Virtual Documents and Answer Priors in Keyword Search over Data Graphs[*]

Yosi Mass[†‡] and Yehoshua Sagiv[‡]
[†]IBM Haifa Research Lab, Haifa 31905, Israel
yosimass@il.ibm.com
[‡]The Hebrew University, Jerusalem 91904, Israel
sagiv@cs.huji.ac.il

ABSTRACT

In keyword search over data graphs, an answer is a non-redundant subtree that contains the keywords of the query. Ranking of answers should take into account both their textual relevance and the significance of their semantic structure. A novel method for answers priors is developed and used in conjunction with query-dependent features. Since the space of all possible answers is huge, efficiency is also a major problem. A new algorithm that drastically cuts down the search space is presented. It generates candidate answers by first selecting top-n roots and top-n nodes for each query keyword. The selection is by means of a novel concept of virtual documents with weighted term frequencies. Markov random field models are used for ranking the virtual documents and then the generated answers. The proposed approach outperforms existing systems on a standard evaluation framework.

Keywords

Data graph, keyword search, query, ranking, answer prior, virtual documents

1. INTRODUCTION

Data graphs are a convenient, flexible way of representing knowledge bases. They can be constructed from a variety of formats (e.g., XML, RDB and RDF). Their semistructured nature makes it possible to create them incrementally, distributively and heterogeneously. Since they do not have a rigid schema, it is essential to support keyword search over them (rather than querying in some formal language, such as XQuery or SPARQL). Yet, it could be possible to get succinct answers that have some semantic structure. In particular, their answers can show semantic connections between distinct units (e.g., Web pages), which is impossible in ordinary keyword search, where a result is always a single unit (e.g., document).

The nodes of a data graph represent entities and relationships, while the connections among them are introduced as edges. Free text can be associated with nodes and edges. In keyword search over data graphs, answer are non-redundant subtrees that contain all the keywords of the query.

Keyword search over data graphs has been investigated extensively in recent years (cf. [5]). It involves two main challenges: *effectiveness* and *efficiency*. The first one means that we have to develop effective methods for ranking of answers (i.e., subtrees) that take into account the structure (e.g., the importance of entities and the strength of relationships among them) as well as the relevance of the keywords of the query. The second challenge is to generate candidate (i.e., potentially relevant) answers efficiently. This is not an easy problem, because there could be a huge number of subtrees that contain all the keywords of the query.

A common approach begins by assigning weights to the nodes and edges of the data graph. Then, the search for answers starts from nodes containing keywords of the query. The weights are intended to focus the search on paths that lead to roots of the most relevant answers. However, those weights are assigned based on local considerations (e.g., the text of a node) and, hence, their effectiveness is limited for the following reason. Nodes belonging to relevant answers are those that have, in their vicinity, other nodes with keywords of the query. To solve this problem, we introduce virtual documents (VDs) that contain a node and its vicinity. We rank VDs by applying (similarly to [1,20]) a Markov random field (MRF) model that combines query-dependent and independent features. In particular, we adapt positional language models [18] by using distances based on static weights that reflect the structure of the data graph. Also, we use node priors as a query-independent feature.

We apply the ranking of VDs to the process of selecting the top-n keyword nodes (i.e., nodes containing keywords of the query) and the top-n roots. Only after selecting both keyword nodes and roots, do we construct paths that connect them, thereby yielding answers. By first selecting roots and not just keyword nodes, we realize a higher degree of both effectiveness and efficiency. It should be noted that several papers [10, 21] use notions of virtual documents that are also returned as answers. In contrast, we apply VDs just as a first step in the construction of answers.

For the final ranking of the generated answers, we use once again an MRF model for combining query-dependent and independent features. Unlike previous work, our query-

[*]This work was supported by the Israel Science Foundation (Grant No. 1632/12).

independent feature is not ad hoc, but based on a rigorous computation of answer priors.

We have evaluated our system on the framework of [5] that consists of three datasets: IMDB, Wikipedia and Mondial, as well as fifty queries for each one. On all three datasets, our method outperforms the state-of-the-art systems. Our experiment include automatic learning of the parameters of the MRF models.

We also show that our approach offers an excellent way of improving the efficiency with just a minor impact on the effectiveness (i.e., quality of answers). It is done by decreasing n (i.e., the number of selected roots and keyword nodes).

In summary, our main contributions are the following.

1. By developing virtual documents and using them for first selecting the top-n keyword nodes *and* also the top-n roots, we have an algorithm for generating answers that is both highly efficient and effective.

2. We formally compute answer priors and use them as a query-independent feature in the final ranking. Our experiments show that they make a highly significant contribution to the effectiveness of our system. In the area of keyword search over data graphs, this is the first time that query-independent features are based on a rigorous formalism, rather than ad hoc intuition.

3. We describe an efficient implementation using inverted indexes. We have done extensive experiments showing that our system is highly effective and efficient.

The rest of the paper is organized as follows. In Section 2, we discuss related work. Section 3 reviews basic concepts. Section 4 develops the virtual documents and their features. Section 5 presents the algorithm that uses the selected roots and keyword nodes for generating answers. Section 6 develops the features of answers. Section 7 describes the implementation and the experiments that verify the effectiveness and efficiency of our approach. We conclude in Section 8.

2. RELATED WORK

We discuss systems for keyword search over data graphs along the dimensions of efficiency (i.e., how to generate answers) and effectiveness (i.e., how to rank answers). The common approach to generate answers efficiently [2, 12, 19] uses backward iterators that start from keyword nodes until they meet at a root node. In [13] they improved [2] by adding forward iterators that can go back from the detected roots, to keyword nodes. Still, they have to start from keyword nodes. In [8], they present a method for keyword search over RDF graphs that starts with triples that match each keyword. Then they produce answers by joining the triples through their subjects and objects. In our work we use virtual documents to start simultaneously from roots and keyword nodes, thus we further reduce the search space.

