Language Models for 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 includes the given keywords. This paper focuses on improving the effectiveness of that type of search. A novel approach that combines language models with structural relevance is described. The proposed approach consists of three steps. First, language models are used to assign dynamic, query-dependent weights to the graph. Those weights complement static weights that are pre-assigned to the graph. Second, an existing algorithm returns candidate answers based on their weights. Third, the candidate answers are re-ranked by creating a language model for each one. The effectiveness of the proposed approach is verified on a benchmark of three datasets: IMDB, Wikipedia and Mondial. The proposed approach outperforms all existing systems on the three datasets, which is a testament to its robustness. It is also shown that the effectiveness can be further improved by augmenting keyword queries with very basic knowledge about the structure.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search processes*

General Terms

Algorithms, Experimentation, Performance

Keywords

Data graphs, language models, semantic weights, ranking

1. INTRODUCTION

Keyword search over free text is a very common and important tool. However, the emergence of large knowledge bases, such as DBPedia,[1] Yago,[2] Freebase,[3] the CIA World-Factbook,[4] to name just a few, enables a different type of search, where a user can get precise and focused answers to her information need. This is much preferable to keyword search that can only provide the user with a list of relevant documents that might contain the answers somewhere.

In general, knowledge bases contain information about entities and their relationships. In practice, they are typically represented as RDF triples consisting of a *subject* and an *object* that are entities connected through a *property*. Knowledge bases can also be stored in relational databases, which represent entities and their relationships by means of tuples and foreign keys.

Those representations can be transformed to data graphs, where nodes represent entities and edges correspond to relationships. Additional free text can be added to each entity or relationship, resulting in a richer representation. That is, the graph structure represents the semantics of entities and of the relationships among them, while the free text helps in deciding the relevancy of a particular entity or relationship to a given query.

Such data graphs enable a shift in the paradigm of search, from a document as the basic returned unit to a finer granularity of entities. An answer can be a single entity or a set of entities connected through some relationships. There are already TREC and INEX tracks for returning entities instead of full documents.

Consider, for example, a user who needs to find the language spoken in Poland. A traditional search engine returns documents about Poland that might contain the needed information. But if we have both Poland and Polish as entities of the types *country* and *language*, respectively, and these two entities are connected by an edge, then we can return Polish as the spoken language of Poland.

The main drawback is that querying data graphs requires a nontrivial understanding of their underlying schemas. Even RDF, which is regarded as having no schema, requires a nontrivial query language, such as SPARQL. For databases, we need to use SQL as the query language. Both SPARQL and SQL are too difficult for end users.

To overcome this problem, a lot of research has been conducted in recent years on efficient and effective methods for keyword search over data graphs. The idea is that users

*This work was partially supported by The German-Israeli Foundation for Scientific Research & Development (Grant 973–150.6/2007).

WSDM'12, February 8–12, 2012, Seattle, Washington, USA.

[1] http://dbpedia.org
[2] http://www.mpi-inf.mpg.de/yago-naga/yago/
[3] http://freebase.com
[4] https://www.cia.gov/library/publications/the-world-factbook/index.html

can still pose queries consisting merely of keywords, and the system exploits the semantics encoded in the data graph in order to return entities and relationships as answers.

In keyword search over data graphs, an answer is a non-redundant subtree that includes the given keywords. Non-redundancy means that an answer does not have a proper subtree that also contains all the keywords. There are two main challenges. The first is to develop an algorithm for efficiently generating answers. In this paper, we use the algorithm of [9]. To apply this algorithm, we first have to create a node for each keyword q_i of the given query, and then add edges from q_i to all the nodes of the data graph that contain q_i.

The second challenge is to find methods for effectively ranking the generated answers according to their relevance to the user. This challenge is the focus of our work. There are two main approaches regarding how to rank answers: IR methods and proximity search. In the former, IR techniques are used, whereas in the latter the idea is to minimize the structural (i.e., semantic) weights of answers. Some papers have tried to combine IR methods with structural weights into a unified approach to the task of ranking answers.

In this paper, we propose a novel ranking method that is based on language models as well as structural weights. Our approach consists of the following steps. First, we assign structural weights to the nodes and edges of the data graph. These weights are derived only from semantic considerations and are *static*, namely, they are independent of any particular query. Second, language models are used to assign weights to the edges that connect the keywords of a given query to the nodes that contain them. These weights are *dynamic*, that is, they depend on the given query. Third, an existing algorithm [9] returns candidate answers based on their weights. Finally, the candidate answers are re-ranked by creating a language model for each one.

The contributions of this paper are the following. First, we describe a novel and effective ranking technique that combines language models with structural weights. Second, we show that our ranking model is robust by testing it on the first (and only) standard benchmark for keyword search over data graphs, which was recently developed by [4]. This benchmark consists of three datasets: IMDB, Wikipedia and Mondial, as well as fifty queries for each one. On each of these datasets, our approach outperforms in terms of MAP (mean average precision) all the existing systems that have been tested so far on this benchmark. Additional tests, using other metrics, also support the robustness of our approach. Finally, we show that the effectiveness can be further improved by augmenting keyword queries with very basic knowledge about the structure of the data graph.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we describe the data model. In Section 4, we describe how to rank answers (which are subtrees) based on a combination of language models and semantic considerations. Section 5 describes the implementation and experiments that verify the effectiveness of our approach. We conclude in Section 6.

2. RELATED WORK

A survey of systems for keywords search over data graphs is given in [4]. There are two main approaches for ranking answers: IR methods and proximity search. In the former IR techniques are used, whereas in the latter the idea is to minimize the weights of trees that contain the keywords of the query. In this paper, we focus on the effectiveness of the search. Thus, we discuss that aspect of the state-of-the-art, and ignore considerations related to efficiency.

The systems of [2, 12] were the first to introduce keyword search over databases. An answer in both systems consists of tuples that contain all the keywords of the query. In [12], they construct an execution plan for generating networks of tuples that together contain all the query keywords. Both [2, 12] use a very simple ranking model that counts the number of tuples in an answer (the fewer, the better).

The systems that rank answers by means of IR methods are [11, 15, 5, 16]. The work of [11] extends the scoring model of [12] as follows. Each tuple in an answer gets an IR score that is produced by some underlying inverted index on columns that contain text attributes. That index could be provided, for example, by an RDBMS. The IR score of a set of tuples is defined as the sum of the IR scores of the individual tuples divided by the number of tuples. Thus, it is the average IR score per tuple of an answer. The work of [15] extends the work of [11] by introducing several normalization factors that are used for combining the individual IR scores into an IR score of an answer. The work of [5] uses a cover density ranking. In [16], instead of combining individual tuple scores as in [11], they view a whole answer as a *virtual document* and use IR methods, such as *tf* and *idf*, to assign a relevance score. They also apply some normalization factors; for example, a *size normalization factor* that depends on the number of tuples in an answer (the fewer tuples, the better). In this work, we assign IR scores to nodes and edges as well as to full answers. We combine the IR scores with structural weights that reflect the semantic strength of answers.

