of a local context, T, denoted as $tkp(T)$, is the marginal probability that it ranks top-k in the possible worlds, given as below.

$$tkp(T) = \sum_{I \in pwd(\mathcal{X}), T \in topk(I)} Pr(I) \tag{8}$$

where $T \in topk(I)$ means that the local context is ranked as one of the top-k local contexts in the instance I. And the top-k probability of a document is defined as below.

$$tkp(d_i) = \sum_{T: doc(T) = d_i} tkp(T). \tag{9}$$

The top-k probability of a document is the summation of the top-k probability of the local contexts that support the document. Note that $\sum_{T:doc(T)=d_i} \mathsf{prob}(T) = 1$ and the $\mathsf{prob}(T)$ has been considered in $tkp(T)$.

Example 4.4. Consider the six local contexts in Fig.3(a), assume the local contexts in decreasing score order are $t_1, t_3, t_2, t_4, t_6, t_5$. That is document d_2 in the local context t_6 has a larger score than d_1 in the local context t_5, $\{t_1, t_2, t_3, t_4\}$ have a larger score than $\{t_5, t_6\}$. Among $\{t_1, t_2, t_3, t_4\}$, t_1 is the most compact, then it has the largest score.

Now, we consider the top-1 probability. $tkp(t_1) = 0.30 + 0.15 + 0.05 = 0.5$, which is the summation of the probability of the first three possible worlds in Fig.3(b). $tkp(t_2) = 0.12 + 0.04 = 0.16$. Note that the local context t_2 ranks the second place in the possible world $\{t_2, t_3\}$. $tkp(t_5) = 0$, because it ranks the first place in no possible world. Therefore, the top-1 probability of the document d_1 is $tkp(d_1) = tkp(t_1) + tkp(t_2) + tkp(t_5) = 0.66$. Similarly, we can get $tkp(d_2) = 0.34$. Comparing $tkp(d_1)$ and $tkp(d_2)$, we can see that document d_1 is more likely to be ranked higher than d_2. It is interesting to note that, without the multi-attribute graph G_R, d_2 is ranked higher than d_1, because we assume that the local context t_6 ranks higher than t_5. Also, consider the expected score (see (7)), then both ranking orders are possible, because it is sensitive to the actually score values, and different score values with the same relative order will result in different ranking orders.

A naive computation of top-k probability needs to enumerate all the possible worlds, which is exponentially many. We show the approach to compute the top-k probability for local contexts efficiently without enumerating possible worlds[9-10], and we concentrate on how to compute $tkp(T)$ for a specific local context T. We assume that whenever we talk about an x-tuple we do not consider the x-tuple contains T, i.e., we only take the x-tuples τ with "$T \notin \tau$" into consideration. Consider an arbitrary x-tuple τ, assume there are two local contexts T_1 and T_2 in it with a score lager than $score(T, Q)$, then we replace these two local contexts by a virtual local context t with probability $\mathsf{prob}(T_1) + \mathsf{prob}(T_2)$ and with an arbitrary score larger than $score(T, Q)$, and the virtual local context t also belongs to the x-tuple τ. This transformation is equivalent to merge the possible worlds that only differ between T_1 and T_2 into one possible world. It is easy to see that $tkp(T)$ is the same if it is computed on the modified and simpler x-Relation. This transformation is also true for any two local context in an x-tuple with a smaller score than $score(T, Q)$. After recursively applying these two transformations, each x-tuple consists of exactly two virtual local contexts, t^l (large virtual local context) and t^s (small virtual local context), with probability

$$\mathsf{prob}(t^l) = \sum_{T_i \in \tau, score(T_i, Q) > score(T, Q)} \mathsf{prob}(T_i)$$

$$\mathsf{prob}(t^s) = \sum_{T_i \in \tau, score(T_i, Q) \leqslant score(T, Q)} \mathsf{prob}(T_i).$$

Note that in our problem setting, $\mathsf{prob}(t^l) + \mathsf{prob}(t^s) = 1$, because $\sum_{T_i \in \tau} \mathsf{prob}(T_i) = 1$. Then in each possible world that contains T, for an x-tuple $\tau = \{t^l, t^s\}$, it contains either t^l or t^s. $tkp(T)$ is equal to the summation of the probability of the possible worlds that contain less than k large virtual local contexts, this can be done efficiently using dynamic programming. Assume $\tau_1, \ldots, \tau_n$ are the x-tuples in $\mathcal{X}$ in arbitrarily order excluding the x-tuple contains T. Let t_i^l and t_i^s denote the two virtual local contexts in τ_i. Let $r_{i,j}$ be a variable which means the probability that a random possible world generated from $\{\tau_1, \ldots, \tau_i\}$ has exactly j large virtual

local contexts. Then $tkp(T)$ equals $\mathsf{prob}(T) \cdot \sum_{j=0}^{k-1} r_{n,j}$. All $r_{i,j}$ values can be computed by the following dynamic programming equation.

$$r_{i,j} = \begin{cases} \mathsf{prob}(t_i^l) \cdot r_{i-1,j-1} + (1 - \mathsf{prob}(t_i^l)) \cdot r_{i-1,j}, & \text{if } i \geqslant j > 0, \\ (1 - \mathsf{prob}(t_i^l)) \cdot r_{i-1,j}, & \text{if } i > j = 0, \\ 1, & \text{if } i = j = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Example 4.5. For the example x-Relation given in Fig.4(a), assume the local contexts $\{t_1, \ldots, t_{11}\}$ are in decreasing score order, the probability of each tuple is specified in the parentheses. The execution steps of computing top-2 probability of t_7 is as follows.