Other papers have also used virtual documents in keyword search over data graphs. In [10, 21], they create a virtual document from a node and its vicinity, and search it for the keywords. In [4], they do entity search over RDF data and their virtual documents contain a node, but only with its literal neighbors. All of these systems return virtual documents or their roots as answers, whereas we use them just as a first step to generate answers (i.e., subtrees).

For an effective ranking of answers (cf. [7]), systems use features that consider both relevance to the keywords of the query and to the structure of answers. In [3] they use pseudo-feedback to apply relevance models to tuples and use them for ranking of answers. The works of [12, 17, 19] resemble our work in that they concatenate the text in the nodes of answers and use IR methods combined with query-independent features for ranking. We also apply IR features on the concatenated text, but different from those works, we present a probabilistic feature that considers the prior of an answer, and we use a well established theory of MRF for combining it with the IR features.

In [16] they builds priors to paths in graphs for the application of recommendation of conferences, papers to cite, or experts for a new paper that one writes. In comparison, we assign different priors to each answer (i.e., subtree) while they assign priors for paths, and their priors are fixed for all paths of the same type.

In [15], they learn per-term weights for each field; in our case, it gave inferior results. So, we use a term-independent weight for each field as in [11, 14].

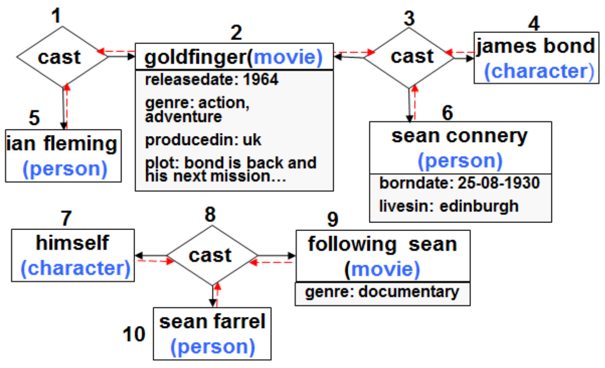

Figure 1: Tiny snippet of the IMDB data graph

3. PRELIMINARIES

3.1 The Data Model

Data graphs can be created from any format (XML, RDF, RDB, etc.). Figure 1 shows a tiny portion of the IMDB data graph. In this paper, we experiment with data graphs that are obtained from relational databases as follows. Each tuple t becomes a node v_t that has the relation name as its `type`. A tuple t can be either an entity or a relationship. In the former case, v_t is an *entity* node and is shown as a rectangle in Figure 1. In the latter case, v_t is a *relationship* node and is shown as a diamond. The attribute-value pairs of v_t are those of the tuple t, excluding foreign keys. That is, a foreign key is represented in the data graph by a directed edge (shown in Figure 1 as a solid arrow). An *opposite* edge (shown as a dashed arrow) is added in the reverse direction in order not to miss relevant answers.

In Figure 1, the diamond of a relationship node shows the `type` (e.g., `cast`). The top line of a rectangle shows the entity's `name` (e.g., `goldfinger`) followed by its `type` (e.g., `movie`). The other attribute-value pairs are listed below the top line.

The value of an attribute could be free text. For the name of an entity node, we choose the value of an appropriate attribute, such as `title`, etc. We shall refer to that attribute generically as `name`. The value of `name` is a short string that serves as a not-necessarily-unique identifier; for example, it could be the title of a movie or the name of a person. Relationship nodes typically do not have the attribute `name`.

3.2 The Content and Title Fields

In [9,11,14], they showed that combining fields yields better results when searching XML, the Web or flat documents. Similarly, we group the attributes of a node into two semantic *fields* (that could overlap). The *content* field consists of the names and values of all the attributes. The *title* field comprises only the value of the attribute `name`. In Figure 1, for example, the title field of node 2 is: `goldfinger`, and the content field is: **`title goldfinger type movie release-date 1964 genre action adventure producedin uk plot bond is back and his next mission ...`**

The content and title fields (of a node) contain textual information that we use to assign IR scores. We distinguish between these two fields in order to control the relative importance of an occurrence (of a query keyword) in the title compared with an occurrence only in the content field.

3.3 Queries and Answers

A query $Q = (q_1, ..., q_m)$ on a data graph is a set of keywords. Each q_i should match a term in the content of some node(s); that is, we use the AND semantics as usually done in keyword search over data graphs (supporting the OR semantics is left for future work).

Answers are subtrees of the data graph, rather than subgraphs, because a tree is easier to understand quickly and is typically an indivisible unit of information. Formally, an *answer* to Q is a *non-redundant* subtree a of the data graph, such that a *contains* all the keywords of Q. Containment means that each keyword appears in some node(s). Non-redundancy requires an answer *not* to have a proper subtree that also contains all the keywords of the query.

As an example, consider the data graph of Figure 1 and the query “sean connery fleming” for finding movies that are related to those names. A possible answer comprises five nodes: `goldfinger` (of type `movie`), `ian fleming` and `james bond` (both of type `person`), and the two connecting nodes of type `cast`. Note that non-redundancy does *not* imply minimality, and a query could have numerous answers.

3.4 Markov Random Fields

Markov random field (MRF) models make it possible to combine query-dependent and independent features. We apply the sequential dependency model of [1, 20] and use unigrams and unordered bigrams as query-dependent features. Our query-independent feature is the prior of either a node or an answer. The score with respect to a query $Q = (q_1, ..., q_m)$ is given by

$$\begin{aligned} score(Q, x) = &\sum_{q_i \in Q} \left[\lambda_T f_T(q_i, x) + \lambda_{\hat{T}} f_{\hat{T}}(q_i, x)\right] \\ &+ \sum_{\{q_i, q_{i+1}\} \in Q} \left[\lambda_U f_U(q_i, q_{i+1}, x) + \lambda_{\hat{U}} f_{\hat{U}}(q_i, q_{i+1}, x)\right] \\ &+ \lambda_L f_L(x), \end{aligned} \tag{1}$$

where x is either a node or an answer, and the potential function are: f_T and $f_{\hat{T}}$ for unigrams of the content and title fields, respectively; similarly, f_U and $f_{\hat{U}}$ for unordered bigrams; and f_L for the query-independent feature. The parameters λ_T, $\lambda_{\hat{T}}$, λ_U, $\lambda_{\hat{U}}$ and λ_L are nonnegative and their sum is 1. We learn them automatically (see Section 7).