The systems of [3, 13, 7, 10, 9] employ the second approach that assigns weights to the nodes and edges of the data graph, and then ranks answers by increasing weight. In [3], they find answers by starting instances of Dijkstra shortest-path algorithm from several nodes that (between them) contain the keywords of the query. When all these instances meet in a node r, they find an answer (having r as its root). In [3], weights are assigned to nodes based on their *prestige.* The work of [13] improves the efficiency of [3] by allowing forward search from potential roots toward leaves, in addition to the backward search of [3]. In [7], they produce answers based on an algorithm for finding the minimum group Steiner tree. They extend this algorithm so that it can find some more trees, in additional to the minimal one. In [9], the emphasis is on provable properties. Their algorithm enumerates answers with polynomial delay in a 2-approximate order by increasing height. That algorithm can provably find all the non-redundant subtrees (of the data graph) that contain the keywords of the query. In [10], the focus is on efficiency which is achieved by keeping, in a bi-level index, some of the shortest paths that have already been computed. They mention that their scoring function for answers combines both the graph structure and content scoring using IR methods. However, since their focus is on efficiency, they do not give details on their scoring function. In this paper, we present a ranking method that uses structural weights and combines them in a novel way with IR scores that are derived by means of language models.

The work of [8] has developed a language-model approach to SPARQL queries that are augmented with keywords. (The evaluation is, of course, over RDF graphs). They use a *wit-*

ness count for each triplet to indicate the number of times that the triplet was extracted from some text. Then they build a language model for a given query as well as a language model for each answer. Ranking is done according to the Kullback-Leibler (KL) divergence between the language models of the query and each answer. This work requires queries to be formulated in SPARQL, so it does not fall under the category of keyword search over data graphs.

Keyword search is attractive, since it does not require any knowledge of the underlying schema of the data graph. However, it still has its limitations, because the user cannot precisely define her information need. Several papers [17, 6, 14, 18] have tried to bridge the gap between fully structured queries and keyword search by letting the user refine her query. The work of [17] suggests an interactive framework, where the user starts with keyword search and then, through an exploration phase, she can focus her query by refining and filtering. Finally, she can also aggregate results that are either similar or complementary. The work of [6] suggests a UI, where the user starts with keyword search and the system presents several interpretations for the query. The user can select an interpretation and the system then formulates a more structured query. Similarly, the work of [14] allows the user to start with keyword search, and then it selects the best interpretation and presents the results to the user. The work of [18] suggests a new query language that is based on keyword search, but with a simple notation for structural constraints using conjunctions and nesting with relations. In this paper, we show that the effectiveness of keyword search can be increased significantly by using very simple syntax that expresses basic knowledge about the schema of the data graph.

3. THE DATA MODEL

Data is stored in a directed graph. Each *node* and *edge* has a set of attribute-value pairs. Some values are short (e.g., an integer or a character string) while others comprise free text. Two special attributes are *name* and *type*. The name is a short string that serves as a not-necessarily-unique identifier; for example, the name could be the title of an article or the name of a country. The type denotes the class to which the given node or edge belongs; for example, the type could be article or country. There can be several edges between a pair of nodes. For example, the graph can represent a social network, where nodes are people and the edges correspond to relationships between people, such as friendship, business associations, etc. In summary, a node or an edge can have any number of attributes, and each one has a value.

The attributes of a node or an edge describe semantic information. We group the attributes into three semantic *fields* (that could overlap). The *title* field comprises the values of the name and type attributes. The *content* field of either a node or an edge consists of all its attributes and their values. Finally, the *structure* field captures the structural semantics of the given node or edge. It contains the value of the type attribute as well as all the other attributes, but without their values.

The title and content fields (of a node or an edge) contain textual descriptions 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. The structure field does not carry a textual description. Typically, terms that appear in the structure field are repeated in many nodes or edges. For example, in the IMDB dataset, "movie" is the value of the type of all the movie nodes; hence, it appears in all them. We use the structure field to improve efficiency. In particular, occurrences of query keywords in the structure field are handled differently, as we explain in Section 4.4.

A data graph is usually derived from a database or an XML file. For example, when creating a data graph from a relational database, each tuple t becomes a node v_t. The value of the type attribute of v_t is the relation name to which t belongs. The attribute-value pairs of the node v_t are those of the tuple t, excluding foreign keys. That is, a foreign key is represented in the data graph by an edge (hence, the corresponding attribute-value pair is not included in the node).

A query $Q = (q_1, ..., q_n)$ on the data graph is given as a set of keywords. Each q_i should match a term in the content field of some node or edge (i.e., we use the AND semantics).

An *answer* to a query $Q = (q_1, ..., q_n)$ 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 the content field of some node or edge. Non-redundancy means that an answer does not have a proper subtree that also contains all the keywords of the query. Note that non-redundancy does *not* imply minimality, and a query could have a large number of answers. As an example, consider the query "movie, love." An answer to this query is not necessarily a single node. There could be a subtree with one node that has "movie" in its structure field (e.g., as the value of its type) and with another node that contains "love" in its content field.

We use a system [9] that handles attribute-value pairs in edges by adding nodes that are called *connectors* [1]. Due to a lack of space, we omit the technical details and simply assume that only nodes have attribute-value pairs.

4. RANKING ANSWERS

Two principles guide us when ranking answers by their relevance to the query. First, a smaller subtree typically represents a stronger semantic connection between its nodes. Second, the relevance of an individual node or edge depends on its text; in particular, to be relevant, the text should include (at least some of) the keywords of the query. Thus, we would prefer subtrees that are small, and have nodes and edges that are highly relevant to the keywords of the query. These two requirements are not congruent. In other words, even in a highly relevant subtree, some nodes and edges may not contain any keyword of the query. Similarly, a relevant subtree is not necessarily small. In this section, we describe how to rank subtrees based on a combination of language models and structural weights.

4.1 Language Models

Language models (abbr. LM) for IR define a probability space for each document d of a given collection. The probability space of d, called the *document model*, measures the probability to generate a term t given the document d. Sometimes a document has multiple *fields*, that is, it can be viewed in different ways. For example, we may consider just the title of each document, or alternatively, take the whole document into account. For a given field f, we use d_f to denote that only the text of field f is taken into account when referring to the document d. For example, d_{title} means that

only the text in the title of d is considered. The field *content* means that all of the text (including the title) is considered.