We first transform the x-Relation in Fig.4(a) according to t_7 into that in Fig.4(b) using the two transformation rules. Then we need to compute $r_{i,0}$ and $r_{i,1}$ for $i = 0, 1, 2, 3$. Initially, $r_{0,0} = 1$ and $r_{0,1} = 0$. $\mathsf{prob}(t_1^l) = 0.7$, using (10), we can get $r_{1,0} = 0.3$ and $r_{1,1} = 0.7$. Similarly, $\mathsf{prob}(t_2^l) = 0.5$, we can get $r_{2,0} = 0.15$ and $r_{2,1} = 0.5$. Because $\mathsf{prob}(t_3^l) = 1$, $r_{3,0} = 0$ and $r_{3,1} = 0.15$. So the top-2 probability t_7 is equal to $tkp(t_7) = \mathsf{prob}(t_7) \cdot (r_{3,0} + r_{3,1}) = 0.3 \cdot 0.15 = 0.045$.

τ_1	$\{t_1(0.3), t_4(0.4), t_{10}(0.3)\}$
τ_2	$\{t_2(0.5), t_8(0.2), t_{11}(0.3)\}$
τ_3	$\{t_3(0.5), t_6(0.5)\}$
τ_4	$\{t_5(0.6), t_7(0.3), t_9(0.1)\}$

(a)

τ_1	$\{t_1^l(0.7), t_1^s(0.3)\}$
τ_2	$\{t_2^l(0.5), t_2^s(0.5)\}$
τ_3	$\{t_3^l(1.0), t_3^s(0)\}$
τ_4	–

(b)

Fig.4. Example x-Relation. (a) x-Relation. (b) To compute $tkp(t_7)$.

4.2 Expected Rank

In this subsection, we discuss the expected rank semantics[13]. Although the top-k probability proposes one solution to rank data with uncertain scores, one drawback is that the top-k results may not be included in the top-$(k+1)$ results[13], i.e., there exist some results belonging to top-k results but not to top-$(k+1)$ results.

The rank of a local context T in a possible world I is defined to be the number of local contexts whose score is lager than $score(T, Q)$ (so the top local context has rank 0), i.e., $rank_I(T) = |\{T_i \in I \mid score(T_i, Q) > score(T, Q)\}|$. The expected rank of a local context is defined as follows.

$$ERank(T) = \sum_{I \in pwd(\mathcal{X}), T \in I} rank_I(T) \cdot Pr(I). \tag{11}$$

The expected rank of a local context is only defined on the possible worlds that contain it. The expected rank of a document is defined as below.

$$ERank(d_i) = \sum_{T: doc(T) = d_i} ERank(T). \tag{12}$$

The expected rank of a document is the summation of the expected rank values of the local contexts that support the document. It means the expected rank position for the document d_i in a randomly generated possible world. Note that the smaller the value $ERank(d_i)$ the higher it ranks.

Example 4.6. Following Example 4.4, consider the six local contexts in Fig.3(a), assume the local contexts in decreasing score order are $t_1, t_3, t_2, t_4, t_6, t_5$. We consider the expected rank. $ERank(t_1) = 0$, because it ranks at the first place in the first three possible worlds in Fig.3(b). $ERank(t_2) = 0.24$, because it ranks at the second place in the possible world $\{t_2, t_3\}$ and ranks at the first place in other possible worlds. $ERank(t_5) = 0.06 + 0.03 + 0.01 = 0.1$, because it ranks the second place in the last three possible worlds. So the expected rank value of document d_1 is $ERank(d_1) = ERank(t_1) + ERank(t_2) + ERank(t_5) = 0.34$. Similarly, we can get $ERank(d_2) = 0.66$. Comparing $ERank(d_1)$ and $ERank(d_2)$, we can see that the document d_1 ranks higher than d_2 based on the expected rank semantics. Note that the smaller the expected rank value the higher it ranks. The ranking is the same of that based on top-1 probability in Example 4.4.

A naive computation of expected rank values needs to enumerate all the possible worlds, which is exponentially many. We present the approach to compute the expected rank value for local contexts efficiently without enumerating possible worlds[13], and we concentrate on how to compute $ERank(T)$ for a specific local context T. It is easy to verify that the two transformation rules proposed in Subsection 4.1 is valid in the expected rank semantics. Assume $\tau_1, \ldots, \tau_n$ are the x-tuples in $\mathcal{X}$ in arbitrarily order excluding the x-tuple contains T. Let t_i^l and t_i^s denote the two virtual local contexts in τ_i. Then in each possible world that contains T, for an x-tuple $\tau = \{t^l, t^s\}$, it contains either t^l or t^s. If it contains t^l then the expected rank value of T increases by one; if it contains t^s the expected rank value of T does not change. Then, the expected rank value of T is equal to the existence probability of T times the expected number of large virtual local contexts in a randomly generated possible world, i.e., $ERank(T) = \mathsf{prob}(T) \cdot \sum_{i=1}^{n} \mathsf{prob}(t_i^l)$.

Example 4.7. For the example x-Relation given in Fig.4(a), assume the local contexts $\{t_1, \ldots, t_{11}\}$ are in decreasing score order, the probability of each tuple is specified in the parentheses. The execution steps of computing expected rank value of t_7 is as follows. We

first transform the x-Relation in Fig.4(a) according to t_7 into that in Fig.4(b) using the two transformation rules. Then $ERank(t_7) = \mathsf{prob}(t_7) \cdot (\mathsf{prob}(t_1^l) + \mathsf{prob}(t_2^l) + \mathsf{prob}(t_3^l)) = 0.3 \cdot 2.2 = 0.66$.