We actually use two variants of Equation (1). The algorithm for generating answers (Section 5) starts by selecting the top-n roots and keyword nodes using the potential functions f^n_T, $f^n_{\hat{T}}$, f^n_U, $f^n_{\hat{U}}$ and f^n_L that are defined in Section 4.4. After the algorithm generates n answers, they are re-ranked using the potential functions f^a_T, $f^a_{\hat{T}}$, f^a_U, $f^a_{\hat{U}}$ and f^a_L that are defined in Section 6.

4. VIRTUAL DOCUMENTS FOR RANKING

4.1 Virtual Documents

We consider a data graph $G = (V, E)$, where V and E are the sets of nodes and edges, respectively. In [19], each node $v \in V$ is deemed a document. We take a different approach and view a small vicinity of a node (including the node itself) as a virtual document. Intuitive motivation is that often keywords of a query are spread over several nodes that are close to one another.

Formally, the *virtual document* (abbr. VD) of a node v, denoted by $v^\star$, consists of all nodes u, such that the following holds. There is a path p in the data graph G from v to some entity node x (where x could be u), such that p includes u and has at most τ entity nodes, excluding v itself. The parameter τ is called the *diameter* of the VD. As a running example, we use Figure 1 with $\tau = 1$. The VD of node 6 consists of nodes 2, 3, 4 and 6 (node 3 is not counted in τ, because it is a relationship). The VD of node 2 comprises nodes 1, 2, 3, 4, 5, and 6. Note that $v^\star$ is defined as a set of nodes. However, $v^\star$ can also be viewed as the subgraph of G induced by its nodes (i.e., the subgraph comprising the nodes of $v^\star$ and the edges between them).

A node consists of two fields: content and title. We denote by v_f the text in the field f of node v. In particular, v_{cnt} and v_{ttl} are ordinary documents consisting of the text in the content and title fields, respectively, of v. Similarly, $v^\star_f$ denotes the field f of the VD $v^\star$, that is, the concatenation of the text in field f of all the nodes comprising $v^\star$. (However, the weighted term frequencies defined later are applied to $v^\star$.) Recall that V is the set of nodes of the data graph. We use V_f to denote the collection that comprises all the v_f.

4.2 Static Weights

In the VD $v^\star$ of a node v, occurrences of terms closer to v are more important. Distances among nodes are determined by *minimal-weight* paths. We assign *static weights* (which are query independent) to nodes and edges as follows.

For an entity node u, the importance is proportional to the number of its neighbors. Thus, the static weight of u, denoted by $wn_s(u)$, is defined as

$$wn_s(u) = \frac{1}{\ln(e + Deg(u))}, \tag{2}$$

where e is Euler's number and $Deg(u)$ is the degree of u (i.e., the number of its edges) in G. Notice that $0 \leq wn_s(u) \leq 1$ and a node with a higher degree has a better (i.e., lower) weight. We use logarithm in the denominator so that the weight will not decay too fast as the degree increases, or else there would be a negligible difference between nodes with large degrees. For example, node 2 in Figure 1 has two neighbors, so its static weight is 0.64; the static weight of node 4 is 0.76, because it has a single neighbor.

Next, we consider relationship nodes and edges. Two entity nodes are *directly related* if they are connected by either a single edge or a pair of edges that pass through a relationship node. In this paper, we do not consider types of nodes and edges when determining static weights. In addition, the degree of a relationship node is usually 2 or 3. Therefore, we apply the rule that the static weight of a direct relationship is always 1. In this way, we give some preference to smaller answers (i.e., with fewer nodes). To conform to the above rule, the static weight $wn_s(u)$ of a relationship node u is 1. The static weight of an edge e, denoted by $we_s(e)$, is 1 if e connects two entity nodes; otherwise e connects an entity with a relationship node and $we_s(e) = 0$.

A path p from node v to node u is written as $p_{v\to u}$. The *static weight* of $p_{v\to u}$, denoted by $w_s(p_{v\to u})$, is the sum of static weights of all the edges and nodes of p; that is,

$$w_s(p_{v\to u}) = \sum_{e\in p} we_s(e) + \sum_{x\in p} wn_s(x), \tag{3}$$

where the first sum is over all edges e of p and the second—over all nodes x of p. Note that if the path consists of only node v, its weight is $wn_s(v)$.

4.3 Weighted Term Frequencies

Next, we define the weighted terms frequencies of unigrams and unordered bigrams in a VD $v^\star$. Given a node u of $v^\star$, the *relative static weight* of u in $v^\star$, denoted by $w_s(v^\star, u)$, is the minimum weight over all paths from v to u in $v^\star$; that is,

$$w_s(v^\star, u) = \min_{p_{v\to u} \text{ is in } v^\star} w_s(p_{v\to u}). \tag{4}$$

Note that all the nodes and edges of $p_{v\to u}$ are in $v^\star$. For example, in Figure 1, the relative static weight of node 4 in the VD of node 2 is $w_s(2^\star, 4) = 0.64 + 1 + 0.76 = 2.4$.

Let t be either a unigram or an unordered bigram. To define the frequency of t in $v^\star$, we adapt the method used in positional language models [18]. That is, the weight of t in a node u of $v^\star$ is inversely proportional to $w_s(v^\star, u) - w_s(v^\star, v)$, which is the weighted distance of u from v in the VD $v^\star$. In particular, a kernel serves as a discounting factor. We use a Gaussian kernel, because it was shown to be the best [18]. Formally, the *weighted term frequency* of t in field f of $v^\star$, denoted by $wtf(t, v^\star_f)$, is

$$wtf(t, v^\star_f) = \sum_{u\in v^\star} e^{\frac{-(w_s(v^\star,u)-w_s(v^\star,v))^2}{2\sigma^2}}\, tf(t, u_f), \tag{5}$$

where $tf(t, u_f)$ is the ordinary term frequency of t in the field f of node u and σ is a parameter that controls the spread of the kernel.