We assume that query terms are independent of one another and use the Jelinek-Mercer smoothing [19]. Thus, for a query $Q = (q_1, ..., q_n)$ and field f of document d, we get

$$P(Q|d_f) = \prod_i ((1-\lambda)P(q_i|d_f) + \lambda P(q_i|C)) \quad (1)$$

where C is the whole collection and $0 \leq \lambda \leq 1$ is a smoothing parameter. The smoothing is done over the whole collection C, rather than just its field f (i.e., C_f), because some fields (e.g., title) may not have enough text to make the smoothing effective. Typically, the probabilities on the right side of Equation (1) are determined by the maximum-likelihood estimate, namely,

$$P(q_i|X) = \frac{tf(q_i, X)}{\sum_{t \in X} tf(t, X)}, \quad (2)$$

where X is either d_f or C, and $tf(t, X)$ is the term frequency of t in X.

4.2 Language Models for Data Graphs

Recall that an answer A is a subtree of a given data graph G. We now describe how to extend the above language model to answers that are graphs rather than documents. In doing so, there is no need to assume that A is a subtree of G; rather, A could be any subgraph of G.

Consider an answer $A = (V, E)$, where V is the set of nodes and E is the set of edges. The text associated with A, denoted by text(A), is obtained by concatenating the texts of all the nodes and edges of A. Similarly to documents, A can have multiple fields. For example, the title field of A, denoted by A_{title}, is the graph A under the restriction that in each node and edge, only the title field of the text is considered. Thus, text(A_{title}) is the result of concatenating the title fields of all the nodes and edges of A.

Analogously to Equation (1), the following is the probability that a query $Q = (q_1, ..., q_n)$ is generated by a field f of an answer A (i.e., by A_f).

$$P(Q|A_f) = \prod_i ((1-\lambda)P(q_i|A_f) + \lambda P(q_i|G)) \quad (3)$$

As explained earlier, the smoothing term considers the whole collection, which in this case is the data graph G.

We need to estimate the probability $P(q_i|X)$, where X is either A_f or G. Ideally, both the graph structure and the text of X should be taken into account. However, developing language models that consider the graph structure is beyond the scope of this paper, and is left for future research. In this section, we only deal with the text, and the graph structure is handled in Section 4.4. Thus, we estimate the probabilities on the right side of Equation (3) similarly to Equation (2), that is,

$$P(q_i|X) = \frac{tf(q_i, \text{text}(X))}{\sum_{t \in \text{text}(X)} tf(t, \text{text}(X))}, \quad (4)$$

where X is either A_f or G, and $tf(t, \text{text}(X))$ is the term frequency of t in text(X).

4.3 Normalization

In this work, language models are used for ranking nodes and answers (which are subtrees of the data graph). The former are just a special case of the latter, because a node is a subtree. Alternatively, a node v can be viewed just as a document that consists of the text in v. Thus, Equations (1) and (3) apply equally to nodes. In order to deal uniformly with nodes and answers, we will use Equation (3) for both.

As explained later, we combine probabilities of language models with weights of semantic connections. A higher probability means that the answer is more relevant, whereas a lower weight indicates a stronger semantic connection. Therefore, to combine probabilities with weights, we need to normalize them to the interval $[0, 1]$; and in the case of probabilities, we should also invert them. Consequently, for both of them, lower normalized values are preferable (i.e., indicate higher relevance). This section describes how we process probabilities. Weights are discussed later.

When using Equation (3) for ranking, we actually apply it twice: once with *title* as the field f and then with *content* as f. Sometimes, the former probability is considerably higher than the latter. Thus, we normalize each of these two probabilities before taking their linear combination. Another consideration is to avoid underflow when taking the product of many probabilities. We do it by applying logarithm to Equation (3), thereby summing instead of multiplying. Note that taking the logarithm does not effect the ranking order of relevant nodes or answers. Next, we describe in detail how the normalization is done.

As already noted, we start by applying logarithm to Equation (3). The result is called the *IR relevance* of A_f with respect to the query Q, denoted by $R(Q, A_f)$. Thus,

$$R(Q, A_f) = \sum_i (\ln((1-\lambda)P(q_i|A_f) + \lambda P(q_i|G))), \quad (5)$$

where f is either *content* or *title*, and A is either a single node or a whole subtree, depending on what we rank.

As explained later, we apply Equation (5) to some elements (which are either nodes or answers) and choose the top-K. Let Γ be the set of those top-K elements. Suppose that $R_{\max}$ is the maximum of Equation (5) over all the elements of Γ, namely, $R_{\max} = \max_{A \in \Gamma}\{R(Q, A_f)\}$. Note that $R_{\max} \leq 0$, because the $R(Q, A_f)$ are obtained by taking logarithm of probabilities. The values obtained for the elements of Γ are in the interval $(-\infty, R_{\max}]$, where $R_{\max} \leq 0$. We need to normalize those values to the interval $[0, 1]$; and since we want lower values to be preferable, $R_{\max}$ should become 0 and $-\infty$ should become 1. So, we apply the following transformation that computes the *IR l-score*[5] of A_f with respect to the query Q, denoted by $lscr^{ir}(Q, A_f)$.

$$lscr^{ir}(Q, A_f) = 1 - \frac{1}{\ln(-R(Q, A_f) + R_{\max} + 2.718)} \quad (6)$$

We use logarithm in the above equation so that $lscr^{ir}(Q, A_f)$ will not grow too fast as $R(Q, A_f)$ gets smaller (i.e., more negative). Note that the values of $lscr^{ir}(Q, A_f)$ are in the interval $[0, 1]$. Furthermore, smaller values of $lscr^{ir}(Q, A_f)$ are better, namely, more relevant.

By using Equation (6), we obtain two normalized scores: $lscr^{ir}(Q, A_{content})$ and $lscr^{ir}(Q, A_{title})$ for the fields *content* and *title*, respectively. The next equation computes the IR l-score of A with respect to the query Q as a linear combi-

[5]The name *l-score* emphasizes that lower scores are better.

nation of the two, where $0 \leq \alpha \leq 1$.

$$lscr^{ir}(Q, A) = \alpha \cdot lscr^{ir}(Q, A_{title}) + (1-\alpha) lscr^{ir}(Q, A_{content}) \quad (7)$$

The above equation enables us to control the weight we give to the title field compared with that of the content field.

The normalization of Equation (6) depends only on $R_{\max}$ (but not on the minimum in the set of values that is normalized). Therefore, we can efficiently find the top-K elements (answers or nodes) according to Equation (7), even though each of $lscr^{ir}(Q, A_{title})$ and $lscr^{ir}(Q, A_{content})$ is normalized before it is used in Equation (7).