5 Query Processing

Given an l-keyword query, Q, against a graph $\mathcal{G}$, we first generate the set of local contexts, $\mathcal{T} = \{t_1, t_2, \ldots\}$, where a local context t_x, scored $score(t_x, Q)$, contains a unique document node, d_i, and has a probability $\mathsf{prob}(T)$, which means that the local context t_x supports that document d_i has a score $score(t_x, Q)$ with probability $\mathsf{prob}(T)$. If the document node d_i is an isolated node in $\mathcal{G}$, then there is a local context in $\mathcal{T}$ that contains only one node d_i, with probability one. Each local context supports one document, given the set of local contexts, the score of documents is a probability distribution. After getting the local contexts, documents are ranked based on the probability score distribution. We integrate the information to rank top-k documents.

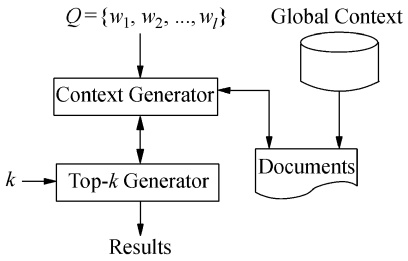

Fig.5. System architecture.

An overview of the system architecture is shown in Fig.5. A user submits an l-keyword query $Q = \{w_1, w_2, \ldots, w_l\}$ and k, and rank the top-k documents under a global context G_R. As a preprocess step, it constructs the graph $\mathcal{G}$ by linking each document $d_i \in D_A$ with several high similar A-typed multi-attribute nodes in G_R. There are two main components in the system, namely, *Local Context Generator* (ConGen) and *Top-k Generator* (TopKGen). Here, ConGen generates the local contexts incrementally in the decreasing order of the score function (see (6)) for a user-given l-keyword query $Q = \{w_1, w_2, \ldots, w_l\}$. It is important to emphasize that ConGen generates the local contexts one by one in the decreasing order of the score function, which means that the local context ranked higher will be generated earlier. TopKGen will process the local contexts in the decreasing score order by iteratively calling ConGen to obtain a generated local context in one iteration.

5.1 Local Context Generation

A naive way to compute a local context is to start from a pair of a document node and the corresponding A-typed multi-attribute node, (d_i, v_j), with $\mathsf{prob}(d_i, v_j) > 0$, extend the local context by finding the shortest distance from the node v_j to a node containing a keyword in $\mathcal{G}$. A node containing a keyword will be included as a part of the local context if it increases the score (see (6)) or in other words makes the rank of the local context higher. All the local contexts are then sorted in the decreasing order and returned to TopKGen one by one in every iteration on demand.

We propose an incremental algorithm, which generates the local contexts incrementally one by one in the decreasing order of the score, and we do not need to compute all the local contexts. There are two main issues. First, the number of possible pairs, (d_i, v_j), in graph $\mathcal{G}$, can be very large. Second, graph $\mathcal{G}$ itself can be very large.

For the first issue, we do not need to compute all the possible local contexts. A local context containing (d_i, v_j) is upper bounded by the following upper bounded score,

$$\alpha \cdot \sum_{w \in Q} \frac{1 + \ln(1 + \ln(tf_w(d_i) + 1))}{(1 - s) + s \cdot \frac{dl_T}{avdl}} \cdot \ln(idf_w). \quad (13)$$

The upper bound is given based on the following reason. When a keyword is newly included in a local context, it will increase the term frequency by one, and also increase the structural cost non-negative. In the ideal situation, the multi-attribute node v_j contains all the keywords in the l-keyword Q, the local context score of (d_i, v_j) will be exactly as (13). If the upper bound score (see (13)) is smaller than the best score of the currently computed local contexts (refer to (6)), then the next computed local context with the highest score will be taken from the currently maintained local contexts. For the second issue, when computing a local context containing (d_i, v_j), we find the nearest multi-attribute node that contains a certain keyword, which takes time when graph $\mathcal{G}$ is large in size. It is impractical to compute all shortest distances between a document and a multi-attribute node containing a keyword. Instead, for a keyword, w, we compute the shortest distance to every multi-attribute node, starting from those nodes that contain the keyword, w. The shortest distance is computed to the extent that it increases the score of the local context, or it can also be computed on demand. We only need to scan the graph G_R of graph $\mathcal{G}$, once for a keyword by starting from a virtual node that connects to all nodes containing the keyword.

The incremental algorithm is outlined in Algorithm 1. Here, D is the set of documents, assume that they are ordered in the decreasing upper bound score

(line 1). And the computed local contexts are maintained in a max-heap H (line 2). D and H will be initialized only on the first execution (lines 1∼2). When $H = \{\varnothing\}$, we assume that the top element in the heap $H.top()$ has a negative infinity score. Every time we get the next local context, if the top local context in H has a score greater than the upper bound score of the top document in D, then the top local context in H will be the next highest score local context. Otherwise, the top document in D needs to be inserted into H, and this step repeats until we ensure that the next highest score local context is the top result in H (lines 4∼7). We return the top local context in H as the next highest score local context (line 8).

Algorithm 1. Next(Q)

Input: a keyword query $Q = \{w_1, w_2, \ldots, w_l\}$.

Output: the next highest scored local context.

1: let D be the list of documents, accessing in the decreasing of estimated score;
2: let H be a max-heap that stores intermediate local contexts ($H \leftarrow \varnothing$ initially);
3: **while** not $D.empty()$ **and** $D.top() > H.top()$ **do**
4: $d_i \leftarrow D.next()$;
5: **for each** possible (d_i, v_j) pair **do**
6: compute the local context containing (d_i, v_j) with the best score;
7: insert the local context into H;
8: **return** $H.next()$.

6 Experiment

We have implemented six algorithms, based on different ranking semantics, to find the top-k documents with the help of a multi-attribute graph $\mathcal{G}$. The six algorithms are denoted as: ERank, tkp, EScore, OptProb, OptScore, and DocOnly. The first three, ERank, tkp, and EScore, are ranking based on the expected rank, top-k probability, and expected score as discussed in Section 4, respectively. OptProb and OptScore are two heuristics to choose the score of a document as the score of the local context in the corresponding supporting set (x-tuple) with the largest probability and score, respectively. DocOnly is to rank the documents based on the IR-score only, without a multi-attribute graph. All the algorithms were written in Visual C++ 2003. We conducted all the experiments on a 2.8 GHz CPU and 2GB memory PC running XP.