Note that the sum in Equations (5) is over all nodes u in $v^\star$. Observe that the weight of a single occurrence of t is at most one, and it is exactly one in v. For example, in Figure 1, the keyword `bond` appears twice in the VD of node 2: once in node 2 itself and once in node 4. If we set $\sigma = 1$, then $wtf(\texttt{bond}, 2^\star_{cnt}) = 1 + 0.21 = 1.21$.

4.4 Node Potential Functions

Consider a query $Q = (q_1, ..., q_m)$. We now define potential functions for unigrams, unordered bigrams and nodes. In Section 5, we use Equation (1) with these functions.

Unigrams. For unigrams, we use two potential functions $f^n_T(q_i, v)$ and $f^n_{\hat{T}}(q_i, v)$ for the content and title fields, respectively. These functions consider the fields of the VD $v^\star$ (rather than node v itself). They are defined by

$$f^n_T(q_i, v) = \ln\big((1-\alpha^n_T)P(q_i|v^\star_{cnt}) + \alpha^n_T P(q_i|V_{cnt})\big), \tag{6}$$

$$f^n_{\hat{T}}(q_i, v) = \ln\big((1-\alpha^n_{\hat{T}})P(q_i|v^\star_{ttl}) + \alpha^n_{\hat{T}} P(q_i|V_{ttl})\big), \tag{7}$$

where α^n_T and $\alpha^n_{\hat{T}}$ are smoothing parameters for the content and title fields, respectively, and V_{cnt} and V_{ttl} are the collections comprising the content and title fields, respectively, of all the nodes of the data graph. We use Dirichlet smoothing for α^n_T and $\alpha^n_{\hat{T}}$, as described in Section 7.3.

We use the maximum likelihood estimate. Hence, in each one of Equations (6) and (7), the first probability in the right side is given by

$$P(q_i|v^\star_x) = \frac{wtf(q_i, v^\star_x)}{\sum_{t\in v^\star_x} wtf(t, v^\star_x)}, \tag{8}$$

where we use weighted term frequencies (defined by Equation (5)) and x is either *cnt* or *ttl*. The summation in the denominator is over all unigrams t that appear in $v^\star_x$ and is called the *length* of $v^\star_x$.

In each of Equations (6) and (7), the second probability in the right side (which does the smoothing with the collection) is given by

$$P(q_i|V_x) = \frac{\sum_{u\in V} tf(q_i, u_x)}{\sum_{u\in V}\sum_{t\in u_x} tf(t, u_x)}. \tag{9}$$

where x is either *cnt* or *ttl*.

Unordered bigrams. For an unordered bigram $\{q_i, q_{i+1}\}$ of Q (similarly to unigrams), we use the following two potential functions for the content and title fields.

$$f^n_U(q_i, q_{i+1}, v) = \ln\big((1-\alpha^n_U)P(\{q_i, q_{i+1}\}|v^\star_{cnt}) + \\ + \alpha^n_U P(\{q_i, q_{i+1}\}|V_{cnt})\big), \tag{10}$$

$$f^n_{\hat{U}}(q_i, q_{i+1}, v) = \ln\big((1-\alpha^n_{\hat{U}})P(\{q_i, q_{i+1}\}|v^\star_{ttl}) + \\ + \alpha^n_{\hat{U}} P(\{q_i, q_{i+1}\}|V_{ttl})\big) \tag{11}$$

Here, α^n_U and $\alpha^n_{\hat{U}}$ are the smoothing parameters for unordered bigrams. Similarly to unigrams, we use Dirichlet smoothing for these parameters. As earlier, the probabilities in Equations (10) and (11) are derived according to the maximum likelihood estimate. That is, they are given by Equations (8) and (9), respectively, except that we substitute $\{q_i, q_{i+1}\}$ for q_i and assume that t denotes unordered bigrams (rather than unigrams).

Query independent. We use one query-independent potential function that is given by the node prior. In particular, we assume that the probability of a node v is proportional to its degree in the graph. Hence,

$$f^n_L(v) = \ln \frac{Deg(v)}{\sum_{u\in V} Deg(u)}. \tag{12}$$

Overall, there are five potential functions and, thus, we have to learn five parameters (i.e., λ^n_T, $\lambda^n_{\hat{T}}$, λ^n_U, $\lambda^n_{\hat{U}}$ and λ^n_L). In the next section, we use the five functions twice: once for selecting roots and a second time for choosing keyword nodes. The learning is done separately for each one of these two cases, as described in Section 7.3.

5. GENERATING ANSWERS

In this section, we rank nodes according to $\mathit{score}(Q, v)$ of Equation (1), where the potential functions are given by Equations (6), (7), (10), (11) and (12).

We begin by selecting roots and keyword nodes. The former will be roots of answers. The latter will appear in answers as nodes containing keywords of the given query $Q = (q_1, \ldots, q_m)$. We do it as follows. First, we consider all nodes v of G, such that $v^\star$ contains every q_i. We rank them according to $\mathit{score}(Q, v)$ and select the top-n. These are the *selected roots.* Second, we consider the set U of all nodes v, such that v is in $r^\star$ where r is a selected root. Let U_i be the subset of U that comprises all nodes v, such that v contains the keyword q_i of Q. For each keyword $q_i \in Q$, we rank the nodes of U_i according to $\mathit{score}(Q, v)$ and choose the top-n. These are the *selected keyword nodes* (for q_i). Let S be the set consisting of all the selected roots and keyword nodes.

We will construct answers from minimal-weight paths that connect the selected roots and keyword nodes. We want the weight of a path from a root r to a keyword node v to reflect also the scores of its endpoints (i.e., $\mathit{score}(Q, r)$ and $\mathit{score}(Q, v)$), rather than just the static weights of Section 4.2. Hence, we convert scores into *dynamic weights.* When converting, we invert the scores, because lower weights are better (whereas it is the opposite for scores). The conversion produces dynamic weights in the interval $[0, 1]$, to make them commensurate with the static weights. Formally, the dynamic weight of a node $v \in S$, denoted by $wn_d(v)$, is

$$wn_d(v) = 1 - \frac{\max_{u \in S} \mathit{score}(Q, u)}{\mathit{score}(Q, v)}. \tag{13}$$

Since $\mathit{score}(Q, v)$ is negative (i.e., obtained by applying logarithm to probabilities), $wn_d(v)$ is in the interval $[0, 1]$. Note that $wn_d(v) = 0$ if node v has the highest score in S.