4.4 Graph Weights

We assign weights to the nodes and edges of a data graph based on two criteria: semantic strength and relevance to the given query. The former induces static weights (i.e., they do not depend on the query), whereas the latter determines dynamic weights.

The static weights are assigned to the graph that is generated from the original data, as explained in Section 3. These weights reflect the semantic strength of the nodes and edges. In this context, lower weights are better, because a smaller answer (i.e., subtree) is deemed more meaningful.

When processing a given query Q, nodes for the keywords of Q are created and connected to nodes of the graph that are relevant to Q. The edges that emanate from (the nodes for) those keywords get dynamic weights that are obtained from the IR l-scores of Equation (7). Recall that IR l-scores, just like weights, are better when they are smaller.

We start by discussing the static weights. The naive approach of assigning equal weights to all the nodes and edges is not effective [4]. We assign static weights as follows.

Static weights of nodes. The weight of a node v, denoted by $w(v)$, is static and depends on its incoming edges. Intuitively, nodes with many incoming edges are more important and, hence, should be assigned lower weights (which are better than higher ones).

Formally, $w(v)$ is defined as follows. Let $idg(v)$ be the in-degree of node v. If $idg(v) > 0$, then

$$w(v) = \frac{1}{\ln(1.718 + idg(v))}\,. \quad (8)$$

Otherwise (for isolated nodes or nodes with no incoming edges), we set $w(v) = 1$ which is the largest weight.

The weight of a node v depends inversely on the number of edges that point to v. The more incoming edges there are, the lower the weight is, which means that nodes with many incoming edges are more important. We use logarithm in the above formula so that the weight will not decay too fast as the number of edges increases. Due to the constant 1.718, the value $w(v)$ is in the interval $[0, 1]$. Nodes with a single incoming edge have weight 1 (namely, they are the least important). Our way of assigning weights to nodes is similar to *node prestige* [3]; however, they use some normalization while our weights are already in the interval $[0, 1]$.

Static weights of edges. When creating a graph from the original data source (see Section 3), each edge $e = (u, v)$ is assigned a static weight $w(e)$ that depends on the number of similar edges [9]. By definition, an edge e' is *similar* to e if at one end they have the same node and at the other end they have the same type. Intuitively, an edge with fewer similar edges is more unique and, hence, describes a stronger semantic relationship between its nodes.

Formally, let $fdg(e)$ be the number of edges that have the same type as e and are from u to nodes with the same type as v. Similarly, let $tdg(e)$ be the number of edges that have the same type as e, emanate from nodes with the same type as u and point to v. Note that e itself is counted in both $fdg(e)$ and $tdg(e)$. Then,

$$w(e) = 1 - \frac{1}{\ln(0.718 + fdg(e) + tdg(e))}\,. \quad (9)$$

The larger the number of edges that are similar to e, the higher the weight of e. Thus, an edge is more important if it has only a few similar edges. The logarithm is used so that the weight will not grow too fast as the number of similar edges increases. The constant 0.718 guarantees that $w(e)$ is in the interval $[0, 1]$. In particular, $w(e) = 0$ when e has no similar edges (other than itself), because in this case, $fdg(e) = 1$ and $tdg(e) = 1$.

Dynamic weights of edges. In search over data graphs, some keywords target the structure field while others are aimed at the content field. For example, a user who wants to find Wikipedia pages that were modified by AlleborgoBot would pose the query "page AlleborgoBot." The keyword "page" targets the type of a relevant node (which is in the structure field) while "AlleborgoBot" is aimed at the title or content of (possibly) another node. When a keyword (e.g., "page") may appear either in the structure field or only in the content field, we prefer occurrences in the former.

To generate answers for a query $Q = (q_1, ..., q_n)$, a node is created for each q_i and connected to all nodes v that contain q_i. The node for q_i has weight 1 (which is actually equivalent to the rule for static weights, because that node does not have any incoming edges). The weight of the edge from (the node for) q_i to v depends on whether q_i appears in the structure field or not; the aforementioned example is a motivation for doing so. If q_i appears in the structure field of v, then the weight is 0 (i.e., the best). If q_i appears in the content field of v but not in its structure field, the weight of the edge e from q_i to v is the IR l-score of v with respect to the whole query Q (rather than just q_i), as given by Equation (7).

For efficiency reasons, we do not connect q_i to all the nodes containing it, but only to those that are most relevant. We apply a two-step process as follows. First, for each keyword q_i, we use Equation (7) to select the top-K nodes among those having q_i in the content (which includes the title) field, but not in the structure field. (K is a parameter of the system.) We connect q_i to each of the selected nodes. The weight of the connecting edge is the IR l-score of Equation (7). We say that a node is *relevant* if it is connected to some keywords in the first step.

In the second step, we connect each q_i to all nodes v, such that v contains q_i in its structure field and is in the vicinity of some relevant node, namely, within a distance[6] r. We call r the *distance parameter*. The connecting edge has weight 0. It could be (e.g., the query "movie" over IMDB) that there are no relevant nodes at the end of the first step (that is, whenever a keyword appears in a node, it is also included in the structure field). In this case, we connect every q_i to the top-K nodes (according to Equation (7)) that contain q_i. The connecting edges have weight 0. Note that the structure field is a subset of the content field, so q_i appears

[6]The *distance* between two nodes is defined as the number of edges on the shortest path that connects them.

in the content field of the nodes to which it is connected, and it could also be in the title field of some of those nodes.

4.5 Scoring Answers

We now describe how to rank answers by combining the IR l-scores of Equation (7) with the graph weights of the previous section. First, we define the *structural weight* of an answer $A = (V, E)$, denoted by[7] $W(Q, A)$, as the sum of weights of all its nodes and edges, that is,

$$W(Q, A) = \sum_{v \in V} w(v) + \sum_{e \in E} w(e)\,. \qquad (10)$$

Before combining IR l-scores with structural weights, we need to normalize the latter (the former were already normalized in Section 4.3). Let $W_{\min} = \min_{A \in \Gamma}\{W(Q, A)\}$ be the minimum structural weight over all the generated answers. The structural weights are in the interval $[W_{\min}, \infty)$, where $W_{\min} \geq 0$. We need to normalize those weights to the interval $[0, 1]$. As a result, $W_{\min}$ will become 0 and ∞ will become 1. We do it by applying the following transformation that computes the *structural l-score* of an answer A with respect to the query Q, denoted by $lscr^s(Q, A)$.

$$lscr^s(Q, A) = 1 - \frac{1}{\ln(W(Q, A) - W_{\min} + 2.718)} \qquad (11)$$

We use logarithm in the above equation so that the structural l-score will not grow too fast as $W(Q, A)$ gets larger. Note that lower structural l-scores are preferable. Clearly, the values of $lscr^s(Q, A)$ are in the interval $[0, 1]$.