For testing the algorithms, we use the real datasets, for both documents and multi-attribute graph. For the documents, we use the movie review dataset, and we crawled 330 201 reviews about different movies from the Amazon website (`http://www.amazon.com`). For the multi-attribute graph, we use an IMDB movie database (`http://www.imdb.com/interfaces`). The multi-attribute graph is created in the same way as used in the literature of keyword search in RDB[15]. That is, we create a node for each tuple in the database, and there is a directed edge from node u to v if and only if there is a foreign key reference from u to v. In our experiments, we use 6 tables from IMDB database using the follow schema: Movie(Mid, Name), Genre(Mid, Genre), Director(Did, Name), Direct(Mid, Did), Actor(Aid, Name), and Play(Aid, Mid, Character), where primary keys are underlined. The numbers of tuples of the 6 tables are: 110 K, 148 K, 62 K, 113 K, 577 K, and 1410 K, respectively. In the resulting graph G_R, there are 2 423 262 nodes and 6 394 016 edges. The similarities between the matchings of movies in IMDB and movie reviews are calculated using the methods given in [16].

We evaluate the ranking accuracy of our algorithms using a (discounted) cumulative gain measure[17] based on human judgments. The (discounted) cumulative gain measure is widely used in measuring ranking accuracy of top-k queries[18-20]. The other measures, like precision, recall, and mean average precision, are inadequate in our problem setting for the following reasons. First, these measures assume that each document is either "definitely relevant" or "definitely irrelevant" to a keyword query, while a document can be relevant to some degree. Second, they need to know or retrieval all the relevant documents, while the number of relevant documents can be very large, and on the other hand, users are only interested in a few highly relevant documents.

In order to calculate the (discounted) cumulative gain, first, for a keyword query Q, a relevance value is assigned to each document. We use the relevance value between 0 and 4 (4 denoting high relevance, 0 denoting no relevance). Let r_i denote the relevance value of the document at the i-th position returned by a top-k query. The cumulative gain at rank position i, CG_i, is computed by summing the relevance value from position 1 to i.

$$CG_i = \begin{cases} r_i, & \text{if } i = 1, \\ CG_{i-1} + r_i, & \text{if } i > 1. \end{cases}$$

Discounted cumulative gain is defined similarly by incorporating rank-based discount factor. In our evaluation, we discount the relevance value of a document by the logarithm of its rank. The discounted cumulative gain at rank position i, DCG_i, is defined as below.

$$DCG_i = \begin{cases} r_i, & \text{if } i = 1, \\ DCG_{i-1} + r_i/\log_2 i, & \text{if } i > 1. \end{cases}$$

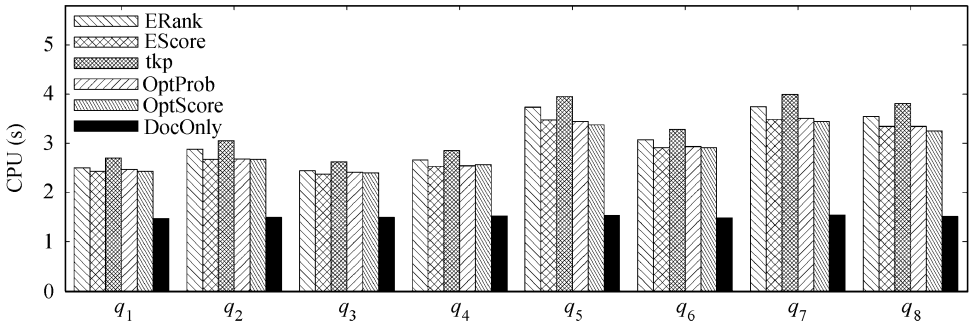

Fig.6. Execution time.

Table 1. Keyword Queries

Query	Keywords	Size
q_1	survive, murdered	2
q_2	marine, video	2
q_3	ducks, vista, disney	3
q_4	civil, war, peace	3
q_5	captain, excellent, music, sea	4
q_6	adventure, webster, fantastic, robert	4
q_7	michael, fan, freedom, range, endure	5
q_8	menagerie, star, hunter, west, river	5

For the keywords, we pick up 8 keyword queries q_1 to q_8 with relevant keywords of size varying from 2 to 5. The keywords selected cover a large range of selectivity, as shown in Table 1. According to the keyword frequency in documents, the query in decreasing average keyword frequency order is q_7, q_5, q_8, q_2, q_4, q_1, q_6, q_3, where q_7 has the keyword frequency 0.04 and q_3 has the keyword frequency 0.007. Consider the multi-attribute graph, the query in decreasing average keyword frequency order is q_7, q_6, q_5, q_4, q_8, q_2, q_3, q_1, where q_7 has the keyword frequency 0.0009 and q_1 has the keyword frequency 0.00003. Specifically, the query q_6 has a low keyword frequency in the documents, but has a high keyword frequency in the multi-attribute graph. The query q_7 has a high frequency in both components of the dataset, and the query q_4 has median keyword frequency in both components. The execution time of top-30 query of keywords from q_1 to q_8 are shown in Fig.6. As expected, DocOnly takes the least amount of time, because it only uses the document data, while other algorithms use the additional multi-attribute graph which is more complex than the plain text data. The other algorithms take almost the same amount of time. These algorithms need to compute the local contexts with the best score, which is the dominating cost.