The static weight of a path $p_{r \to v}$ is given by Equation (3). The *combined weight* of $p_{r \to v}$, denoted by $w_c(p_{r \to v})$, also incorporates the dynamic weights of r and v (while omitting their static weights). That is,

$$\begin{aligned} w_c(p_{r \to v}) = {} & wn_d(r) + wn_d(v) + \\ & + \sum_{e \in p} we_s(e) + \sum_{x \in p \wedge x \notin \{r, v\}} wn_s(x). \end{aligned} \tag{14}$$

Let r be a selected root. The set U_{ri} consists of all the keyword nodes that were selected for q_i and are in $r^\star$. Observe that every combination of m minimal-weight paths, such that each one is from r to a keyword node of U_{ri} ($1 \le i \le m$), yields an answer to $Q = (q_1, \ldots, q_m)$.[1] For each root r and keyword q_i, we generate these paths and keep them in a separate sorted list.

To generate answers, a priority queue A stores for each selected root r, the next best answer (with r as the root) that has not yet been added to the output (or discarded if it is not valid). Answers are removed from A by increasing height. Note that the *height* of a tree is the maximum combined weight over all the paths from the root to some leaf.

An answer a is valid if it satisfies the following conditions. First, a is non-redundant, that is, a does not have a proper subtree that also contains all the keywords of the query. If a is redundant, we convert it to a non-redundant answer by recursively removing the root r, thereby decreasing its height. Second, a is not a duplicate of another answer that is already in the output. Duplicates are removed based on an undirected semantics (to conform to the evaluation framework of Section 7). Third, a does not have a relationship node with fewer than two adjacent entity nodes.

[1]Formally, such a combination may not create a tree. However, it can be easily modified to form a tree.

6. ANSWER POTENTIAL FUNCTIONS

We view an answer a as a document by concatenating the instances of each field over all the nodes of a. Thus, a_{cnt} and a_{ttl} are ordinary documents obtained by concatenating the content and title fields, respectively, of all the nodes of a. Similarly to nodes, we define potential functions for unigrams, unordered bigrams and query-independent answer priors. These functions are used for scoring answers with respect to Q by means of Equation (1).

Unigrams. Analogously to Equations (6) and (7), we use the following two potential functions for the content and title fields, respectively.

$$f_T^a(q_i, a) = \ln\big((1 - \alpha_T^a) P(q_i | a_{cnt}) + \alpha_T^a P(q_i | V_{cnt})\big) \tag{15}$$

$$f_{\hat{T}}^a(q_i, a) = \ln\big((1 - \alpha_{\hat{T}}^a) P(q_i | a_{ttl}) + \alpha_{\hat{T}}^a P(q_i | V_{ttl})\big) \tag{16}$$

Recall that V_{cnt} and V_{ttl} are the collections comprising the content and title fields, respectively, of all the nodes of the data graph.

Unordered bigrams. Similarly to Equations (10) and (11), for an unordered bigram $\{q_i, q_{i+1}\}$ of Q, we define two potential functions for the content and title fields as follows.

$$\begin{aligned} f_U^a(q_i, q_{i+1}, a) = \ln\big(& (1 - \alpha_U^a) P(\{q_i, q_{i+1}\} | a_{cnt}) + \\ & + \alpha_U^a P(\{q_i, q_{i+1}\} | V_{cnt})\big) \end{aligned} \tag{17}$$

$$\begin{aligned} f_{\hat{U}}^a(q_i, q_{i+1}, a) = \ln\big(& (1 - \alpha_{\hat{U}}^a) P(\{q_i, q_{i+1}\} | a_{ttl}) + \\ & + \alpha_{\hat{U}}^a P(\{q_i, q_{i+1}\} | V_{ttl})\big) \end{aligned} \tag{18}$$

As in Section 4.4, we use the maximum likelihood estimate for the probabilities in the above equations. Since a_{cnt} and a_{ttl} are ordinary documents, we employ the usual term frequencies, rather than the weighted ones. We use Dirichlet smoothing for α_T^a, $\alpha_{\hat{T}}^a$, α_U^a and $\alpha_{\hat{U}}^a$ (see Section 7.3).

Query independent. We use one query-independent potential function f_L^a that is derived from an answer prior as follows. Let a_r be an answer. The subscript of a_r means that node r is the root. To derive the prior $P(a_r)$, we consider the *skeleton* s_r that is obtained by deleting the content of a_r. Namely, s_r has the same nodes and edges as a_r, but without attribute-value pairs.

To obtain the probability $P(a_r)$, we assume that a_r is generated in two steps. First, the skeleton s_r is generated with probability $P(s_r)$. Second, s_r is instantiated to a_r with probability $P(a_r | s_r)$; this is also done in two steps. First, the root of s_r is instantiated to a specific node of the data graph G. Second, the following is repeated. After instantiating a node v of s_r to some node v' of G, we select a neighbor of v' for each child of v.