Now, we combine the IR l-score of Equation (7) with the structural l-score of Equation (11) as follows. The *l-score* of an answer A with respect to a query Q, denoted by $lscr(Q, A)$, is given by the linear combination

$$lscr(Q, A) = \beta \cdot lscr^s(Q, A) + (1 - \beta) \cdot lscr^{ir}(Q, A)\,, \quad (12)$$

where $0 \leq \beta \leq 1$ controls the importance we give to the structural l-score compared to the IR l-score.

5. EXPERIMENTS

5.1 Datasets

In information retrieval, TREC[8] is an established methodology for evaluating search algorithms over free text. INEX[9] is a similar effort aimed at information retrieval from XML documents. However, only recently has an evaluation framework been developed by Coffman and Weaver [4] for testing the effectiveness of systems for keyword search over data graphs. The framework consists of three datasets: IMDB, Wikipedia and Mondial. For each data set, there are fifty queries and their assessment.

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. Table 1 describes all the relations of IMDB and Wikipedia, as well as some of the main relations of Mondial.

IMDB and Wikipedia contain relatively large chunks of text, while Mondial has only short strings, such as names of countries and cities, etc. Specifically, the relations that have large amounts of text are *MovieInfo* from IMDB, which contains quotes from movies, and *PageContent* from Wikipedia, which comprises the actual content of pages. On the other hand, the Mondial data graph has a complex structure with a large number of edges, compared to the other two datasets.

Table 1: The three datasets

Dataset	Tuples
IMDB (6 relations)	1,673,074
Movie (id, title, year)	181,706
Person (id, name)	273,034
Character (id, name)	206,951
Role (id, type)	11
MovieInfo (id, movieId, info)	198,678
Cast (id, **movieId, roleId, personId, characterId**)	812,694
Wikipedia (6 relations)	206,318
Page (id, title)	5,540
PageContent (id, text)	5,540
User (id, name)	1,745
Revision (id, **pageId, pageContentId,userid**)	5,540
PageLinks (id, **pageId1, pageId2**)	187,951
UserGroups (id, group)	2
Mondial (24 relations)	17,115
Country (id, name)	195
Province (id, name, **countryId**)	1382
City (id, **countryId, provinceId**)	3,051
Organization (id, name, **countryId, provinceId, cityId**)	168
Language/Economy/Politcs (id, **countryId**)	711
Is_member(id,**countryId,organizationId**)	7,766
Borders(id,**countryId1,countryId1**)	305

5.2 Queries

For each dataset, the benchmark of [4] has fifty queries and their qrels (i.e., query relevance judgments). The average number of keywords per query is 2.91. This average excludes five queries of IMDB that consist of very long citations from movies. The average number of answers per query is 4.49.

For some queries, each answer is just a single tuple, while for others, an answer is a subtree consisting of several tuples and the edges connecting them. For example, the IMDB query #1, which is "denzel washington," should return the tuple of that actor. As another example, for the Wikipedia query #24, which is "lance armstrong tour de france," an answer is a subtree consisting of three tuples. For this particular query, there is just one relevant answer comprising the tuples: Page(id=183266, title="lance armstrong"), PageContent(id=188071, text="tour de france"), and Revision(id=1507622, pageId=183266, pageContentId=188071). The two edges of this answer connect the last tuple with the first and second ones.

5.3 System Implementation

We have translated each dataset into a data graph, where tuples are nodes and foriegn keys (shown in bold in Table 1) are edges. Thus, text is associated only with nodes but not with edges.

Each data graph is indexed into two data structures. First, the skeleton of the graph (i.e., the nodes and edges with their ids only) is built as required by the system [9] that we have used. We refer to this data structure as the *graph index*. The second data structure is an Apache Lucene[10] inverted index,

[7] We denote it by $W(Q, A)$, rather than $W(A)$, because some weights are dynamic, i.e., depend on the query Q.
[8] http://trec.nist.gov/
[9] http://inex.is.informatik.uni-duisburg.de/
[10] http://lucene.apache.org/

where each node appears as a separate document consisting of three fields: content, title and structure, as described in Section 3. We use the Lucene *Document* and *Field* classes to implement those three fields. We have applied stemming and stop-word removal to the three fields of each node.

When given a query $Q = (q_1, \ldots, q_n)$, we need to connect each q_i to the data graph according to the two-step process of Section 4.4. Lucene enables us to do the first step efficiently (without having to implement it from scratch). For the second step, we use the graph index.

After attaching the keywords and assigning the dynamic weights to their edges, as describe in Section 4.4, we need to generate the top-k subtrees according to Equation (12). However, there is no practical algorithm for doing that. Thus, we apply the algorithm of [9] that produces answers based on the structural weight of Equation (10). As done in the benchmark of [4], we generate k answers (where k is a parameter) and re-rank them by the score of Equation (12). These answers do not include the nodes of the keywords.

5.4 The Setup of the Experiments

We ran our system on the benchmark of [4]. We show the results in the same manner as in [4], and include a comparison with all the systems that they tested. The experimental results for those systems were taken from [4]. The following systems were evaluated in [4]: BANKS [3], DISCOVER [12], Efficient [11], Bidirectional [13], Effective [15], DPBF [7], BLINKS [10], SPARK [16], and CD [5]. Our system is called GraphLM.

There are several parameters that should be learned in our algorithm. Those are the title weight α in Equation (7), the structure weight β in Equation (12), the collection-smoothing parameter λ in Equation (5), the parameter K that controls the number of nodes retrieved for each keyword of the query, and the distance parameter r. (The last two parameters are discussed in Sections 4.4.) Section 5.6 describes how we have empirically learned the first two parameters α and β. For λ we have selected the common value 0.1 [19]. Learning the optimal values of K and r is left for future work. We have selected $K = 10,000$ and $r = 4$. Finally, as in [4], our system produced $k = 1,000$ answers for each query.

We use the same four IR metrics as in [4], namely, *MAP* (mean average precision), *top*-1 (which counts how many queries returned the correct top result), *MRR* (mean reciprocal rank) and *11-point interpolated precision*. To compute the mean reciprocal rank, we take for each query, the inverse of the rank of the first correct answer (i.e., $1/rank$), and then average over all queries. The 11-point interpolated precision computes the precision at 11 recall points (from 0.0 to 1.0), and it is the average over all queries.