6.1 Case Study for Query q_5

We evaluate the ranking accuracy of the keyword query $q_5 = \{$"captain", "excellent", "music", "sea"$\}$ in Table 1. For this query, we expect to find some reviews regarding excellent musical movies which are about sea and a captain. Most of the algorithms find the most relevant movie review in a top-1 result, which is a review about the movie "The Bounty (1995)". We run all the six algorithms for a top-30 query, and obtain 30 documents for each algorithm. Then, we assign relevance values (0∼4) to all these documents. Also, top-10 documents are selected in the rank order.

Based on the top-10 documents picked, we get a Precision-Recall style figure for each algorithm as shown in Fig.7. We report the least number of results needed to be returned by the algorithms to contain the top-k documents, for k ranged from 1 to 10. All the algorithms report correctly the top-1 document, except the OptProb algorithm. Because the top-1 document includes all the keywords of q_5, and the multi-attribute graph do not provide any informative information of this document for this keyword query. Overall, the uncertainty aware ranking algorithms (ERank, tkp, EScore) perform better than other heuristic algorithms.

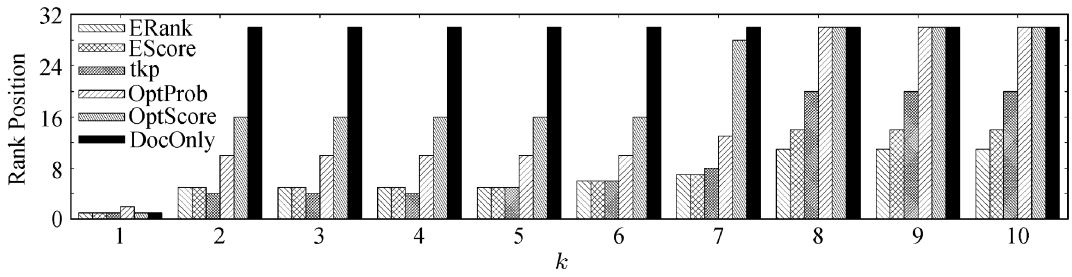

Fig.7. Least rank position to include the top-k (1 to 10) documents.

Fig.8 shows the cumulative gain and discounted cumulative gain for the top-10 documents returned by these algorithms. Different from Fig.9, the three algorithms: ERank, tkp, EScore have very similar cumulative and discounted cumulative gains, because in the cumulative gain measure different documents with the same relevance value are indistinguishable. Although the ranking between documents with the same relevance value are different for the algorithms, they result in the same (discounted) cumulative gain. Consistently, the uncertainty aware algorithms ERank, tkp, EScore perform better than the other algorithms.

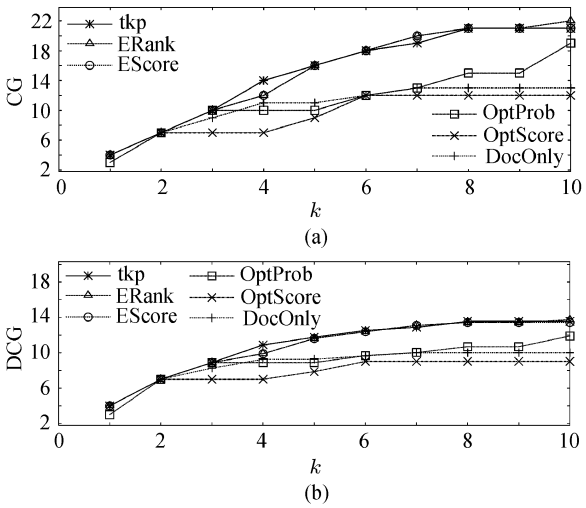

Fig.8. (Discounted) Cumulative gain for q_5. (a) Cumulative gain. (b) Discounted cumulative gain.

6.2 Case Study for Query q_6

We evaluate the ranking accuracy of the keyword query q_6 = {"adventure", "webster", "fantastic", "robert"} as shown in Table 1. This query intents to find the reviews for fantastic movies, either the genre of the movie is adventure or the review categorizes it as an adventure movie, and "webster" and "robert" are either character names or actor/director names. We run all the six algorithms for top-30 query, and obtain the 30 documents for each algorithm. We assign the relevance values ($0 \sim 4$) to all these documents. One different characteristic of q_6 from q_5 is that, while the relevant document for q_5 contains a lot of keywords in q_5, the keywords of q_6 mostly appear in the multi-attribute graph. Only very few long documents contain some (not all) keywords in q_6, and other documents only contain one out of the four keywords.

Fig.9 shows the cumulative gain and discounted cumulative gain for top-10 documents returned by these algorithms. Different from Fig.8, tkp performs better than ERank and EScore, because most part of the score of a local context comes from the multi-attribute graph for q_6. Note that, although the multi-attribute graph will increase the structure cost, it also increases the IR-score and completeness factor. The probability of a local context has more impacts on the documents ranking. The uncertainty aware algorithms ERank, tkp, and EScore perform better than other algorithms. The OptScore algorithm works badly in the scenario of q_6 such that only one document is relevant in the top-10 results.

Fig.10 shows the top-1 result returned by ERank, tkp, and EScore. In this case, the document contains only one keyword "fantastic". The multi-attribute graph provides other keywords for the documents, and the relationships between these keywords and the document to be ranked is very tight. This top-1 result cannot be found by DocOnly, as it contains only one keyword. OptProb and OptScore cannot find this top-1 result either, because there exist other local contexts with a larger score but lower probability.