The prior of an answer a_r is $P(a_r) = P(a_r | s_r) P(s_r)$. For simplicity, we assume that $P(s_r)$ is the same for all skeletons, so $P(a_r) = P(a_r | s_r)$. To compute $P(a_r | s_r)$, we make another simplifying assumption, namely, the probability of choosing a specific node u of a_r depends only on its parent, denoted by $par(u)$. In particular, the probability of choosing the root r of a_r depends only on the data graph G. And as in the derivation of Equation (12), it is proportional to the degree or r. Thus,

$$P(r) = \frac{Deg(r)}{\sum_{v \in V} Deg(v)}. \tag{19}$$

For a node $u \neq r$ of a_r, the probability of choosing u is proportional to its degree when compared with all the adjacent nodes of its parent. Hence,

$$P(u|par(u)) = \frac{Deg(u)}{\sum_{v \in N(par(u))} Deg(v)}, \tag{20}$$

where $N(par(u))$ consists of all the neighbors of $par(u)$. Therefore, the probability of a_r is

$$P(a_r) = P(r) \prod_{u \in a, u \neq r} P(u|par(u)). \tag{21}$$

Note that the product of the conditional probabilities is over all nodes u of a, except the root. Since an answer a is an undirected subtree, we can pick any one of its nodes as the root; hence, we define

$$f_L^a(a) = \ln \left(\max_{r \text{ is a node of } a} P(a_r) \right). \tag{22}$$

7. EXPERIMENTS

7.1 The Benchmark

We use the evaluation framework of [5] that was specifically developed for testing the effectiveness of systems for keyword search over data graphs. The framework consists of three datasets: IMDB, Wikipedia and Mondial.

The datasets are given as relations. Each tuple of those relations has a unique id and may have some foreign keys pointing to other tuples. IMDB and Wikipedia contain six relations each, and Mondial contains twenty four relations. IMDB has 1.6M tuples, Wikipedia has 206K tuples and Mondial—only 17K tuples. IMDB and Wikipedia contain relatively large chunks of text, while Mondial has only short strings, such as names of countries and cities, etc. The Mondial data graph has on average a large number of edges per node, compared with the other two datasets.

For each dataset, the evaluation framework of [5] has fifty queries and their qrels (i.e., query relevance judgments). The average number of keywords per query is 2.91, when excluding five queries of IMDB that consist of very long quotations from movies. The average number of answers per query is 4.49 and the largest number of tuples in an answer is 5.

7.2 System Implementation

We translated each dataset into a data graph, as explained in Section 3.1. Each data graph is indexed into three data structures. First, the *graph index* comprises the nodes and edges, and is kept in a Berkeley DB.[2] It is used for traversing the data graph and for storing query-independent information that is needed for computing the potential functions. The stored information includes, for example, the static weights and for each node v, the lengths of $v^\star_{cnt}$ and $v^\star_{ttl}$; the latter two are used in Equation (8).

[2]http://www.oracle.com/technetwork/products/berkeleydb

Table 1: Index sizes (in MB)

	raw data	graph index	node index	virtual index	
				$\tau = 1$	$\tau = 2$
Mondial	2	22.8	4.3	6.1	137
Wikipedia	220	330	316	299	23K
IMDB	304	1800	519	734	40K

The second data structure, called the *node index*, is an Apache Lucene[3] inverted index. It handles each node as a separate document consisting of the title and content fields, after applying stemming and stop-word removal.[4] We use the Lucene *Field* class to implement those two fields. We keep two instances of the inverted node index, one for unigrams and another for unordered bigrams. The node index also stores the full text of each node, because it is needed for building the virtual index that is described next.

The third data structure, called the *virtual index*, is a Lucene inverted index for the VDs (virtual documents) (one per node). Similarly to the node index, it has title and content fields, and two instances (for unigrams and unordered bigrams). The posting list of t keeps, as a Lucene *payload*, the weighted term frequency of t (see Equation (5)).

Table 1 gives the sizes of the indexes for each dataset.[5] For IMDB, the graph index is relatively big, due to the large number of entities and relationships in that dataset. The node index and virtual index for $\tau = 1$ have similar sizes. This is due to the fact that the node index also stores the full text of the title and content fields (rather than just the inverted lists). The virtual index for $\tau = 2$ is larger by two orders of magnitude than the one for $\tau = 1$.

The selection of roots and keyword nodes in Section 5 is efficiently implemented as follows. Let $Q = (q_1, \ldots, q_m)$ be the given query. In Lucene, we run Q as a boolean conjunctive query on the virtual index and find all nodes v, such that the VD $v^\star$ contains all the q_i. We rank those nodes by $score(Q, v)$ of Equation (1) and the potential functions of Section 4.4; the top-n are the selected roots. We select the top-n keyword nodes for each q_i ($1 \leq i \leq m$) as follows. Using the node index and the *Filter* tool of Lucene, we run a Lucene query to find all the nodes that contain q_i and are in the VD of some selected root, and then choose the top-n according to $score(Q, v)$.

7.3 Parameters Tuning

We use Equation (1) in three places. First, to select the top-n roots, second, to select the top-n keyword nodes and finally to rank the top-k answers. Thus, we have to learn the parameters λ_T, $\lambda_{\hat{T}}$, λ_U, $\lambda_{\hat{U}}$ and λ_L for each of the three cases. We use the coordinate ascent algorithm [20], as implemented in RankLib,[6] to learn the parameters.

We generated three labeled files—for roots, keyword nodes and answers—with positive and negative examples for each query as follows. The qrels in the benchmark [5] contain only

[3]`https://lucene.apache.org/`

[4]We used stop words from `http://www.textfixer.com/resources/common-english-words.txt`, added some negation words, such as can't and won't, and removed stop words that could be names of people, such as Will.

[5]The sizes of the node and virtual indexes are shown for unigrams. The unordered-bigram indexes are between two to eight times larger than those of the unigrams.

[6]http://sourceforge.net/p/lemur/wiki/RankLib/

the correct answers that are labeled with 1. For each node v of those answers, if v can be a root (i.e., its VD contains all the query keywords) or a keyword node, then it is added to the corresponding file with the label 1. To add negative examples, we ran our system with equal weights of 0.2 for all of the five parameters (when scoring roots, keyword nodes and answers). We used the default values of $n = 1,000$ and $k = 1,000$. Among the top-k answers obtained in this way, we chose those that are not in the qrels and labeled them with 0. Each node that can be a root or a keyword node of an answer that is not in the qrels was also labeled with 0 and added to the corresponding file, provided that it had not previously been labeled with 1.