5.5 GraphLM Compared to State-of-the-Art

Figure 1 shows the MAP of GraphLM in comparison with all the other systems. For the GraphLM runs, we have used $\alpha = 0.7$ and $\beta = 0.8$ which are the optimal values for all of the three collections (see Section 5.6).

GraphLM outperforms all the other systems on each of the three datasets. On IMDB, it gets MAP=0.623, while for BANKS [3] (the second-best system) the MAP is 0.5. This is an improvement of about 25%. On Mondial, GraphLM has MAP=0.828 compared with MAP=0.82 for DPBF [7] (the second-best system). On Wikipedia, GraphLM gets MAP=0.626, whereas CD [5] (the second-best system) has MAP=0.57. The fact that our system is the best on all three datasets is a testament to its robustness.

Overall, the MAP values for Mondial are much higher than for the other two datasets. This is quite expected, because Mondial does not have a lot of free text attached to its data graph.

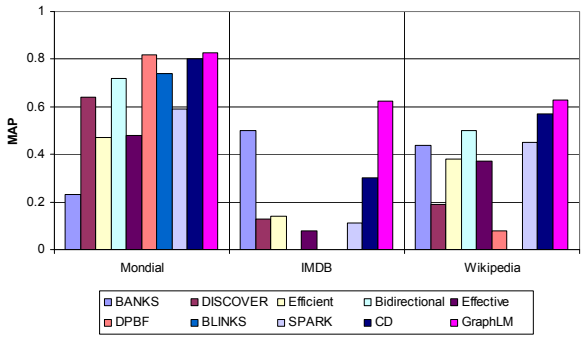


Figure 1: MAP of GraphLM compared with state-of-the-art systems ($\alpha = 0.7$, $\beta = 0.8$)

Figure 2 shows the MRR of all the systems, for the three datasets, when running a subset of the queries where exactly one database tuple is relevant (20, 20, and 15 queries for the Mondial, IMDB and Wikipedia datasets respectively).

On IMDB, GraphLM outperforms all the other systems, having MRR = 0.925 compared with 0.86 for BANKS [3], which is the second. GraphLM is also the best on Wikipedia, having MRR = 1.0 compared with 0.94 for CD [5], which is the second. On Mondial, GraphLM is the second with MRR = 0.891, whereas Bidirectional [13] is the best with MRR = 0.97. Once again, these results confirm the robustness of our system.

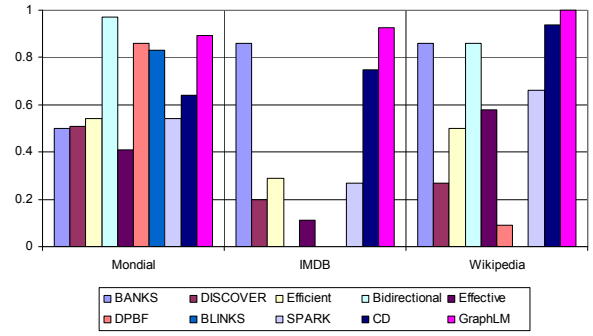


Figure 2: Mean reciprocal rank of GraphLM compared with state-of-the-art systems for the queries where exactly one tuple is relevant ($\alpha = 0.7$, $\beta = 0.8$)

Table 2 shows the top-1 and MRR for the fifty queries of Mondial. Our system achieves top-1 = 37 and MRR = 0.8 compared with 37 and 0.823, respectively, for DPBF [7]. The fact that our system has the second highest MRR on Mondial is consist with the observation that on this dataset, its MAP is just slightly higher than the best state-of-the-art. The values of top-1 and MRR are not given in [4] for the other two datasets. Our values are shown in Table 3.

Figure 3 shows the MAP for the three datasets as a function of k, which is the number of answers returned by the

Table 2: Top-1 and reciprocal rank for Mondial

	top-1	MRR
BANKS	16	0.358
DISCOVER	31	0.671
Efficient	21	0.514
Bidirectional	34	0.730
Effective	22	0.495
DPBF	37	0.823
BLINKS	36	0.770
SPARK	27	0.607
CD	36	0.804
GraphLM	37	0.800

Table 3: Top-1 and reciprocal rank of GraphLM for all the three datasets

	top-1	MRR
Mondial	37	0.800
IMDB	28	0.615
Wikipedia	32	0.716

system. We see that the top-50 answers have almost the same MAP as the top-1,000. Even for the top-20, the MAP is close to that of the top-1,000. These results are due to the fact that many queries of the benchmark have only a small number of correct answers. When $k \leq 10$, the largest drop of the MAP is for Mondial, because this dataset has the largest average number (5.9) of answers per query. The results of Figure 3 indicate that our system could perform well as a fact-retrieval engine.

Figure 4 shows the 11-point interpolated precision of all the systems for a subset of 10 Trec-style queries (#36–#45) of Wikipedia. GraphLM has a MAP of 0.276 for these 10 queries compared with 0.518 for Efficient [11], which is the best in this test. Although the MAP of our system is low, for the top-1 metric, we get 6 out of 10. (The top-1 results of this test are not reported in [4].) The cause of the poor performance of our system is probably the AND semantics, which we use. The qrels of the 10 Trec-style queries have answers that do not contain all the keywords. For example, query #36 is "soviet casualties world war ii" and its qrel includes the tuple "world war ii" of the relation Page, although that tuple does not have the rest of the keywords. The tuple "world war ii" of the relation PageContent is also in the qrel, and so is the subtree comprising these two tuples as well as their connecting tuple of the relation Revision. Since all the keywords appear in the tuple of the relation PageConent, our system returns two out of these three answers. A similar situation occurs with the tuple "Eastern_Front_(World_War_II)" of the relation Page and its corresponding tuples of the relations PageContent and Revision. We plan to extend our system so that it will support the OR semantics.