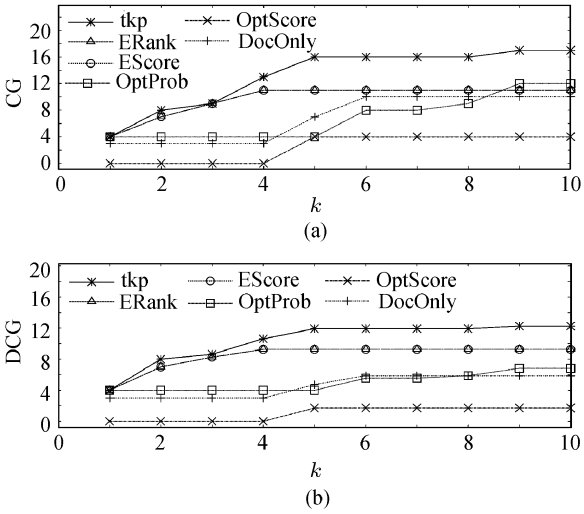

Fig.9. (Discounted) Cumulative gain for q_6. (a) Cumulative gain. (b) Discounted cumulative gain.

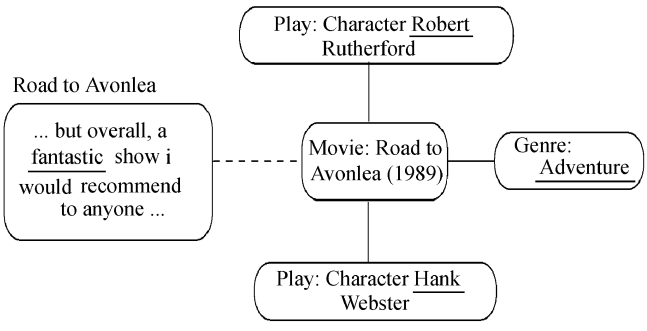

Fig.10. Top-1 document with context.

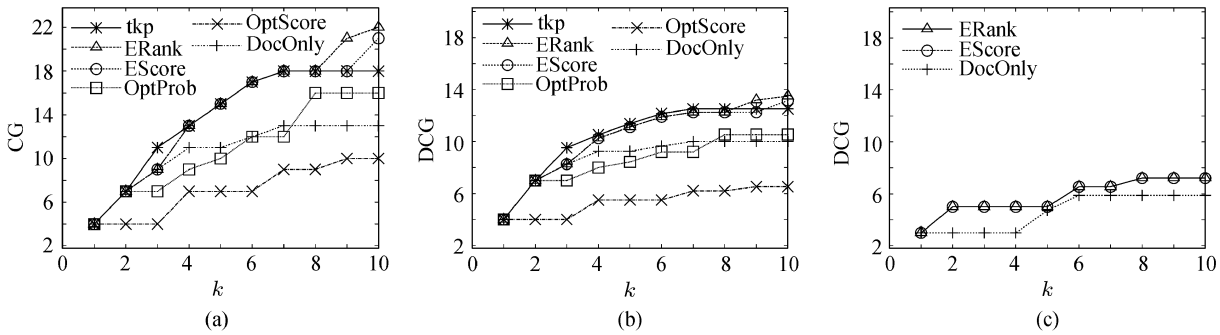

Fig.11. (Discounted) Cumulative gain for q_5 and q_6. (a) CG for q_5. (b) DCG for q_5. (c) DCG for q_6.

6.3 Multi-Attribute Graph

We also test the influence of the multi-attribute graph to the rank accuracy of keyword queries. We create another multi-attribute graph, G'_R, by removing the table Genre(Mid, Genre) from IMDB, where the keyword "music" used in the query q_5 and "adventure" used in the query q_6 appear in this table. Fig.11 shows the (discounted) cumulative gains for q_5 and q_6 respectively using the multi-attribute graph G'_R. As observed, with less information of the multi-attribute graph, for the query q_5, CG_8 decreases from 21 to 18, for the query q_6, the performance degrades much faster than that of q_5. For the query q_6, DCG_{10} decreases from 9.2 to 7.2, where it only decreases 0.2 for the query q_5. Another point is that tkp, OptProb, OptScore are more sensitive to the multi-attribute graph, because for the query q_6, those three algorithms return very few relevant answers.

7 Related Work

Fuzzy Object Matching. In the data integration and data cleaning, fuzzy object matching is a central problem, which is to reconcile objects from different collections that have used different naming conventions. Fuzzy object matching is also known as record linkage, de-duplication or merge-purge[4,21-23]. For text documents, edit distance[24] and Jaccard similarity on q-grams[25] are commonly used as the similarity score function. Efficient algorithms for computing the similarity scores are discussed in [26-27].

Keyword Search in RDBs (*Relational DataBases*). Keyword search in relational databases has been studied extensively recently. It provides users an easy way to get insights of the underling RDB. Most of the works concentrate on finding minimal connected tuple trees from an RDB[2-3,6-7,28-33]. There are two approaches, based on the representation of data. One accesses data directly using SQL, where the data is stored in an RDB[6,29-32]. The other materializes an RDB as a graph, then all the operations are on the graph[2-3,7,28,33]. As an exception, there is work on finding communities, which is a multi-center induced subgraph, on a materialized graph based on a keyword query[15].

All the works above only consider one single database. Li *et al.*[34] propose keyword search over heterogeneous data, e.g., unstructured data (e.g., Web pages, linked together through hyper links), semi-structured data (e.g., XML), structured data (e.g., Database). But in their work, there is no connection between different datasets, they are different connected components in a whole graph. Sayyadian *et al.*[35] propose a system Kite to answer a keyword query over multiple databases. Kite combines schema matching and structure discovery techniques to find approximate foreign-key joins across heterogeneous databases. In their work, the confidence of foreign-key joins in a result contributes linearly to the final score.

Top-k Queries in Probabilistic Data. Uncertain data has received increasing attention recently. Most of them represent the uncertainty as a probability value, also called probabilistic data. Trio system[11,36-37] is a frequently used uncertain database model, which is a disjoint-independent probabilistic database[38]. Even on a totally independent probabilistic database, answering a general SQL query is #P-complete[39], specifically it is #P-complete to compute the existence probability of a result.