Our experiments use cross validation. Namely, we learn the parameters on two datasets and use them on the third one. We report the results for each dataset with those learned parameters. The files with both 0 and 1 labels are just for training. The MAP is measured on the third dataset by using the original qrels of the evaluation framework.

We use Dirichlet smoothing with the parameter $\frac{\mu}{\mu+|x|}$, where $|x|$ is the length of x. For α^n_T, $\alpha^n_{\hat{T}}$, α^n_U and $\alpha^n_{\hat{U}}$ in Equations (6) and (10), the variable x ranges over $v^\star_{cnt}$, whereas in Equations (7) and (11), it ranges over $v^\star_{ttl}$. The parameter μ is the average of $|x|$ over all $v^\star_{cnt}$ and $v^\star_{ttl}$ when smoothing $v^\star_{cnt}$ and $v^\star_{ttl}$, respectively.

For the parameters α^a_T, $\alpha^a_{\hat{T}}$, α^a_U and $\alpha^a_{\hat{U}}$ in Equations (15) and (17), the variable x ranges over the a_{cnt} field of the top-k answers, whereas in Equations (16) and (18), it ranges over their a_{ttl} field. We view an answer a as an ordinary document. But since answers have a variable number of nodes, we define the length of a_f (where f is either cnt or ttl) as the average per node; namely, $\frac{1}{s}\sum_{t\in a_f} tf(t,a_f)$, where s is the number of nodes of a. When smoothing a_f, the parameter μ is the average of $|v_f|$ over all nodes v of the data graph; similarly for unordered bigrams.

The diameter of VDs is $\tau = 1$ (see Section 4.1). Unless otherwise specified, when selecting the top-n roots and keyword nodes for each q_i (see Section 5), we use $n = 1,000$.

7.4 The Setup of the Experiments

We compare our approach with the top systems, namely, BANKS [2], Bidirectional [13] and CD [6], among those tested in [5], as well as with GraphLM [19]. In [7], they extended the binary relevance set of [5] to include answers with marginal relevance. However, they have not made that extended framework available, so we cannot use it. Similarly to [5,19], we use the default $k = 1,000$ when producing the top-k answers for each query.

7.5 Comparison with the State of the Art

We compare our approach, called *MRF-KS*, with the state-of-the-art systems using the evaluation framework of [5].[7]

Figure 2 shows that MRF-KS outperforms the other systems on each of the three datasets.[8] The second best is GraphLM. On Wikipedia, MRF-KS achieves a MAP of 0.76 compared with 0.63 for GraphLM (an improvement of 20%). On IMDB, MRF-KS has a MAP of 0.76 compared with 0.68 for GraphLM (an improvement of 11.3%). And on Mondial, MRF-KS has a MAP of 0.89 compared with 0.83 for GraphLM (an improvement of 7.6%).

The advantage of MRF-KS over the second best system GraphLM is statistically significant on all three datasets, as measured in a one-tailed t-test (p-value < 0.05). We exclude the work of [3] from this comparison due to the following. Some essential details (e.g., the algorithm for generating answers in ranked order) are missing from their paper, which has made it impossible for us to reproduce their results. Moreover, they declined our request to get their code or, at least, the AP (average precision) they obtained for each individual query of the evaluation framework of [5]; hence, verifying their results is problematic. In any case, they reported that for Mondial, Wikipedia and IMDB, they got a MAP of 0.9, 0.78 and 0.79, respectively. Our results are almost the same: 0.89, 0.76 and 0.76, respectively.

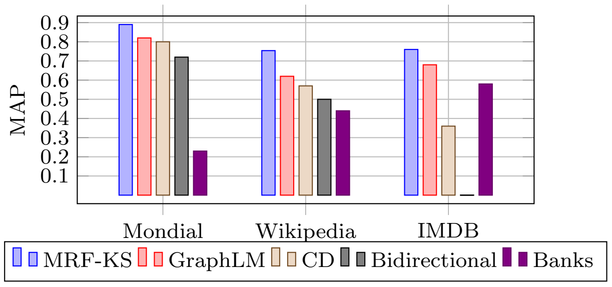

Figure 2: MAP of MRF-KS and state-of-the-art systems ($k = 1,000$, $n = 1,000$)

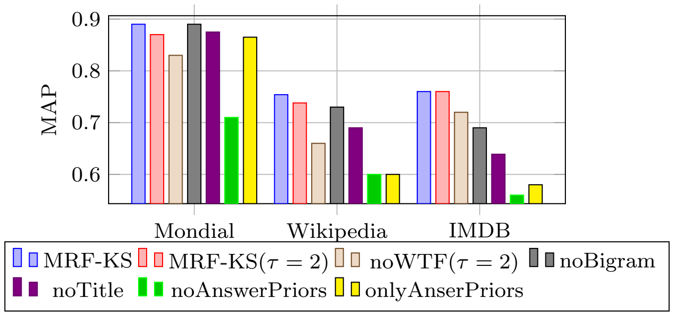

Figure 3: Effect of individual features ($k = 1,000$, $n = 100$)

7.6 The Effect of Different Components

We now show the effect of various components of our system by measuring the MAP yielded by different configurations, as shown in Figure 3. Unless otherwise specified, VDs have the default diameter of $\tau = 1$. To magnify the effect of the different configurations, we select a relatively small number ($n = 100$) of roots and keyword nodes. For each configuration, we relearned the relevant parameters on two datasets and then measured the MAP on the third one, as explained in Section 7.3.

The first two columns show that increasing the diameter τ of VDs from 1 to 2 slightly lowers the MAP. The third column (labeled with noWTF) is when the diameter is 2 and ordinary term frequencies (instead of the weighted ones) are used. In this case, the MAP drops on all three datasets and the larger effect is on Wikipedia (from 0.74 to 0.66).

[7]For all the systems, the MAP on IMDB is based on the updated qrels that appear in `http://www.cs.virginia.edu/~jmc7tp/resources.php#search`.

[8]We show only the top performing state-of-the-art systems.