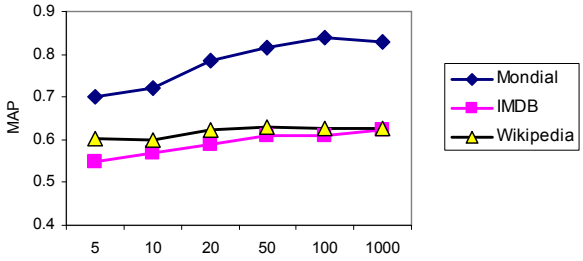


Figure 3: Varying the value of k ($\alpha = 0.7$, $\beta = 0.8$)

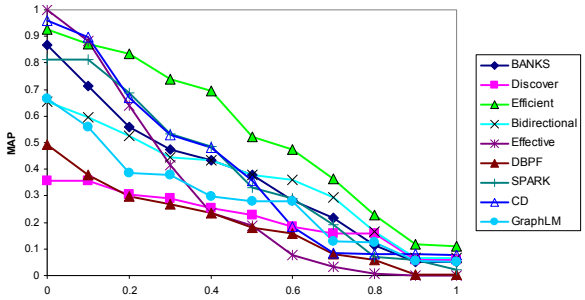


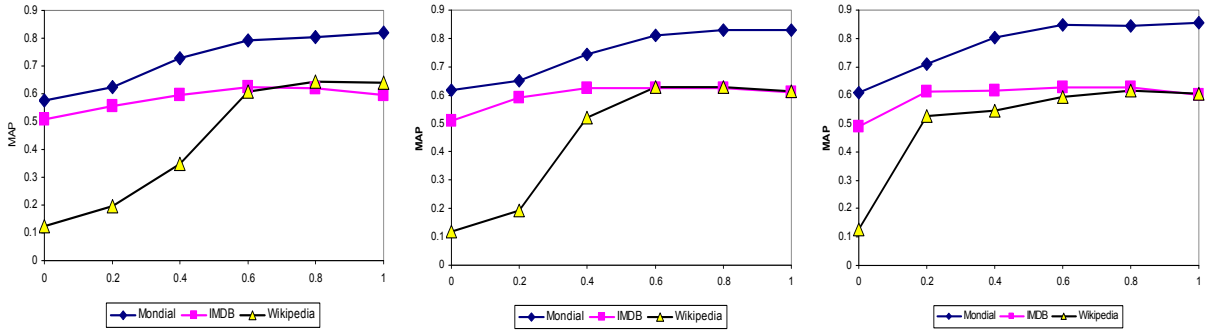
Figure 4: 11-point interpolated precision for the 10 Trec-style queries (#36–#45) of Wikipedia

5.6 Tuning the Parameters

We now describe experiments with different values for the parameters. Our goal is to determine the optimal setup. In particular, we have tried different values for the title weight α of Equation (7) and the structure weight β of Equation (12). Figure 5(a,b,c) shows the MAP when varying β for $\alpha = 0.5$, $\alpha = 0.7$ and $\alpha = 0.9$. We can see that the best MAP for each of the three datasets is achieved for $\beta \geq 0.6$, regardless of the value of α. For $\beta < 0.6$, the MAP starts dropping. Generally, the drop for $\alpha = 0.9$ is gentler whereas the drop for $\alpha = 0.5$ is sharper. Furthermore, we can see that in Wikipedia the drop is very fast, in Mondial there is a moderate drop, and in IMDB the MAP is the stablest across all values of β.

A closer look at the queries of the three datasets can explain this phenomenon. Wikipedia has 15 queries whose only correct answer is a single Page node that contains all the query keywords in its title. In addition, Wikipedia has a large number of nodes of type PageLinks (i.e., they represent hyperlinks), such that the name attribute of their title has the same value as in the Page nodes they link to. Thus, the probability $P(Q|v_{title})$ is the same for both the PageLinks nodes and the Page nodes they point to. But $P(Q|v_{content})$ is a little bit better for the PageLinks nodes than for the Page nodes, because the Page nodes have some more attributes. When β is large, most of the score of the nodes comes from their structural weight, which is a function of the number of incoming edges. Hence, Page nodes have a better score, because they have several incoming edges while PageLinks nodes have none. When β is small, the IR l-score is more dominant. Therefore, if α is also small, PageLinks nodes get a better IR l-score (and, hence, also an overall better score) and they are returned before the Page nodes (even though the latter are the desired answers).

A similar phenomenon exists (albeit less strongly) for Mondial, which has five queries with a single Country node as the only answer. Some of those countries have a province with the same name. In addition, a Province node has an attribute with the name of its country. Hence, when a keyword is a country name and there is also a province with that name, $P(Q|v_{content})$ is better for the Province node than for the Country node. So, when β and α are small, the Province



(a) Title weight $\alpha = 0.5$ (b) Title weight $\alpha = 0.7$ (c) Title weight $\alpha = 0.9$

Figure 5: Varying the structure weight β for different values of the title weight α

node is returned before the Country node (even though the latter is the desired answer). When β is large, the Country node is returned first, because it has more incoming edges than the Province node, so its structural l-score is better.

In IMDB, there are fewer nodes with the above characteristics and, so, it is less sensitive to changes in β and α.

Interestingly, for each of the three datasets, the MAP is almost unchanged when β ranges between 0.6 and 1.0. For example, consider $\beta = 0.8$ and $\beta = 1.0$ in Figure 5(b), where $\alpha = 0.7$. The MAP for IMDB is 0.623 and 0.609 for these two values of β, respectively. For Wikipedia, the MAP is 0.626 and 0.613, respectively. For Mondial, the MAP is 0.828 and 0.83, respectively. When $\beta = 1.0$, the final ranking by Equation (12) depends only on the structural score, whereas the IR l-score is ignored. However, the language models still determine the dynamic weights (of the edges connecting query keywords to the graph), which are part the structural l-score (see Equations (10) and (11)). It is important to take $\beta < 1.0$ when computing the final score (Equation (12)) if answers have internal[11] nodes that contain some of the query keywords. In the benchmark, there are not many queries with such answers.

Another interesting observation is that for a large β, Mondial has a higher MAP for $\alpha = 0.9$ whereas Wikipedia has a higher MAP for $\alpha = 0.5$. For example, when $\beta = 0.8$, Mondial has a MAP of 0.806 and 0.845 for $\alpha = 0.5$ and $\alpha = 0.9$, respectively, while the values for Wikipedia are 0.643 and 0.617, respectively. This can be explained by the fact that Wikipedia has more than 20 queries that return subtrees with multiple tuples, where some of the keywords appear in the content filed of nodes but not in the title field. For example, there are 10 queries whose relevant answers consist of three tuples: $Page \leftarrow Revision \rightarrow PageContent$. Two such queries are "lance armstrong tour de france" and "european hawfinch." In addition, there are 10 TREC-style queries, such as "soviet casualties world war ii" and "einstein special relativity," that return subtrees with more than one tuple where some of the keywords appear in the content field but in the title field. For those queries, a smaller α is better, because relevant answers have the query keywords in the content field but not in the title field. For example, when $\beta = 0.8$, the MAP of the above 20 queries is 0.323 and 0.194 for $\alpha = 0.5$ and $\alpha = 0.9$, respectively. For Mondial, the content and title fields are almost the same (except for some additional attributes with short values in the content field) and, therefore, a larger α is better.

Figure 6 shows the MAP for $\beta = 0.8$, when varying the value of α in Equation (7). For the three datasets, the MAP drops moderately for smaller values of α. For Mondial, the MAP drops from 0.845 when $\alpha = 0.9$ to 0.74 when $\alpha = 0$ (i.e., only the content field is taken into account). For Wikipedia, the MAP drops from 0.643 when $\alpha = 0.5$ to 0.415 when $\alpha = 0$.