Apart from the works on different models of uncertain relational database, some recent works are concerned with answering top-k queries in uncertain databases. There are two scenarios in ranking the query results. One is to integrate the probabilities in the score function[40] (or score function is just the existence probability[16]). The other is based on the possible worlds semantics[8-10,13], where probability is used to define possible worlds and a score is used to define the relative order of results in each possible world.

Possible worlds based query ranking determines the top-k result by the interplay between score and probability. In [8-10], they get the top-k tuples in an

x-Relation[11], where each tuple has both a score and a probability. U-Top$K$ and U-kRanks queries are first proposed in [8], Yi *et al.*[9] improve the performance of the two queries using a dynamic programming approach. Hua *et al.* in [10] define the PT-k query, and proposed three approaches to answer the PT-k query, which are, dynamic programming method, sampling method, and Poisson approximation based method. Jin *et al.* in [12] adapt the U-Top$K$/U-kRanks/PT-k/Global-Top$k$ (where Global-Top$k$ is the same as Pk-top$k$ in [12]) queries in an uncertain stream environment based on a sliding-window model, and design both space- and time-efficient synopses to continuously monitor the top-k results. In [9-10], to answer a U-kRanks query or PT-k query, they compute the $r_{i,j}$ variables for $1 \leqslant i \leqslant N$ and $0 \leqslant j \leqslant K-1$. In [12], they just consider the single-alternative case.

8 Conclusion

In this paper, we focus on a new keyword search problem, which is to rank documents in a context (a multi-attribute graph), for a user-given l-keyword query. There are two main data sources in the problem. One is a set of documents, and the other is a relational database which the multi-attribute graph represents, the relationships between documents and the tuples in the relational database are with uncertainty, which is caused by similarity match. We discuss how to model using a graph $\mathcal{G}$, and how to score and rank. Our work involves two different issues: ranking and uncertainty, and the unique issue is that the ranking needs to take uncertainty into consideration because a result with a higher rank may be with a lower probability to occur. We present our score function with two components (the IR-styled score and the structural cost). We confirmed the effectiveness of our approach using real datasets in this paper.

References

[1] Baeza-Yates R, Ribeiro-Neto B. Modern Information Retrieval. 1st Edition, Addison Wesley, May 1999.

[2] Kacholia V, Pandit S, Chakrabarti S, Sudarshan S, Desai R, Karambelkar H. Bidirectional expansion for keyword search on graph databases. In *Proc. VLDB 2005*, Trondheim, Norway, Aug. 30-Sept. 2, 2005, pp.505-516.

[3] Ding B, Yu J X, Wang S, Qin L, Zhang X, Lin X. Finding top-k min-cost connected trees in databases. In *Proc. ICDE 2007*, Istanbul, Turkey, April 15-20, 2007, pp.836-845.

[4] Ananthakrishna R, Chaudhuri S, Ganti V. Eliminating fuzzy duplicates in data warehouses. In *Proc. VLDB 2002*, Hong Kong, China, Aug. 20-23, 2002, pp.586-597.

[5] Li W S, Candan K S, Vu Q, Agrawal D. Retrieving and organizing web pages by "information unit". In *Proc. WWW 2001*, Hong Kong, China, May 1-5, 2001, pp.230-244.

[6] Luo Y, Lin X, Wang W, Zhou X. Spark: Top-k keyword query in relational databases. In *Proc. SIGMOD 2007*, Beijing, China, June 11-14, 2007, pp.115-126.

[7] He H, Wang H, Yang J, Yu P S. Blinks: Ranked keyword searches on graphs. In *Proc. SIGMOD 2007*, Beijing, China, June 11-14, 2007, pp.305-316.

[8] Soliman M A, Ilyas I F, Chang K C C. Top-k query processing in uncertain databases. In *Proc. ICDE 2007*, Istanbul, Turkey, April 15-20, 2007, pp.896-905.

[9] Yi K, Li F, Kollios G, Srivastava D. Efficient processing of top-k queries in uncertain databases with x-Relations. *IEEE Trans. Knowl. Data Eng.*, 2008, 20(12): 1669-1682.

[10] Hua M, Pei J, Zhang W, Lin X. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. SIGMOD 2008*, Vancouver, Canada, June 9-12, 2008, pp.673-686.

[11] Agrawal P, Benjelloun O, Sarma A D, Hayworth C, Nabar S U, Sugihara T, Widom J. Trio: A system for data, uncertainty, and lineage. In *Proc. VLDB 2006*, Seoul, Korea, Sept. 12-15, 2006, pp.1151-1154.

[12] Jin C, Yi K, Chen L, Yu J X, Lin X. Sliding-window top-k queries on uncertain streams. In *Proc. VLDB 2008*, Auckland, New Zealand, Aug. 23-28, 2008, pp.301-312.

[13] Cormode G, Li F, Yi K. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. ICDE 2009*, Shanghai, China, March 29-April 2, 2009, pp.305-306.

[14] Li J, Deshpande A. Consensus answers for queries over probabilistic databases. In *Proc. PODS 2009*, Providence, USA, June 29-July 1, 2009, pp.259-268.

[15] Qin L, Yu J X, Chang L, Tao Y. Querying communities in relational databases. In *Proc. ICDE 2009*, Shanghai, China, Mar. 29-Apr. 2, 2009, pp.724-735.

[16] Re C, Dalvi N N, Suciu D. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE 2007*, Istanbul, Turkey, April 15-20, 2007, pp.886-895.

[17] Järvelin K, Kekäläinen J. IR evaluation methods for retrieving highly relevant documents. In *Proc. SIGIR 2000*, Athens, Greece, July 24-28, 2000, pp.41-48.

[18] Burges C J C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G N. Learning to rank using gradient descent. In *Proc. ICML 2005*, Bonn, Germany, Aug. 7-11, 2005, pp.89-96.