The rest of the columns show an ablation test that disables one feature at a time. The column *noBigram* is when disabling the unordered bigrams in the content and title fields (setting $\lambda_U, \lambda_{\hat{U}} = 0$) for roots, keyword nodes and answers. The column *noTitle* denotes the effect of disabling the title field for unigrams and unordered bigrams (setting $\lambda_{\hat{T}}, \lambda_{\hat{U}} = 0$) for roots, keyword nodes and answers. The column *noAnswerPriors* is when using all features except the answer priors (setting $\lambda_L = 0$). The last column *onlyAnswerPriors* shows the MAP when using only answer priors for ranking the answers (setting $\lambda_T, \lambda_U, \lambda_{\hat{T}}, \lambda_{\hat{U}} = 0$ for answers and keeping all features for roots and keyword nodes).

We can see that the most dominant feature is the answer priors. When disabled, the MAP drops sharply on all three datasets. For example, on Wikipedia it drops from 0.75 to 0.6 and on IMDB—from 0.76 to 0.56. The contribution of the answer priors is statistically significant. The title field has the second largest effect on IMDB and Wikipedia. The unordered bigrams have a moderate effect on Mondial and Wikipedia, but a larger one on IMDB.

7.7 Efficiency vs. Effectiveness

In this section, we show that our method offers a useful trade-off between effectiveness and efficiency that can be easily tuned, thereby substantially improving the running time while only slightly lowering the MAP. This ability is hardly found in any other system. Figure 4 gives the average running time per query and the MAP for different values of n (i.e., the number of selected roots and keyword nodes). The results are for $k = 100$ rather than the default $k = 1,000$, thereby making the results more significant.

Figure 4 shows that the running time drops at a much faster rate than the MAP, as n gets smaller. On the three datasets, even for the small value of $n = 50$, the MAP is almost the same as for $n = 1,000$. Hence, the parameter n enables us to substantially increase the efficiency without sacrificing effectiveness.

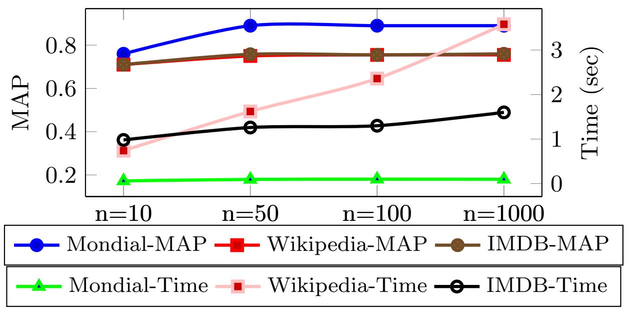

Figure 4: Effect of n (number of selected roots and keyword nodes) for $k = 100$

8. CONCLUSIONS

We presented a novel approach, couched in probability theory, to finding the top-k answers in keyword search over data graphs. It is based on new ideas and concepts. First, we showed how to estimate the prior of an answer (i.e., subtree) and showed its contribution to the final ranking of answers. Second, we defined virtual documents with weighted term frequencies and showed their effectiveness in selecting the most promising roots and keyword nodes of candidate answers. Third, we presented an efficient algorithm for generating and ranking answers. The ranking of nodes (i.e., roots and keyword nodes) and answers is based on a combination of query-dependent and independent features.

We compared our approach with other systems on the evaluation framework of [5] that consists of three datasets: IMDB, Wikipedia and Mondial. In terms of MAP, our approach has a statistically significant advantage over other tested systems on each of these datasets. For example, on Wikipedia, we achieved an improvement of 20% compared with the best state-of-the-art system.

We performed an extensive analysis of the contribution of the various components. The most significant feature is the answer priors. We further showed that that the MAP remains almost the same even when selecting a small number of keyword nodes and roots, thereby reducing the search space and increasing the efficiency.

9. ACKNOWLEDGMENTS

The authors thank Oren Kurland for helpful comments.

10. REFERENCES

[1] M. Bendersky, W. B. Croft, and Y. Diao. Quality-biased ranking of web documents. In *WSDM*, 2011.
[2] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.
[3] V. Bicer, T. Tran, and R. Nedkov. Ranking support for keyword search on structured data using relevance models. In *CIKM*, 2011.
[4] R. Blanco, P. Mika, and S. Vigna. Effective and efficient entity search in rdf data. In *ISWC*, 2011.
[5] J. Coffman and A. C. Weaver. A framework for evaluating database keyword search strategies. In *CIKM*, 2010.
[6] J. Coffman and A. C. Weaver. Structured data retrieval using cover density ranking. In *KEYS*, pages 115–126, 2010.
[7] J. Coffman and A. C. Weaver. Learning to rank results in relational keyword search. In *CIKM*, 2011.
[8] S. Elbassuoni and R. Blanco. Keyword search over rdf graphs. In *CIKM*, 2011.
[9] B. He and I. Ounis. Combining fields for query expansion and adaptive query expansion. In *Information Processing and Management*, volume 43, pages 1294–1307, 2007.
[10] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: Ranked keyword searches on graphs. In *SIGMOD*, 2007.
[11] D. Himestra. Statistical language models for intelligent XML retrieval. In *Intelligent Search on XML Data, LNCS 2818*. Springer-Verlag Berlin Heidelberg, 2003.
[12] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.
[13] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, and R. Desai. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.
[14] J. Kamps, G. Mishne, and M. de Rijke. Language models for searching in Web corpora. In *TREC*, 2004.
[15] J. Y. Kim and W. B. Croft. A field relevance model for structured document retrieval. In *European Conference on Information Retrieval (ECIR)*, pages 97–108, 2012.
[16] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. In *Mach Learn*, volume 81, pages 53–67, 2010.
[17] Y. Luo, X. Lin, W. Wang, and X. Zhou. Top-k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.
[18] Y. Lv and C. Zhai. Positional language models for information retrieval. In *SIGIR*, 2009.
[19] Y. Mass and Y. Sagiv. Language models for keyword search over data graphs. In *WSDM*, pages 363–372, 2012.
[20] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *SIGIR*, 2005.
[21] Q. Su and J. Widom. Indexing relational database content offline for efficient keyword-based search. In *IDEAS*, pages 297–306, 2005.