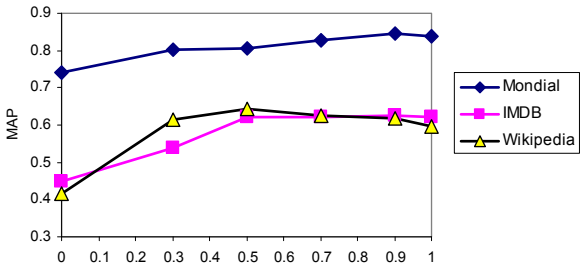


Figure 6: Varying the title weight α for the structure weight $\beta = 0.8$

To summarize, a larger value of β is preferred across all datasets. Regarding α, it depends on whether there are relevant answers where some query keywords appear only in the content (but not in the title) field. If so, a smaller α is preferred; otherwise, a larger α is better. Overall, $\alpha = 0.7$ is a good choice (i.e., close to the optimum). In practice, query logs can be used to determine the optimal value of α. If most queries target entities through their name, a larger α should be used, whereas if most queries target the content, a smaller value of α is better.

5.7 Advanced Options: Query Rewriting

We now describe how to manually rewrite queries so that they better reflect the intended meaning, as defined in the benchmark. This type of rewriting is straightforward and can be easily done by naive users. It utilizes two extensions to pure keyword search over data graphs [1]. The first enables a user to explicitly specify that a keyword should match only the structure by adding ? as a suffix. The second is grouping of several keywords by enclosing them inside curly brackets, which means that they all must appear

[11]Recall that after generating an answer, and before computing its score, we remove the nodes for the query keywords. A node is *internal* if it is not a leaf after that removal.

in the same node or edge. For example, consider again the Wikipedia query #24, which is "lance armstrong tour de france." According to its information need, we formulate the query as "{Page? lance armstrong} {PageContent? tour de france}." In this particular case, our system returns the only relevant result as the top answer.

To do the rewriting, some elementary knowledge about the structure of the data graph is required. The user may obtain that knowledge before she begins to submit queries to the system (e.g., by browsing some help files). Alternatively, the user can derive that knowledge by doing some exploratory search, as enunciated in [17]. Once that knowledge is obtained, the rewriting itself is straightforward.

We now describe how to connect the query keywords to the data graph. For ordinary keywords (i.e., that are neither inside curly brackets nor with ?), we apply the two steps as described in Section 4.4. Next, suppose that the query has a group of keywords inside curly brackets, such that at least one of them is without ? as a suffix.[12] A single node g is created for the whole group (rather than a node for each keyword of that group). We connect g to the top-K nodes that contain all the keywords of the group, but require that keywords with ? as a suffix would appear in the structure field of the connected nodes. The weight of an edge from g to a node v is determined as before using Equation (7), except that it is multiplied by the number of keywords in g that do not have ? as a suffix. We multiply the weight in order to achieve the same effect as when connecting each keyword of the group by a separate edge. For single keywords (i.e., not inside curly brackets) that have ? as a suffix, we apply just the second step that is described in Section 4.4.

We did the rewriting manually, but in strict adherence to the information need of each query, as specified in [4]. The numbers of rewritten queries (out of a total of 50 in each dataset) are: 23 in Mondial, 50 in IMDB and 25 in Wikipedia. Table 4 compares the MAP values before and after the rewriting (for all 50 queries of each dataset).

Table 4: MAP before and after query rewriting for all the three datasets ($\alpha = 0.7$, $\beta = 0.8$)

	Original Queries	After Rewriting
Mondial	0.828	0.856
IMDB	0.623	0.711
Wikipedia	0.626	0.847

There is a significant improvement for all three datasets. Moreover for Wikipedia, the rewriting increased the number of top-1 queries from 32 to 47.

6. CONCLUSIONS AND FUTURE WORK

We presented a novel and effective ranking technique for keyword search over data graphs. Our method combines language models with structural weights. The language models are used twice. First, they determine the dynamic weights that are assigned to the edges that connect query keywords to the data graph. Second, they are a part of the scoring function used in the final ranking of answers. The other part of that function incorporates the structural weights that encode the semantic strength of nodes and edges. We showed empirically how to tune the parameters of our system so as to achieve an optimal effectiveness over different types of data graphs and various types of queries.

We compared our approach with other systems on a standard benchmark of three datasets: IMDB, Wikipedia and Mondial. In terms of MAP, our system is the best on each of these datasets. For example on IMDB, it achieved an improvement of about 25% in the MAP compared with the second-best system. We further showed that by a simple rewriting of the queries, we can improve the performance of our system. For example, after query rewriting of some of the Wikipedia queries, we achieved an improvement of more than 35% in the MAP, and the top-1 value was increased from 32 to 47.

We conclude that systems for keyword search can also be used as effective fact-retrieval engines. However, there could still be limitations in interpreting correctly the user needs. Hence, a minimal user intervention may be needed in some cases, in order to get better results.

For future work, we plan to study how language models can also take the graph structure of an answer into account, and to incorporate support for disjunctive queries.

[12]We do not allow groups in which all the keywords have ? as a suffix.

7. REFERENCES

[1] H. Achiezra, K. Golenberg, B. Kimelfeld, and Y. Sagiv. Exploratory keyword search on data graphs. In *SIGMOD*, 2010.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[4] J. Coffman and A. C. Weaver. A framework for evaluating database keyword search strategies. In *CIKM*, 2010.

[5] J. Coffman and A. C. Weaver. Structured data retrieval using cover density ranking. In *KEYS*, pages 115–126, 2010.

[6] E. Demidova, X. Zhou, and W. Nejdl. Iqp: Incremental query construction, a probabilistic approach. In *ICDE*, 2010.

[7] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[8] S. Elbassuoni1, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum1. Language-model-based ranking for queries on rdf-graphs. In *CIKM*, 2009.

[9] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, 2008.

[10] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: Ranked keyword searches on graphs. In *SIGMOD*, 2007.

[11] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[12] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, 2002.

[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] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar semantic search: a database approach to information retrieval. In *SIGMOD*, 2006.

[15] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[16] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: top-k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.

[17] Y. Mass, M. Ramanath, Y. Sagiv, and G. Weikum. Iq: The case for iterative querying for knowledge. In *CIDR*, 2011.

[18] J. Pound, I. F. Ilyas, and G. Weddell. Expressive and flexible access to web-extracted data: A keyword-based structured query languag. In *SIGMOD*, 2010.

[19] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.