[19] Geng X, Liu T Y, Qin T, Arnold A, Li H, Shum H Y. Query dependent ranking using k-nearest neighbor. In *Proc. SIGIR 2008*, Singapore, July 20-24, 2008, pp.115-122.

[20] Dou Z, Song R, Yuan X, Wen J R. Are click-through data adequate for learning web search rankings? In *Proc. CIKM 2008*, Napa Valley, USA, Oct. 26-30, 2008, pp.73-82.

[21] Chaudhuri S, Ganjam K, Ganti V, Motwani R. Robust and efficient fuzzy match for online data cleaning. In *Proc. SIGMOD 2003*, San Diego, USA, June 9-12, 2003, pp.313-324.

[22] Hernandez M A, Stolfo S J. The merge/purge problem for large databases. In *Proc. SIGMOD 1995*, San Jose, USA, May 22-25, 1995, pp.127-138.

[23] Cohen W W, Ravikumar P, Fienberg S E. A comparison of string distance metrics for name-matching tasks. In *Proc. IIWeb 2003*, Acapulco, Mexico, Aug. 9-10, 2003, pp.73-78.

[24] Ukkonen E. On approximate string matching. In *Proc. FCT*, Borgholm, Sweden, Aug. 21-27, 1983, pp.487-495.

[25] Gravano L, Ipeirotis P G, Jagadish H V, Koudas N, Muthukrishnan S, Srivastava D. Approximate string joins in a database (almost) for free. In *Proc. VLDB 2001*, Rome, Italy, Sept. 11-14, 2001, pp.491-500.

[26] Bayardo R J, Ma Y, Srikant R. Scaling up all pairs similarity search. In *Proc. WWW 2007*, Banff, Canada, May 8-12, 2007, pp.131-140.

[27] Xiao C, Wang W, Lin X, Yu J X. Efficient similarity joins for near duplicate detection. In *Proc. WWW 2008*, Beijing, China, Apr. 21-25, 2008, pp.131-140.

[28] Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword searching and browsing in databases using banks. In *Proc. ICDE 2002*, San Jose, USA, Feb. 26-Mar. 1, 2002, pp.431-440.
[29] Hristidis V, Papakonstantinou Y. Discover: Keyword search in relational databases. In *Proc. VLDB 2002*, Hong Kong, China, Aug. 20-23, 2002, pp.670-681.
[30] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-style keyword search over relational databases. In *Proc. VLDB 2003*, Berlin, Germany, Sept. 9-12, 2003, pp.850-861.
[31] Agrawal S, Chaudhuri S, Das G. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. ICDE 2002*, San Jose, USA, Feb. 26-Mar. 1, 2002, p.5.
[32] Liu F, Yu C T, Meng W, Chowdhury A. Effective keyword search in relational databases. In *Proc. SIGMOD 2006*, Chicago, USA, June 27-29, 2006, pp.563-574.
[33] Golenberg K, Kimelfeld B, Sagiv Y. Keyword proximity search in complex data graphs. In *Proc. SIGMOD 2008*, Vancouver, Canada, 2008, pp.927-940.
[34] Li G, Ooi B C, Feng J, Wang J, Zhou L. Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proc. SIGMOD 2008*, Vancouver, Canada, June 9-12, 2008, pp.903-914.
[35] Sayyadian M, LeKhac H, Doan A, Gravano L. Efficient keyword search across heterogeneous relational databases. In *Proc. ICDE 2007*, Istanbul, Turkey, April 15-20, 2007, pp.346-355.
[36] Sarma A D, Benjelloun O, Halevy A Y, Widom J. Working models for uncertain data. In *Proc. ICDE 2006*, Atlanta, USA, April 3-8, 2006, p.7.
[37] Benjelloun O, Sarma A D, Halevy A Y, Widom J. Uldbs: Databases with uncertainty and lineage. In *Proc. VLDB 2006*, Seoul, Korea, Sept. 12-15, 2006, pp.953-964.
[38] Dalvi N N, Suciu D. Management of probabilistic data: Foundations and challenges. In *Proc. PODS 2007*, Beijing, China, June 11-13, 2007, pp.1-12.
[39] Dalvi N N, Suciu D. Efficient query evaluation on probabilistic databases. *VLDB J.*, 2007, 16(4): 523-544.
[40] Ljosa V, Singh A K. Top-k spatial joins of probabilistic objects. In *Proc. ICDE 2008*, Cancun, Mexico, Apr. 7-12, 2008, pp.566-575.

Li-Jun Chang received the B.S. degree from Renmin University of China in 2007. He is currently a Ph.D. candidate in the Department of Systems Engineering & Engineering Management, The Chinese University of Hong Kong, Hong Kong, China. His research interests include uncertain data management and keyword search in databases.

Jeffrey Xu Yu received his B.E., M.E. and Ph.D. degrees in computer science, from the University of Tsukuba, Japan, in 1985, 1987 and 1990, respectively. Dr. Yu held teaching positions in the Institute of Information Sciences and Electronics, University of Tsukuba, Japan, and the Department of Computer Science, The Australian National University. Currently, he is a professor in the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. Dr. Yu served as an associate editor of IEEE Transactions on Knowledge and Data Engineering, and is serving as a VLDB Journal editorial board member, and an ACM SIGMOD Information Director. His current main research interest includes graph database, graph mining, keyword search in relational databases, XML database, and Web-technology. He has published over 190 papers including the papers published in reputed journals and major international conferences.

Lu Qin received the B.S. degree from Renmin University of China in 2006. He is currently a Ph.D. candidate in the Department of Systems Engineering & Engineering Management, The Chinese University of Hong Kong, Hong Kong, China. His research interests include keyword search in relational databases, keyword search in large graphs and multi-query optimization in RDBMSs.